PICmicro® 18C MCU Family Reference Manual

Trademarks

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

PAGE

NOTES:

Company Profile

The Embedded Control Solutions Company®

Since its inception, Microchip Technology has focused its resources on delivering innovative semiconductor products to the global embedded control marketplace. To do this, we have focused our technology, engineering, manufacturing and marketing resources on synergistic product lines: PICmicro® microcontrollers (MCUs), high-endurance Serial EEPROMs, an expanding product portfolio of analog/interface products, RFID tags and KEELOQ® security devices – all aimed at delivering comprehensive, high-value embedded control solutions to a growing base of customers.

Inside Microchip Technology you will find:

- An experienced executive team focused on innovation and committed to listening to our customers
- A focus on providing high-performance, cost-effective embedded control solutions
- Fully integrated manufacturing capabilities
- A global network of manufacturing and customer support facilities
- A unique corporate culture dedicated to continuous improvement
- Distributor network support worldwide including certified distribution FAEs
- A Complete Product Solution including:
  - RISC OTP, FLASH, EEPROM and ROM MCUs
  - A full family of advanced analog MCUs
  - KEELOQ security devices featuring patented code hopping technology
  - Stand-alone analog and interface products plus microID™ RFID tagging devices
  - A complete line of high-endurance Serial EEPROMs
  - World-class, easy-to-use development tools
  - An Automotive Products Group to engage with key automotive accounts and provide necessary application expertise and customer service

Business Scope

Microchip Technology Inc. designs, manufactures, and markets a variety of CMOS semiconductor components to support the market for cost-effective embedded control solutions.

Microchip's products feature compact size, integrated functionality, ease of development and technical support so essential to timely and cost-effective product development by our customers.

Chandler, Arizona: Company headquarters near Phoenix, Arizona; executive offices, R&D and wafer fabrication occupy this 242,000 square-foot multi-building campus.

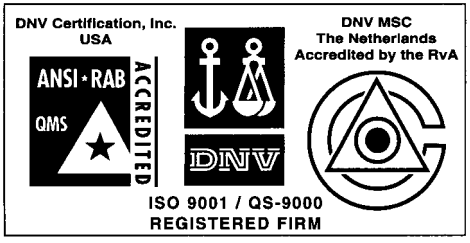
Tempe, Arizona: Microchip's 200,000 square-foot wafer fabrication facility provides increased manufacturing capacity today and for the future.

Market Focus

Microchip targets select markets where our advanced designs, progressive process technology and industry-leading product performance enables us to deliver decidedly superior performance. Our Company is positioned to provide a complete product solution for embedded control applications found throughout the consumer, automotive, telecommunication, office automation and industrial control markets. Microchip products are also meeting the unique design requirements of targeted embedded applications including internet, safety and security.

Certified Quality Systems

Microchip's quality systems have been certified to QS-9000 requirements. Its worldwide headquarters and wafer fabrication facilities in Chandler and Tempe, Arizona, received certification on July 23, 1999. The scope of this certification is the design and manufacture of RISC-based MCUs, related non-volatile memory products and microperipheral devices. The quality systems for Microchip's product test facility in Bangkok, Thailand, were QS-9000 certified on February 26, 1999. The scope of this certification is the design and testing of integrated circuits. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.



QS-9000 was developed by Chrysler, Ford and General Motors to establish fundamental quality systems that provide for continuous improvement, emphasizing defect prevention and the reduction of variation and waste in the supply chain. Microchip was audited by QS-9000 registrar Det Norske Veritas Certification Inc. of Houston, the same firm which granted Microchip its ISO 9001 Quality System certification in 1997. QS-9000 certification recognizes Microchip's quality systems conform to the stringent standards set forth by the automotive industry, benefiting all customers.

Fully Integrated Manufacturing

Microchip delivers fast turnaround and consistent quality through total control over all phases of production. Research and development, design, mask making, wafer fabrication, and the major part of assembly and quality assurance testing are conducted at facilities wholly-owned and operated by Microchip. Our integrated approach to manufacturing along with rigorous use of advanced Statistical Process Control (SPC) and a continuous improvement culture has resulted in high and consistent yields which have positioned Microchip as a quality leader in its global markets. Microchip's unique approach to SPC provides customers with excellent pricing, quality, reliability and on-time delivery.

Bangkok, Thailand: Microchip's 200,000 square-foot manufacturing facility houses the technology and assembly/test equipment for high speed testing and packaging.

A Global Network of Plants and Facilities

Microchip is a global competitor providing local services to the world's technology centers. The Company's design and technology advancement facilities, and wafer fabrication sites are located in Chandler and Tempe, Arizona.

The Tempe facility provides an additional 200,000 square feet of manufacturing space that meets the increased production requirements of a growing customer base, and provides production capacity which more than doubles that of Chandler.

Microchip facilities in Bangkok, Thailand, and Shanghai, China, serve as the foundation of Microchip's extensive assembly and test capability located throughout Asia. The use of multiple fabrication, assembly and test sites, with more than 640,000-square-feet of facilities worldwide, ensures Microchip's ability to meet the increased production requirements of a fast growing customer base.

Microchip supports its global customer base from direct sales and engineering offices in Asia, North America, Europe and Japan. Offices are staffed to meet the high quality expectations of our customers, and can be accessed for technical and business support. The Company also franchises more than 60 distributors and a network of technical manufacturer's representatives serving 24 countries worldwide.

Embedded Control Overview

Unlike “processor” applications such as personal computers and workstations, the computing or controlling elements of embedded control applications are embedded inside the application. The consumer is only concerned with the very top-level user interface such as keypads, displays and high-level commands. Very rarely does an end-user know (or care to know) the embedded controller inside (unlike the conscientious PC users, who are intimately familiar not only with the processor type, but also its clock speed, DMA capabilities and so on).

It is, however, most vital for designers of embedded control products to select the most suitable controller and companion devices. Embedded control products are found in all market segments: consumer, commercial, PC peripherals, telecommunications, automotive and industrial. Most embedded control products must meet special requirements: cost effectiveness, low-power, small-footprint and a high level of system integration.

Typically, most embedded control systems are designed around an MCU which integrates on-chip program memory, data memory (RAM) and various peripheral functions, such as timers and serial communication. In addition, these systems usually require complementary Serial EEPROM, analog/interface devices, display drivers, keypads or small displays.

Microchip has established itself as a leading supplier of embedded control solutions. The combination of high-performance PIC12CXXX, PIC16C5X, PIC16CXXX, PIC17CXXX and PIC18CXXX MCU families with Migratable Memory™ technology, along with non-volatile memory products, provide the basis for this leadership. By further expanding our product portfolio to provide precision analog and interface products, Microchip is committed to continuous innovation and improvement in design, manufacturing and technical support to provide the best possible embedded control solutions to you.

PICmicro MCU Overview and Roadmap

Microchip PICmicro MCUs combine high-performance, low-cost, and small package size, offering the best price/performance ratio in the industry. More than one billion of these devices have shipped to customers worldwide since 1990. Microchip offers five families of MCUs to best fit your application needs:

- PIC12CXXX 8-pin 12-bit/14-bit program word
- PIC16C5X 12-bit program word
- PIC16CXXX 14-bit program word
- PIC17CXXX 16-bit program word
- PIC18CXXX enhanced 16-bit program word

All families offer OTP, low-voltage and low-power options, with a variety of package options. Selected members are available in ROM, EEPROM or reprogrammable FLASH versions.

PIC12CXXX: 8-Pin, Family

The PIC12CXXX family packs Microchip’s powerful RISC-based PICmicro architecture into 8-pin DIP and SOIC packages. These PIC12CXXX products are available with either a 12-bit or 14-bit wide instruction set, a low operating voltage of 2.5V, small package footprints, interrupt handling, a deeper hardware stack, multiple channels and EEPROM data memory. All of these features provide an intelligence level not previously available in applications because of cost or size considerations.

PIC16C5X: 12-Bit Architecture Family

The PIC16C5X is the well-established base-line family that offers the most cost-effective solution. These PIC16C5X products have a 12-bit wide instruction set and are currently offered in 14-, 18-, 20- and 28-pin packages. In the SOIC and SSOP packaging options, these devices are among the smallest footprint MCUs in the industry. Low-voltage operation, down to 2.0V for OTP MCUs, makes this family ideal for battery operated applications. Additionally, the PIC16HV5XX can operate up to 15 volts for use directly with a battery.

PIC16CXXX: 14-Bit Architecture Family

With the introduction of new PIC16CXXX family members, Microchip now provides the industry’s highest performance Analog-to-Digital Converter capability at 12-bits for an MCU. The PIC16CXXX family offers a wide-range of options, from 18- to 68-pin packages as well as low to high levels of peripheral integration. This family has a 14-bit wide instruction set, interrupt handling capability and a deep, 8-level hardware stack. The PIC16CXXX family provides the performance and versatility to meet the more demanding requirements of today’s cost-sensitive marketplace for mid-range applications.

PIC17CXXX: 16-Bit Architecture Family

The PIC17CXXX family offers the world’s fastest execution performance of any MCU family in the industry. The PIC17CXXX family extends the PICmicro MCU’s high-performance RISC architecture with a 16-bit instruction word, enhanced instruction set and powerful vectored interrupt handling capabilities. A powerful array of precise on-chip peripheral features provides the performance for the most demanding applications.

PIC18CXXX: 16-Bit Enhanced Architecture Family

The PIC18CXXX is a family of high performance, CMOS, fully static, 16-bit MCUs with integrated analog-to-digital (A/D) converter. All PIC18CXXX MCUs incorporate an advanced RISC architecture. The PIC18CXXX has enhanced core features, 32 level-deep stack, and multiple internal and external interrupts sources. The separate instruction and data busses of the Harvard architecture allow a 16-bit wide instruction word with the separate 8-bit wide data. The two-stage instruction pipeline allows all instructions to execute in a single cycle, except for program branches, which require two cycles. A total of 77 instructions (reduced instruction set) are available. Additionally, a large register set gives some of the architectural

innovations used to achieve a very high performance of 10 MIPS for an MCU. The PIC18CXXX family has special features to reduce external components, thus reducing cost, enhancing system reliability and reducing power consumption. These include programmable Low Voltage Detect (LVD) and programmable Brown-Out Detect (BOD).

The Mechatronics Revolution

The nature of the revolution is the momentous shift from analog/electro-mechanical timing and control to digital electronics. It is called the Mechatronics Revolution, and it is being staged in companies throughout the world, with design engineers right on the front lines: make it smarter, make it smaller, make it do more, make it cost less to manufacture – and make it snappy.

To meet the needs of this growing customer base, Microchip is rapidly expanding its already broad line of PICmicro MCUs. The PIC12CXXX family's size opens up new possibilities for product design.

PICmicro MCU Naming Convention

The PICmicro architecture offers users a wider range of cost/performance options than any MCU family. In order to identify the families, the following naming conventions have been applied to the PICmicro MCUs:

TABLE 1: PICmicro MCU NAMING CONVENTION*

Family		Architectural Features	Name	Technology
PIC18CXXX	8-bit High-Performance MCU Family	• 10 MIPS @ 40 MHz • 4x PLL clock • 16-bit wide instruction set • C compiler efficient instruction set • Internal/external vectored interrupts	PIC18CXX2 PIC18FXXX	OTP program memory with higher resolution analog functions FLASH program memory
PIC17CXXX	8-bit High-Performance MCU Family	• 16-bit wide instruction set • Internal/external vectored interrupts • DC - 33 MHz clock speed • 120 ns instruction cycle (@ 33 MHz) • Hardware multiply	PIC17C4X	OTP program memory, digital only
			PIC17CR4X	ROM program memory, digital only
			PIC17C7XX	OTP program memory with mixed-signal functions
PIC16CXXX	8-bit Mid-Range MCU Family	• 14-bit wide instruction set • Internal/external interrupts • DC - 20 MHz clock speed (Note 1) • 200 ns instruction cycle (@ 20 MHz)	PIC14CXXX	OTP program memory with A/D and D/A functions
			PIC16C55X	OTP program memory, digital only
			PIC16C6X	OTP program memory, digital only
			PIC16CR6X	ROM program memory, digital only
			PIC16C62X	OTP program memory with comparators
			PIC16CR62X	ROM program memory with comparators
			PIC16CE62X	OTP program memory with comparators and EEPROM data memory
			PIC16F62X	FLASH program memory with comparators and EEPROM data memory
			PIC16C64X	OTP program memory with comparators
			PIC16C66X	OTP program memory with comparators
			PIC16C7X	OTP program memory with analog functions (i.e. A/D)
			PIC16CR7X	ROM program memory with analog functions
			PIC16C7XX	OTP program memory with higher resolution analog functions
			PIC16F8X	FLASH program memory and EEPROM data memory
			PIC16CR8X	ROM program memory and EEPROM data memory
			PIC16F87X	FLASH program memory with higher resolution analog functions
			PIC16C9XX	OTP program memory, LCD driver
PIC16C5X	8-bit Base-Line MCU Family	• 12-bit wide instruction set • DC - 20 MHz clock speed • 200 ns instruction cycle (@ 20 MHz)	PIC16C5X	OTP program memory, digital only
			PIC16CR5X	ROM program memory, digital only
			PIC16C505	OTP program memory, digital only, internal 4 MHz oscillator
			PIC16HV540	OTP program memory with high voltage operation
PIC12CXXX	8-bit, 8-pin MCU Family	• 12- or 14-bit wide instruction set • DC - 10 MHz clock speed • 400 ns instruction cycle (@ 10 MHz) • Internal 4 MHz oscillator	PIC12C5XX	OTP program memory, digital only
			PIC12CE5XX	OTP program memory, digital only with EEPROM data memory
			PIC12CR5XX	ROM program memory, digital only
			PIC12C67X	OTP program memory with analog functions
			PIC12CE67X	OTP program memory with analog functions and EEPROM data memory

Note 1: The maximum clock speed for some devices is less than 20 MHz.

*Please check with your local Microchip distributor, sales representative or sales office for the latest product information.

Development Systems

Microchip is committed to providing useful and innovative solutions to your embedded system designs. Our installed base of application development systems has grown to an impressive 170,000 systems worldwide.

Among support products offered are MPLAB®-ICE 2000 In-Circuit Emulator running under the Windows® environment. This new real-time emulator supports low-voltage emulation, to 2.0 volts, and full-speed emulation. MPLAB, a complete Integrated Development Environment (IDE), is provided with MPLAB-ICE 2000. MPLAB allows the user to edit, compile and emulate from a single user interface, making the developer productive very quickly. MPLAB-ICE 2000 is designed to provide product development engineers with an optimized design tool for developing target applications. This universal in-circuit emulator provides a complete MCU design toolset for PICmicro MCUs in the PIC12CXXX, PIC16C5X, PIC16CXXX, PIC17CXXX and PIC18CXXX families. MPLAB-ICE 2000 is CE compliant.

Microchip's newest development tool, MPLAB In-Circuit Debugger (ICD) Evaluation Kit, uses the in-circuit debugging capabilities of the PIC16FXXX and PIC18FXXX MCU family and Microchip's ICSP™ capability to debug source code in the application, debug hardware in real time and program a target PIC16FXXX and PIC18FXXX device.

PRO MATE® II, the full-featured, modular device programmer, enables you to quickly and easily program user software into PICmicro MCUs, HCS products and Serial EEPROMs. PRO MATE II runs under MPLAB IDE and operates as a stand-alone unit or in conjunction with a PC-compatible host system.

The PICSTART® Plus development kit is a low-cost development system for the PIC12CXXX, PIC16C5X, PIC16CXXX and PIC17CXXX MCUs.

PICDEM low-cost demonstration boards are simple boards which demonstrate the basic capabilities of the full range of Microchip's MCUs. Users can program the sample MCUs provided with PICDEM boards, on a PRO MATE II or PICSTART Plus programmer, and easily test firmware. KEELOQ Evaluation Tools support Microchip's HCS Secure Data Products.

The Serial EEPROM Designer's Kit includes everything necessary to read, write, erase or program special features of any Microchip Serial EEPROMs. The *Total Endurance*™ Disk is included to aid in trade-off analysis and reliability calculations. The total kit can significantly reduce time-to-market and result in an optimized system.

The FilterLab™ Active Filter Design Tool simplifies active filter design for embedded systems designers. The unique FilterLab software automates the design of the anti-aliasing filter for an analog-to-digital converter-based data acquisition system. FilterLab also provides full schematic diagrams of the filter circuit with component values, a SPICE model, and displays the frequency and phase response.

In addition to the FilterLab Active Filter Design Tool, Microchip offers a second analog development tool, the MXDEV™1 Analog Evaluation System, making it easier for embedded systems designers to evaluate and develop with Microchip's line of stand-alone analog products. The hardware and software within the MXDEV 1 system is configured device-specific and allows single or continuous conversions ofr the analog-to-digital converter under evaluation.

The MCP2510 Controller Area Network (CAN) Developer's Kit makes software developing easy by using a variety of features to manipulate the functionality of the MCP2510. The MCP2510 CAN Developer's kit provides the ability to read, display and modify all registers of the MCP2510 on a bit-by-bit or a byte-by-byte basis.

The microID™ Developer's Kit is an easy-to-use tool for design engineers at all skill levels. Available in a variety of configurations, the microID family of RFID tags can be configured to match existing tags and be directly installed - upgrading to contactless programmability at no added cost. This kit includes all the hardware, software, reference designs and samples required to get started in RFID designs.

TABLE 2: PICmicro SYNERGISTIC DEVELOPMENT TOOLS

Development Tool	Name	PIC12CXXX	PIC16C5X	PIC16CXXX	PIC16F87X	PIC17CXXX	PIC18CXXX
Integrated Development Environment (IDE)	MPLAB™	✔	✔	✔	—	✔	✔
C Compiler	MPLAB-C17	—	—	—	—	✔	
	MPLAB-C18	—	—	—	—	—	✔
Full-Featured, Modular In-Circuit Emulator	MPLAB-ICE 2000	✔	✔	✔	—	✔	✔
In-Circuit Debugger Evaluation Kit	MPLAB-ICD	—	—	—	✔	—	—
Full-Featured, Modular Device Programmer	PRO MATE® II	✔	✔	✔	—	✔	✔
Entry-Level Development Kit with Programmer	PICSTART® Plus	✔	✔	✔	—	✔	✔

Software Support

MPLAB Integrated Development Environment (IDE) is a Windows-based development platform for Microchip's PICmicro MCUs. MPLAB IDE offers a project manager and program text editor, a user-configurable toolbar containing four pre-defined sets and a status bar which communicates editing and debugging information.

MPLAB-IDE is the common user interface for Microchip development systems tools including MPLAB Editor, MPASM Assembler, MPLAB-SIM Software Simulator, MPLIB, MPLINK, MPLAB-C17 Compiler, MPLAB-C18 Compiler, MPLAB-ICE 2000, PRO MATE II Programmer and PICSTART Plus Development Programmer.

Microchip endeavors at all times to provide the best service and responsiveness possible to its customers. The Microchip Internet site can provide you with the latest technical information, production released software for development tools, application notes and promotional news on Microchip products and technology. The Microchip World Wide Web address is http://www.microchip.com.

Secure Data Products Overview

Microchip's patented KEELOQ® code hopping technology is the perfect solution for remote keyless entry and logical/physical access control systems. The initial device in the family, the HCS300 encoder, replaces current fixed code encoders in transmitter applications providing a low cost, integrated solution. The KEELOQ family is continuing to expand with the HCS301 (high voltage encoder), HCS200 (low-end, low-cost encoder), and high-end encoders (HCS360 and HCS361) that meet OEM specifications and requirements. The HCS410, a self-powered transponder superset of the HCS360, is the initial device in a new and expanding encoder/transponder family.

Microchip provides flexible decoder solutions by providing optimized routines for Microchip's PICmicro MCUs. This allows the designer to combine the decoder and system functionality in a MCU. The decoder routines are available under a license agreement. The HCS500, HCS512 and HCS515 are the first decoder devices in the KEELOQ family. These devices are single chip decoder solutions and simplify designs by handling learning and decoding of transmitters.

The KEELOQ product family is expanding to include enhanced encoders and decoders. Typical applications include automotive RKE, alarm and immobilizer systems, garage door openers and home security systems.

KEELOQ® Rock Solid Security

KEELOQ Encoder Devices					
Product	**Transmission Code Length Bits**	**Code Hopping Bits**	**Prog. Encryption Key Bits**	**Seed Length**	**Operating Voltage**
HCS101*	66	—	—	—	3.5V to 13.0V
HCS200	66	32	64	32	3.5V to 13.0V
HCS201*	66	32	64	32	3.5V to 13.0V
HCS300	66	32	64	32	2.0V to 6.3V
HCS301	66	32	64	32	3.5V to 13.0V
HCS320	66	32	64	32	3.5V to 13.0V
HCS360	67	32	64	48	2.0V to 6.6V
HCS361	67	32	64	48	2.0V to 6.6V
HCS365*	69	32	2 x 64	60	2.0V to 6.6V
HCS370*	69	32	2 x 64	60	2.0V to 6.6V
HCS410	69	32	64	60	2.0V to 6.6V
HCS412*	69	32	64	60	2.0V to 6.6V
HCS470*	69	32	2 x 64	60	2.0V to 6.6V
KEELOQ Decoder Devices					
Product	**Reception Length Bits**	**Transmitters Supported**	**Functions**		**Operating Voltage**
HCS500	67	Up to 7	15 Serial Functions		4.5V to 5.5V
HCS512	67	Up to 4	15 (S0, S1, S2, S3); VLOW, Serial		3.0V to 6.0V
HCS515	67	Up to 7	15 Serial; 3 Parallel		4.5V to 5.5V

*Contact Microchip Technology Inc. for availability.

Analog/Interface Products

Using its technology achievements in developing analog circuitry for its PICmicro MCU family, the Company launched a complementary line of stand-alone analog and interface products. Many of these stand-alone devices support functionality that may not currently available on PICmicro MCUs. Stand-alone analog IC products currently offered include:

- Analog-to-Digital Converters
- Operational Amplifiers
- System Supervisors

Microchip also offers innovative silicon products to support a variety of bus interfaces used to transmit data to and from embedded control systems. The first interface products support Controller Area Network (CAN), a bus protocol highly integrated into a variety of networked applications including automotive.

High-Performance 12-Bit Analog-to-Digital Converters

The MCP320X 12-bit analog-to-digital converter (ADC) family is based on a successive approximation register architecture. The first four members include: MCP3201, MCP3202, MCP3204 and MCP3208. The MCP320X family features 100K samples per second throughput, low power of 400 microamps active and 500 nanoamps standby, wide supply voltage of 2.7-5.5 volts, extended industrial temperature range of –40° to 85°, +/- 1 LSB DNL and +/- 1 LSB INL max. at 100 ksps., no missing codes, and a serial output with an industry-standard SPI™ bus interface. The MCP320X is available in 1-, 2-, 4-, and 8-input channel versions (the MCP3201, MPC3202, MCP3204 and MCP3208, respectively). The devices

are offered in PDIP, SOIC and TSSOP packages. Applications include data acquisition, instrumentation and measurement, multi-channel data loggers, industrial PCs, motor control, robotics, industrial automation, smart sensors, portable instrumentation, and home medical appliances.

Operational Amplifiers

The MCP60X Operational Amplifier family includes four devices: MCP601, MCP602, MCP603 and MCP604. These devices are Microchip's first 2.7 volt single supply operational amplifier products. The MCP60X family offers a gain bandwidth product of 2.8 MHz with low typical operating current of 230 µA. The MCP60X devices use Microchip's advanced CMOS technology which provides low bias current, high speed operation, high open-loop gain and rail-to-rail output swing.

System Supervisors

Microchip offers a complete family of system supervisor products. The new devices include the MCP809/810 and MCP100/101 supervisory circuits with push-pull output and the MCP120/130 supervisory circuits with open drain output. The devices are functionally and pin-out comparable to products from other analog suppliers.

Controller Area Network (CAN)

Microchip is enhancing its product portfolio by introducing the CAN Product Family. The MCP2510 is the smallest, easiest-to-use, CAN controller on the market today. Combining the MCP2510 with Microchip's broad range of high-performance PICmicro MCUs enables Microchip to support for virtually all of today's CAN-based applications. Other potential benefits of having a separate CAN controller include the ability for system designers to select from a much wider variety of MCUs for an optimal performance solution.

Additional products planned for Microchip's CAN product portfolio include other CAN peripherals and a family of PICmicro MCUs with integrated CAN support.

microID™ RFID Tagging Devices

Only Microchip manufactures world-class components for every application in the radio frequency identification (RFID) system. From the advanced, feature-packed microID family of RFID tags and high-endurance Serial EEPROMs to high performance PICmicro MCUs and KEELOQ code hopping encoders - Microchip's full range of RFID solutions are available for your tag, peripheral and reader application designs.

The microID family can emulate almost any standard on the market today. It provides drop-in compatible solutions to the most commonly used 125 kHz and 13.56 MHz tags and an upgrade migration path for virtually any application with higher performance and new features.

Serial EEPROM Overview

Microchip's high-endurance Serial EEPROMs complement the diverse MCU product families. Serial EEPROMs are available in a variety of densities, operating voltages, bus interface protocols, operating temperature ranges and space-saving packages.

Densities:

The densities range from 128 bits to 256 Kbits with higher density devices in development.

Bus Interface Protocols:

We offer all popular protocols: I^2C™, Microwire® and SPI.

Operating Voltages:

In addition to standard 5V devices there are two low voltage families. The "LC" devices operate down to 2.5V, while the breakthrough "AA" family operates, in both read and write mode, down to 1.8V, making these devices highly suitable for alkaline and NiCd battery powered applications.

Temperature Ranges:

Like all Microchip devices, many Serial EEPROMs are offered in Commercial (0°C to +70°C), Industrial (-40°C to +85°C) and Extended (-40°C to +125°C) operating temperature ranges.

Packages:

Small footprint packages include: industry standard 5-lead SOT-23, 8-lead DIP, 8-lead SOIC in JEDEC and EIAJ body widths, and 14-lead SOIC. The SOIC comes in two body widths; 150 mil and 207 mil.

Technology Leadership:

Selected Microchip Serial EEPROMs are backed by a 1 million Erase/Write cycle. Microchip's erase/write cycle endurance is among the best in the world, and only Microchip offers such unique and powerful development tools as the Total Endurance disk. This mathematical software model is an innovative tool used by system designers to optimize Serial EEPROM performance and reliability within the application.

Microchip offers Plug-and-Play to the DIMM module market with the 24LCS52, a special function single-chip EEPROM that is available in space saving packages. For Plug-and-Play video monitor applications, Microchip offers the 24LC21, a single-chip DDC1™/DDC2™-compatible solution. In addition, Microchip released a high-speed 1 MHz 2-wire Serial EEPROM device ideal for high-performance embedded systems.

Microchip is a high-volume supplier of Serial EEPROMs to all the major markets worldwide. The Company continues to develop new Serial EEPROM solutions for embedded control applications.

OTP EPROM Overview

Microchip's CMOS EPROM devices are produced in densities from 64K to 512K. Typical applications include computer peripherals, instrumentation, and automotive devices. Microchip's expertise in surface mount packaging on SOIC and TSOP packages led to the development of the surface mount OTP EPROM market where Microchip is a leading supplier today. Microchip is also a leading supplier of low-voltage EPROMs for battery powered applications.

MIGRATABLE MEMORY™ TECHNOLOGY

Microchip's innovative Migratable Memory technology (MMT) provides socket and software compatibility among all of its equivalent ROM, OTP and FLASH memory MCUs. MMT allows customers to match the selection of MCU memory technology to the product life cycle of their application, providing an easy migration path to a lower cost solution whenever appropriate.

FLASH memory is an ideal solution for engineers designing products for embedded systems – especially during the development and early stages of the product. In certain products and applications, FLASH memory may be used for the life of the product because of the advantages of field upgradability or where product inventory flexibility is required.

Once the design enters the pre-production stage and continues through introduction and growth stages, OTP program memory provides maximum programming flexibility and minimum inventory scrappage. The OTP device is pin and socket compatible with the FLASH device – providing a lower cost, high-volume flexible solution.

As the design enters a mature stage and program code stabilizes, a lower cost, socket compatible ROM memory device could be used. In some cases, OTP memory may still be used as the most cost-effective memory technology for the product. Compatibility and flexibility are key to the success of the PICmicro MCU product family, and ultimately the success of our customers.

FLEXIBLE PROGRAMMING OPTIONS

To meet the stringent design requirements placed on our customers, the following innovative programming options are offered. These programming options address procurement issues by reducing and limiting work-in-process liability and facilitating finished goods code revisions. Microchip's worldwide distributors stock reprogrammable and one-time programmable inventory, allowing customers to respond to immediate sales opportunities or accommodate engineering changes off the shelf.

FLASH (electrically reprogrammable)

PICmicro FLASH MCUs allow erase and reprogramming of the MCU program memory. Reprogrammability offers a highly flexible solution to today's ever-changing market demands – and can substantially reduce time to market. Users can program their systems very late in the manufacturing process or update systems in the field. This allows easy code revisions, system parameterization or customer-specific options with no scrappage. Reprogrammability also reduces the design verification cycle.

One-Time Programmable (OTP)

PICmicro OTP MCUs are manufactured in high volumes without customer specific software and can be shipped immediately for custom programming. This is useful for customers who need rapid time to market and flexibility for frequent software updates.

In-Circuit Serial Programming™ (ICSP™)

Microchip's PICmicro FLASH and OTP MCUs feature ICSP capability. ICSP allows the MCU to be programmed after being placed in a circuit board, offering tremendous flexibility, reduced development time, increased manufacturing efficiency and improved time to market. This popular technology also enables reduced cost of field upgrades, system calibration during manufacturing, the addition of unique identification codes to the system and system calibration. Requiring only two I/O pins for most devices, Microchip offers the most non-intrusive programming methodology in the industry.

Self Programming

Microchip's PIC16F87X family features self programming capability. Self programming enables remote upgrades to the FLASH program memory and the end equipment through a variety of medium ranging from Internet and Modem to RF and Infrared. To setup for self programming, the designer programs a simple boot loader algorithm in a code protected area of the FLASH program memory. Through the selected medium, a secure command allows entry into the PIC16F87X MCU through the USART, I^2C or SPI serial communication ports. The boot loader is then enabled to reprogram the PIC16F87X FLASH program memory with data received over the desired medium. And, of course, self programming is accomplished without the need for external components and without limitations on the PIC16F87X's operating speed or voltage.

Quick-Turn Programming (QTP)

Microchip offers a QTP programming service for factory production orders. This service is ideal for customers who choose not to program a medium to high unit volume in their own factories, and whose production code patterns have stabilized.

Serialized Quick-Turn Programming (SQTP^{SM})

SQTP is a unique, flexible programming option that allows Microchip to program serialized, random or pseudo-random numbers into each device. Serial programming allows each device to have a unique number which can serve as an entry-code, password or ID number.

Masked ROM

Microchip offers Masked ROM versions of many of its most popular PICmicro MCUs, giving customers the lowest cost option for high volume products with stable firmware.

Future Products and Technology

Microchip is constantly developing advanced process technology modules and new products that utilize our advanced manufacturing capabilities. Current production technology utilizes lithography dimensions down to 0.7 micron.

Microchip's research and development activities include exploring new process technologies and products that have industry leadership potential. Particular emphasis is placed on products that can be put to work in high-performance broad-based markets.

Equipment is continually updated to bring the most sophisticated process, CAD and testing tools online. Cycle times for new technology development are continuously reduced by using in-house mask generation, a high-speed pilot line within the manufacturing facility and continuously improving methodologies.

Objective specifications for new products are developed by listening to our customers and by close co-operation with our many customer-partners worldwide.

NOTES:

Section 1. Introduction

HIGHLIGHTS

This section of the manual contains the following major topics:

1.1 Introduction

Microchip is the Embedded Control Solutions Company®. The company's focus is on products that meet the needs of the embedded control market. We are a leading supplier of:

- 8-bit general purpose microcontrollers (PICmicro® MCUs)
- Speciality and standard non-volatile memory devices
- Security devices (KEELOQ®)
- Application specific standard products

Please request a Microchip Product Line Card for a listing of all the interesting products that we have to offer. This literature can be obtained from your local sales office, or downloaded from the Microchip web site (www.microchip.com).

In the past, 8-bit MCU users were fixed on the traditional MCU model for production, a ROM device. Microchip has been the leader in changing this perception by showing that OTP devices can give a better lifetime product cost compared to ROM versions.

Microchip has strength in FLASH and EPROM technology. This makes it the memory technology of choice for the PICmicro MCU's program memory. Microchip has minimized the cost difference between EPROM and ROM memory technology. Therefore, Microchip can pass these benefits on to our customers. This is not true for other MCU vendors, and is seen in the price difference between their FLASH/EPROM and ROM versions.

The growth of Microchip's 8-bit MCU market share is a testament to the PICmicro MCU's ability to meet the needs of many customers. This growth has made the PICmicro architecture one of the top two architectures available in the general market today. This growth was fueled by the Microchip vision of the benefits of a low cost Field Programmable MCU solution. Some of the benefits for the customer include:

- Quick time to market
- Allows code changes to product during production run
- No Non-Recurring Engineering (NRE) charges for Mask Revisions
- Ability to easily serialize the product
- Ability to store calibration data without additional hardware
- Better able to maximize use of PICmicro MCU inventory
- Less risk, since the same device is used for development as well as for production.

Microchip's PICmicro 8-bit MCUs offer a price/performance ratio that allows them to be considered for any traditional 8-bit MCU application, as well as some traditional 4-bit applications (Base-Line family), low-end 16-bit applications (PIC17CXXX and PIC18CXXX families), dedicated logic replacement and low-end DSP applications (High-End and Enhanced families). These features and price-performance mix make PICmicro MCUs an attractive solution for most applications.

1.2 Manual Objective

PICmicro devices are grouped by the size of their Instruction Word and their Instruction Set. The four current PICmicro families and their Instruction Word length are:

1. Base-Line: 12-bit Instruction Word length
2. Mid-Range: 14-bit Instruction Word length
3. High-End: 16-bit Instruction Word length
4. Enhanced: 16-bit Instruction Word length

This manual focuses on the Enhanced MCU family of devices, which are also referred to as the PIC18CXXX MCU family.

The operation of the Enhanced MCU family architecture and peripheral modules is explained, but does not cover, the specifics of each device. This manual is not intended to replace the device data sheets, but complement them. In other words, this guide supplies the general details and operation of the PICmicro architecture and peripheral modules, while the data sheet gives the specific details (such as device memory mapping).

Initialization examples are given throughout this manual. These examples sometimes need to be written as device specific as opposed to family generic, though they are valid for most other devices. Some modifications may be required for devices with variations in register file mappings.

1.3 Device Structure

Each part of a device can be placed into one of three groups:

1. Core
2. Peripherals
3. Special Features

1.3.1 Core

The core pertains to the basic features that are required to make the device operate. These include:

1.	Oscillator	Revision “DS39502A”
2.	Reset	Revision “DS39503A”
3.	Architecture	Revision “DS39504A”
4.	CPU (Central Processing Unit) and ALU (Arithmetic Logical Unit)	Revision “DS39505A”
5.	Hardware 8x8 Multiplier	Revision “DS31006A”
6.	Memory	Revision “DS31007A”
7.	Table Read / Table Write	Revision “DS39508A”
8.	System Bus	Revision “DS39509A”
9.	Interrupts	Revision “DS39510A”
10.	Instruction Set	Revision “DS39532A”

1.3.2 Peripherals

Peripherals are the features that add a differentiation from a microprocessor. These ease in interfacing to the external world (such as general purpose I/O, A/D inputs, and PWM outputs), and internal tasks, such as keeping different time bases (i.e. timers). The peripherals that are discussed are:

1.	I/O	Revision “DS39511A”
1.	Parallel Slave Port (PSP)	Revision “DS39512A”
2.	Timer0	Revision “DS39513A”
3.	Timer1	Revision “DS39514A”
4.	Timer2	Revision “DS39515A”
5.	Timer3	Revision “DS39516A”
6.	Capture/Compare/PWM (CCP)	Revision “DS39517A”
7.	Serial Slave Port (SSP)	Revision “DS39519A”
8.	Master Synchronous Serial Port (MSSP)	Revision “DS39520A”
9.	Addressable USART	Revision “DS39521A”
10.	CAN	Revision “DS39522A”
11.	Comparator Voltage Reference	Revision “DS31023A”
12.	Comparators	Revision “DS39525A”
13.	Compatible 10-bit A/D Converter	Revision “DS31026A”
14.	10-bit A/D Converter	Revision “DS31027A”

1.3.3 Special Features

Special features are the unique features that help to:

- Decrease system cost
- Increase system reliability
- Increase design flexibility

The Enhanced PICmicro MCUs offer several features that help achieve these goals. The special features discussed are:

1.	Low Voltage Detect	Revision “DS39528A”
2.	WDT and Sleep Operation	Revision “DS31029A”
3.	Device Configuration Bits	Revision “DS39530A”
4.	In-Circuit Serial Programming™ (ICSP™)	Revision “DS39531A”

1.3.4 Other Sections

This section provides the cross references for the remaining sections of this manual.

1.	Introduction	Revision “DS39501A”
2.	Electrical Specifications	Revision “DS31033A”
3.	Device Characteristics	Revision “DS31034A”
4.	Development Tools	Revision “DS31035A”
5.	Code Development	Revision “DS31036A”
6.	Appendix	Revision “DS39537A”
7.	Glossary	Revision “DS39538A”

1.4 Development Support

Microchip offers a wide range of development tools that allow users to efficiently develop and debug application code. Microchip's development tools can be broken down into four categories:

1. Code generation
2. Hardware/Software debug
3. Device programmer
4. Product evaluation boards

All tools developed by Microchip operate under the MPLAB® Integrated Development Environment (IDE), while some third party tools may not. The code generation tools include:

- MPASM
- MPLAB®-C17 (for PIC17CXXX family only)
- MPLAB®-C18 (for PIC18CXXX family only)

These software development programs include device header files. Each header file defines the register names (as shown in the device data sheet) to the specified address or bit location. Using the header files eases code migration and reduces the tediousness of memorizing a register's address or a bit's position in a register.

Note:	Microchip strongly recommends that the supplied header files be used in the source code of your program. This eases code migration, improves code readability, and increases the quality and depth of the technical support that Microchip can offer.

Tools which ease in debugging software are:

- MPLAB®-ICE In-Circuit Emulator
- PICMASTER® In-Circuit Emulator
- ICEPIC In-Circuit Emulator
- MPLAB®-SIM Software Simulator

After generating and debugging the application software, the device will need to be programmed. Microchip offers two levels of programmers:

1. PICSTART® Plus programmer
2. PRO MATE® II programmer

Demonstration boards allow the developer of software code to evaluate the capability and suitability of the device to the application. The demo boards offered are:

- PICDEM-1
- PICDEM-2 (can be used with PIC18CXX2 devices)
- PICDEM-3
- PICDEM-14A
- PICDEM-17

At the time of publication, only PICDEM-2 could be used with some Enhanced MCU devices.

A full description of each of Microchip's development tools is discussed in the **"Development Tools"** section. As new tools are developed, product briefs and user guides may be obtained from the Microchip web site (www.microchip.com) or from your local Microchip Sales Office.

Code development recommendations and techniques are provided in the **"Code Development"** section.

Microchip offers other reference tools to speed the development cycle. These include:

- Application Notes
- Reference Designs
- Microchip web site
- Local Sales Offices with Field Application Support
- Corporate Support Line

The Microchip web site lists other sites that may be useful references.

1.5 Device Varieties

Once the functional requirements of the device are specified, other choices need to be made. These include:

- Memory technology
- Operating voltage
- Operating temperature range
- Operating frequency
- Packaging

Microchip has a large number of options and option combinations, one of which should fulfill your requirements.

1.5.1 Memory Varieties

Memory technology has no effect on the logical operation of a device. Due to the different processing steps required, some electrical characteristics may vary between devices with the same feature set/pinout but with different memory technologies. An example is the electrical characteristic VIL (Input Low Voltage), which may have some difference between a typical EPROM device and a typical ROM device.

Each device has a variety of frequency ranges and packaging options available. Depending on application and production requirements, the proper device options can be identified using the information in the Product Identification System section at the end of each data sheet. When placing orders, please use the “Product Identification System” at the back of the data sheet to specify the correct part number.

When discussing the functionality of the device, the memory technology and the voltage range do not matter. Microchip offers three program memory types. The memory type is designated in the part number by the first letter(s) after the family affiliation designators.

1. **C**, as in PIC18**C**XXX. These devices have EPROM type memory.
2. **CR**, as in PIC18**CR**XXX. These devices have ROM type memory.
3. **F**, as in PIC18**F**XXX. These devices have FLASH type memory.

1.5.1.1 EPROM

Microchip focuses on Erasable Programmable Read Only Memory (EPROM) technology to give customers flexibility throughout their entire design cycle. With this technology, Microchip offers various packaging options as well as services.

1.5.1.2 Read Only Memory (ROM) Devices

Microchip offers a masked Read Only Memory (ROM) version of several of the highest volume parts, thus giving customers a lower cost option for high volume, mature products.

ROM devices do not allow serialization information in the program memory space.

For information on submitting ROM code, please contact your local Microchip sales office.

1.5.1.3 FLASH Memory Devices

These devices are electrically erasable, and can therefore be offered in a low cost plastic package. Being electrically erasable, these devices can be erased and reprogrammed without removal from the circuit. A device will have the same specifications whether it is used for prototype development, pilot programs or production.

1.5.2 Operating Voltage Range Options

All Enhanced PICmicro MCUs operate over the standard voltage range. Devices are also offered which operate over an extended voltage range (and reduced frequency range). Table 1-1 shows all possible memory types and voltage range designators for the PIC18CXXX MCU family. The designators are in **bold** typeface.

Table 1-1: Device Memory Type and Voltage Range Designators

Memory Type	Voltage Range	
	Standard	Extended
EPROM	PIC18**C**XXX	PIC18**LC**XXX
ROM	PIC18**CR**XXX	PIC18**LCR**XXX
FLASH	PIC18**F**XXX	PIC18**LF**XXX

Note: Not all memory types may be available for a particular device.

As you can see in Table 1-2, Microchip specifies its extended range devices at a more conservative voltage range until device characterization has ensured they will be able to meet the goal of their final design specifications.

Table 1-2: Typical Voltage Ranges for Each Device Type

Typical Voltage Range		EPROM		ROM		Flash	
Standard		**C**	4.2 - 5.5V	**CR**	4.2 - 5.5V	**F**	4.2 - 5.5V
Extended	Before device characterization	**LC**	3.0 - 5.5V	**LCR**	3.0 - 5.5V	**LF**	3.0 - 5.5V
	Final specification (1)	**LC**	2.5 - 5.5V	**LCR**	2.5 - 5.5V	**LF**	2.0 - 5.5V

Note 1: This voltage range depends on the results of device characterization.

Figure 1-1 and Figure 1-2 show example Voltage to Frequency Charts for the Enhanced MCU family. The voltages and frequencies of any given device may be different than those shown. These figures are intended to show that with an LC (extended voltage) device, the user can make a trade-off between voltage of operation and the frequency of the device. The device FMAX is given below the graph. As the voltages and frequencies of the device change, the equation for the device FMAX will change. Equation 1-1 shows the general equation to determine the device FMAX.

Equation 1-1: Generic FMAX Equation

$$F_{MAX} = (\text{slope})\ (V_{DDAPPMIN} - V_{DDMIN}) + \text{offset}$$

Figure 1-1: PIC18CXXX Voltage-Frequency Graph - Example

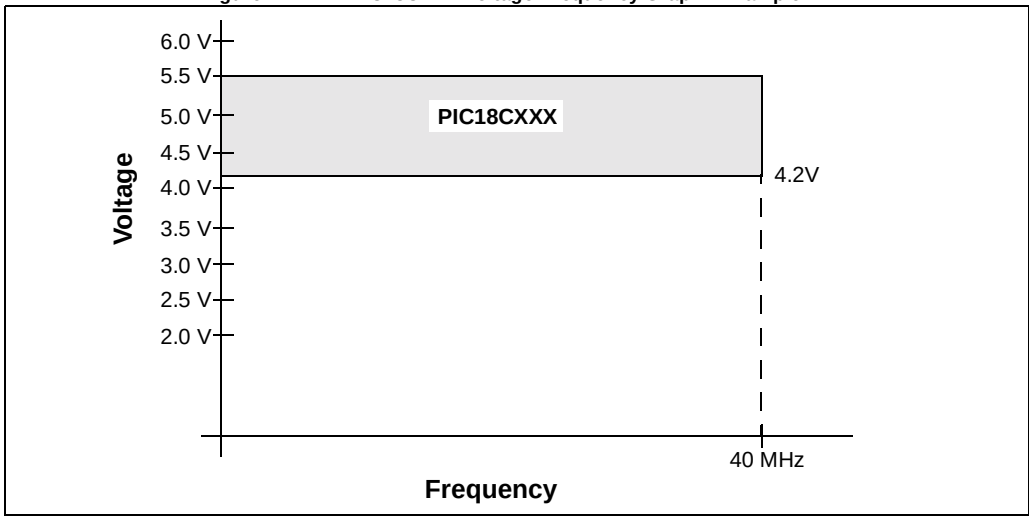
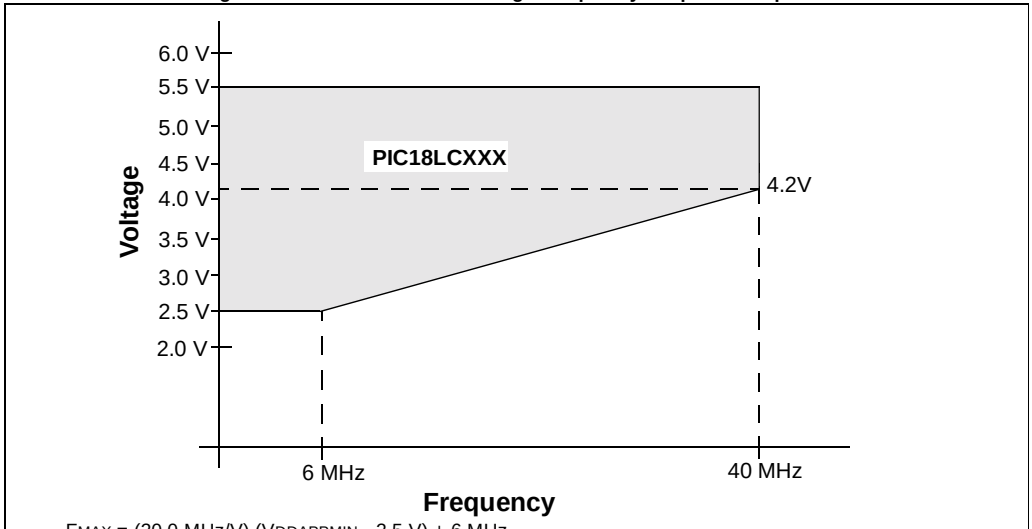


Figure 1-2: PIC18LCXXX Voltage-Frequency Graph - Example



$F_{MAX} = (20.0\ \text{MHz/V})\ (V_{DDAPPMIN} - 2.5\ \text{V}) + 6\ \text{MHz}$

Note: $V_{DDAPPMIN}$ is the minimum voltage of the PICmicro device in the application.

1.5.3 Packaging Varieties

Depending on the development phase of your project, one of three package types would be used.

The first package type is ceramic with an erasure window. The window allows ultraviolet light into the package so that the device inside may be erased. The package is used for the development phase, since the device's program memory can be erased and reprogrammed many times.

The second package type is a low cost plastic package. This package type is used in production where device cost is to be kept to a minimum.

Lastly, there is the Die option. A Die is an unpackaged device that has been tested. Dies are used in low cost designs and designs where board space is at a minimum. For additional information on die support, please refer to the Die Support Document (DS30258).

Table 1-3 shows a quick summary of the typical use for each package type.

Table 1-3: Typical Package Uses

Package Type	Typical Usage
Windowed	Development mode
Plastic	Production
Die	Special applications, such as those which require minimum board space

1.5.3.1 UV Erasable Devices

The UV erasable version of EPROM program memory devices is optimal for prototype development and pilot programs.

These devices can be erased and reprogrammed to any of the configuration modes. Third party programmers are available. Refer to Microchip's *Third Party Guide* (DS00104) for a list of sources.

The amount of time required to completely erase a UV erasable device depends on the:

- Wavelength of the light
- Intensity of the light
- Distance of the device from the UV source
- Process technology of the device (size of the memory cells).

Note: Fluorescent lights and sunlight both emit ultraviolet light at the erasure wavelength. Leaving a UV erasable device's window uncovered could cause, over time, the device's memory cells to become erased. The erasure time for a fluorescent light is about three years, while sunlight requires only about one week. To prevent the memory cells from losing data, an opaque label should be placed over the erasure window.

1.5.3.2 One-Time-Programmable (OTP) Devices

The availability of OTP devices is especially useful for customers expecting code changes and updates.

OTP devices in plastic packages permit the user to program them once. Often the system can be designed so that programming may be performed in-circuit (after the device has been mounted on the circuit board).

1.5.3.3 FLASH Devices

These devices are electrically erasable, and can therefore be offered in a low cost plastic package. Being electrically erasable, these devices can be both erased and reprogrammed without removal from the circuit. A device will have the same specifications whether it is used for prototype development, pilot programs, or production.

1.5.3.4 ROM Devices

ROM devices have their program memory fixed at the time of the silicon manufacture. Since the program memory cannot be changed, the device is usually housed in the low cost plastic package.

1.5.3.5 Die

The Die option allows the board size to become as small as physically possible. The Die Support document (DS30258) explains general information about using and designing with Die. There are also individual specification sheets that detail Die specific information. Manufacturing with Die requires special knowledge and equipment. This means that the number of manufacturing houses that support Die will be limited. If you decide to use the Die option, please research your manufacturing sites to ensure that they will be able to meet the specialized requirements of Die use.

1.5.3.6 Specialized Services

For OTP customers with established code, Microchip offers two specialized services. These two services, Quick Turn Production Programming and Serialized Quick Turn Production Programming, allow customers to shorten their manufacturing cycle time.

1.5.3.6.1 Quick Turn Production (QTP) Programming

Microchip offers this programming service for factory production orders. This service is made available for users who choose not to program a medium to high quantity of units at their factory and whose code patterns have stabilized. The devices are identical to the OTP devices, but with all EPROM locations and configuration options already programmed by Microchip. Certain code and prototype verification procedures apply before production shipments are available. Please contact your local Microchip sales office for more details.

1.5.3.6.2 Serialized Quick Turn Production (SQTP^{SM}) Programming

Microchip offers a unique programming service where a few user-defined locations in each device are programmed with unique numbers. These numbers may be:

- Random numbers
- Pseudo-random numbers
- Sequential numbers

Serial programming allows each device to have a unique number which can serve as an entry-code, password, ID, or serial number.

1.6 Style and Symbol Conventions

Throughout this document, certain style and font format conventions are used. Most format conventions imply a distinction should be made for the emphasized text. The MCU industry has many symbols and non-conventional word definitions/abbreviations. Table 1-4 provides a description for many of the conventions contained in this document. A glossary is provided in the **"Glossary"** section, which contains more word and abbreviation definitions that are used throughout this manual.

1.6.1 Document Conventions

Table 1-4 defines some of the symbols and terms used throughout this manual.

Table 1-4: Document Conventions

Symbol or Term	Description
set	To force a bit/register to a value of logic '1'.
clear	To force a bit/register to a value of logic '0'.
reset	1) To force a register/bit to its default state. 2) A condition in which the device places itself after a device reset occurs. Some bits will be forced to '0' (such as interrupt enable bits), while others will be forced to '1' (such as the I/O data direction bits).
0xnn or nnh	Designates the number 'nn' in the hexadecimal number system. These conventions are used in the code examples.
B'bbbbbbbb'	Designates the number 'bbbbbbbb' in the binary number system. This convention is used in the text and in figures and tables.
R-M-W	Read - Modify - Write. This is when a register or port is read, then the value is modified, and that value is then written back to the register or port. This action can occur from a single instruction (such as bit set file, BSF) or a sequence of instructions.
: (colon)	Used to specify a range or the concatenation of registers/bits/pins. One example is TMR1H:TMR1L, which is the concatenation of two 8-bit registers to form a 16-bit timer value, while SSPM3:SSPM0 are 4-bits used to specify the mode of the SSP module. Concatenation order (left-right) usually specifies a positional relationship (MSb to LSb, higher to lower).
< >	Specifies bit(s) locations in a particular register. One example is SSPCON<SSPM3:SSPM0> (or SSPCON<3:0>) which specifies the register and associated bits or bit positions.
`Courier Font`	Used for code examples, binary numbers and for Instruction Mnemonics in the text.
Times Font	Used for equations and variables.
Times, Bold Font, Italics	Used in explanatory text for items called out from a graphic/equation/example.
Note	A Note presents information that we wish to reemphasize, either to help you avoid a common pitfall, or make you aware of operating differences between some device family members. A Note is always in a shaded box (as below), unless used in a table, where it is at the bottom of the table (as in this table). **Note:** This is a Note in a note box.
Caution (1)	A caution statement describes a situation that could potentially damage software or equipment.
Warning (1)	A warning statement describes a situation that could potentially cause personal harm.

Note 1: The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

1.6.2 Electrical Specifications

Throughout this manual, there will be references to electrical specification parameter numbers. A parameter number represents a unique set of characteristics and conditions that is consistent between every data sheet, though the actual parameter value may vary from device to device.

The **"Electrical Specifications"** section shows all the specifications that are documented for all devices. No one device has all these specifications. This section is intended to let you know the types of parameters that Microchip specifies. The value of each specification is device dependent, though we strongly attempt to keep them consistent across all devices.

Table 1-5: Electrical Specification Parameter Numbering Convention

Parameter Number Format	Comment
DXXX	DC Specification
AXXX	DC Specification for Analog Peripherals
XXX	Timing (AC) Specification
PDXXX	Device Programming DC Specification
PXXX	Device Programming Timing (AC) Specification

Legend: XXX represents a number.

1.7 Related Documents

Microchip, as well as other sources, offers additional documentation which can aid in your development with PICmicro MCUs. These lists contain the most common documentation, but other documents may also be available. Please check the Microchip web site (www.microchip.com) for the latest published technical documentation.

1.7.1 Microchip Documentation

The following documents are available from Microchip. Many of these documents provide application specific information that give actual examples of using, programming and designing with PICmicro MCUs.

1. **MPASM and MPLINK (w/ MPLIB) User's Guide (DS33014)**
   This document explains how to use Microchip's MPASM assembler.
2. **MPLAB®-CXX Compiler User's Guide (DS51217)**
   This document explains how to use Microchip's MPLAB-C17 and MPLAB-C18 compilers.
3. **MPLAB® IDE, Simulator, Editor User's Guide (DS51025)**
   This document explains how to use Microchip's MPLAB Integrated Development Environment.
4. **MPLAB®-CXX Reference Guide Libraries and Precompiled Object Files (DS51224)**
   This document explains how to use Microchip's MPLAB Reference Guide Libraries and Precompiled Object Files.
5. **PICMASTER® User's Guide (DS30421)**
   This document explains how to use Microchip's PICMASTER In-Circuit Emulator.
6. **PRO MATE® User's Guide (DS30082)**
   This document explains how to use Microchip's PRO MATE Universal Programmer.
7. **PICSTART®-Plus User's Guide (DS51028)**
   This document explains how to use Microchip's PICSTART-Plus low-cost universal programmer.
8. **PICmicro® Mid-Range MCU Family Reference Manual (DS33023)**
   This document discusses the operation of PICmicro Mid-Range MCU devices, explaining the detailed operation of the architecture and peripheral modules. It is a compliment to the device data sheets for the Mid-Range family.
9. **Embedded Control Handbook Volume I (DS00092)**
   This document contains a plethora of application notes. This is useful for insight on how to use the device (or parts of it), as well as getting started on your particular application due to the availability of extensive code files.
10. **Embedded Control Handbook Update 2000 (DS00711)**
    This document contains additional application notes.
11. **Embedded Control Handbook Volume II Math Library (DS00167)**
    This document contains the Math Libraries for PICmicro MCUs.
12. **In-Circuit Serial Programming Guide™ (DS30277)**
    This document discusses implementing In-Circuit Serial Programming.
13. **PICDEM-1 User's Guide (DS33015)**
    This document explains how to use Microchip's PICDEM-1 demo board.
14. **PICDEM-2 User's Guide (DS30374)**
    This document explains how to use Microchip's PICDEM-2 demo board.
15. **PICDEM-3 User's Guide (DS51079)**
    This document explains how to use Microchip's PICDEM-3 demo board.
16. **PICDEM-14A User's Guide (DS51097)**
    This document explains how to use Microchip's PICDEM-14A demo board.
17. **PICDEM-17 User's Guide (DS39024)**
    This document explains how to use Microchip's PICDEM-17 demo board.
18. **Third Party Guide (DS00104)**
    This document lists Microchip's third parties, as well as various consultants.
19. **Die Support (DS30258)**
    This document gives information on using Microchip products in Die form.

1.7.2 Third Party Documentation

There are several documents available from third party sources around the world. Microchip does not review these documents for technical accuracy. However, they may be a helpful source for understanding the operation of Microchip PICmicro MCU devices. This is not necessarily a complete list, but are the documents that we were aware of at the time of printing. For more information on how to contact some of these sources, as well as any new sources that we become aware of, please visit the Microchip web site (www.microchip.com).

DOCUMENT	LANGUAGE
The PIC16C5X Microcontroller: A Practical Approach to Embedded Control Bill Rigby/ Terry Dalby, Tecksystems Inc. 0-9654740-0-3	English
Easy PIC'n David Benson, Square 1 Electronics 0-9654162-0-8	English
A Beginner's Guide to the Microchip PIC® Nigel Gardner, Bluebird Electronics 1-899013-01-6	English
PIC Microcontroller Operation and Applications DN de Beer, Cape Technikon	English
Digital Systems and Programmable Interface Controllers WP Verburg, Pretoria Technikon	English
Mikroprozessor PIC16C5X Michael Rose, Hüthig 3-7785-2169-1	German
Mikroprozessor PIC17C42 Michael Rose, Hüthig 3-7785-2170-5	German
Les Microcontrolleurs PIC et mise en oeuvre Christian Tavernier, Dunod 2-10-002647-X	French
Microcontrolleurs PIC a structure RISC C.F. Urbain, Publitronic 2-86661-058-X	French
New Possibilities with the Microchip PIC RIGA	Russian

DOCUMENT	LANGUAGE
PIC16C5X/71/84 Development and Design, Part 1 United Tech Electronic Co. Ltd 957-21-0807-7	Chinese
PIC16C5X/71/84 Development and Design, Part 2 United Tech Electronic Co. Ltd 957-21-1152-3	Chinese
PIC16C5X/71/84 Development and Design, Part 3 United Tech Electronic Co. Ltd 957-21-1187-6	Chinese
PIC16C5X/71/84 Development and Design, Part 4 United Tech Electronic Co. Ltd 957-21-1251-1	Chinese
PIC16C5X/71/84 Development and Design, Part 5 United Tech Electronic Co. Ltd 957-21-1257-0	Chinese
PIC16C84 MCU Architecture and Software Development ICC Company 957-8716-79-6	Chinese

1.8 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced MCU family (they may be written for the Base-Line, Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to an introduction to Microchip's PICmicro MCUs are:

Title	Application Note #
A Comparison of Low End 8-bit Microcontrollers	AN520
PIC16C54A EMI Results	AN577
Continuous Improvement	AN503
Improving the Susceptibility of an Application to ESD	AN595
Plastic Packaging and the Effects of Surface Mount Soldering Techniques	AN598
Migrating Designs from PIC16C74A/74B to PIC18C442	AN716
PIC17CXXX to PIC18CXXX Migration	AN726

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

1.9 Revision History

Revision A

This is the initial released revision of Enhanced MCU Introduction.

Section 2. Oscillator

HIGHLIGHTS

This section of the manual contains the following major topics:

2.1 Introduction

The device system clock is required for the device to execute instructions and for the peripherals to function. Four device system clock periods (TSCLK) generate one internal instruction clock cycle (TCY).

The device system clock (TSCLK) is derived from an external system clock. This external system clock can be generated in one of eight different oscillator modes. The device configuration bits select the oscillator mode. Device configuration bits are nonvolatile memory locations and the operating mode is determined by the value written during device programming. The oscillator modes are:

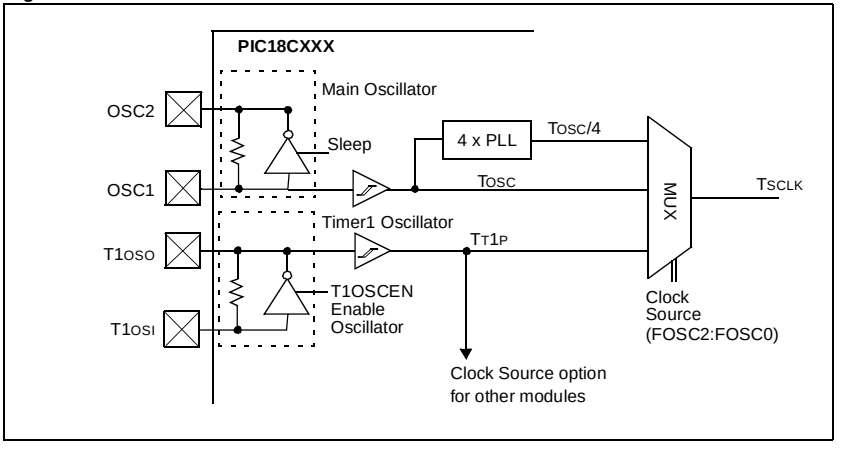
- EC External Clock
- ECIO External Clock with I/O pin enabled
- LP Low Frequency (Power) Crystal
- XT Crystal/Resonator
- HS High Speed Crystal/Resonator
- RC External Resistor/Capacitor
- RCIO External Resistor/Capacitor with I/O pin enabled
- HS4 High Speed Crystal/Resonator with 4x frequency PLL multiplier enabled

Multiple oscillator circuits can be implemented on an Enhanced Architecture device. There is the default oscillator (OSC1), and additional oscillators may be available, such as the Timer1 oscillator. Software may allow these auxiliary oscillators to be switched in as the device oscillator. The Timer1 oscillator is a low frequency (low power) oscillator that is designed to be operated at 32kHz. Figure2-1 shows a block diagram of the oscillator options.

The output signal of the Timer1 oscillator circuitry is a low frequency (power) clock source (TT1P). The source for the device system clock can be switched from the default clock (TSCLK) to the 32kHz-clock low power clock source (TT1P) under software control. Switching to the 32kHz low frequency (power) clock source from any of the eight default clock sources may allow power saving.

These oscillator options are made available to allow a single device type the flexibility to fit applications with different oscillator requirements. The RC oscillator option saves system cost, while the LP crystal option saves power. The HS4 option allows frequency of incoming crystal oscillator signal to be multiplied by four for higher internal clock frequency. This is useful for customers who are concerned with EMI due to high frequency crystals. The device configuration bits are used to select these various options. For more details on the device configuration bits, see the **"Device Configuration Bits"** section.

Figure 2-1: Device Clock Sources



2.2 Control Register

Register 2-1 shows the OSCCON register which contains the control bit to allow switching of the system clock between the primary oscillator and the Timer1 oscillator.

Register 2-1: OSCCON Register

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-1
—	—	—	—	—	—	—	SCS
bit 7							bit 0

bit 7-1 **Unimplemented:** Read as '0'

bit 0 **SCS:** System Clock Switch bit
when OSCSEN configuration bit = '0' and T1OSCEN bit is set:
1 = Switch to Timer1 Oscillator/Clock pin
0 = Use primary Oscillator/Clock input pin
when OSCSEN and T1OSCEN are in other states:
bit is forced clear

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: The Timer1 oscillator must be enabled to switch the system clock source. The Timer1 oscillator is enabled by setting the T1OSCEN bit in the Timer1 control register (T1CON). If the Timer1 oscillator is not enabled, then any write to the SCS bit will be ignored (SCS bit forced cleared) and the main oscillator will continue to be the system clock source.

2.3 Oscillator Configurations

The oscillator selection is configured at time of device programming. The user can program up to three device configuration bits (FOSC2:FOSC0) to select one of eight modes.

2.3.1 Oscillator Types

PIC18CXXX devices can have up to eight different oscillator modes for the default clock source (TSCLK). These eight modes are:

- EC External Clock
- ECIO External Clock with IO pin enabled
- LP Low Frequency (Power) Crystal
- XT Crystal/Resonator
- HS High Speed Crystal/Resonator
- RC External Resistor/Capacitor
- RCIO External Resistor/Capacitor with IO pin enabled
- HS4 High Speed Crystal/Resonator with 4x frequency PLL multiplier enabled

The main difference between the LP, XT and HS modes is the gain of the internal inverter of the oscillator circuit, which allows the different frequency ranges. Table 2-1 gives information to aid in selecting an oscillator mode. In general, use the oscillator option with the lowest possible gain that still meets specifications. This will result in lower dynamic currents (IDD). The frequency range of each oscillator mode is the recommended frequency cutoff, but the selection of a different gain mode is acceptable as long as a thorough validation is performed (voltage, temperature, component variations (resistor, capacitor, and internal microcontroller oscillator circuitry).

Switching the system clock source to the alternate clock source is controlled by the application software. The user can switch from any of the eight default clock sources. This is done by setting the SCS (System Clock Switch) bit in the OSCCON register. The requirements for switching to the alternate clock source are:

- Timer1 clock oscillator must be enabled (T1OSCEN is set '1').
- The $\overline{\text{OSCEN}}$ configuration bit must be cleared ('0').

Table 2-1: Selecting the Oscillator Mode for Devices

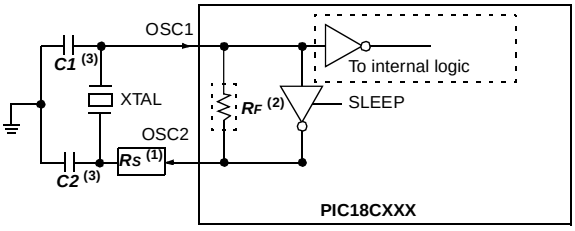
FOSC2:FOSC0 Configuration Bits	OSC Mode	OSC Feedback Inverter Gain	OSC2/CLKO Function	Comment
1 1 1	RCIO	Zero Gain. Device turned off to save current.	I/O	Least expensive solution for device oscillation (only an external resistor and capacitor is required). Most variation in time-base. Device's default mode. OSC2/CLKO is configured as general purpose I/O pin. This pin is multiplexed with one of the device's PORT pins.
1 1 0	HS4	High Gain	—	Highest frequency application. This works with the HS oscillator circuit mode and phase lock loop. This mode consumes the most current. The internal phase lock loop circuit multiplies the external oscillator frequency by 4.
1 0 1	ECIO	Zero Gain. Device turned off to save current.	I/O	External clock mode with OSC2/CLKO configured as general purpose I/O pin. This pin is multiplexed with one of the device's PORT pins. OSC1/CLKI is hi-impedance and can be driven by CMOS drivers.
1 0 0	EC	Zero Gain. Device turned off to save current.	Clock out with oscillator frequency divided by 4.	External clock mode with OSC2/CLKO configured with oscillator frequency divided by 4. OSC1/CLKI is hi-impedance and can be driven by CMOS drivers.
0 1 1	RC	Zero Gain. Device turned off to save current.	Clock out with oscillator frequency divided by 4.	Inexpensive solution for device oscillation. Most variation in timebase. CLKOUT is enabled on OSC2/CLKO with oscillator frequency divided by 4.
0 1 0	HS	High Gain	—	High frequency application. Oscillator circuit's mode consumes the most current of the three crystal modes.
0 0 1	XT	Medium Gain	—	Standard crystal/resonator frequency.
0 0 0	LP	Low Gain	—	Low power/frequency applications. Oscillator circuit's mode consumes the least current of the three crystal modes.

2.4 Crystal Oscillators/Ceramic Resonators

In XT, LP, HS and HS4 modes, a crystal or ceramic resonator is connected to the OSC1 and OSC2 pins to establish oscillation (Figure2-2). The PIC18CXXX oscillator design requires the use of a parallel cut crystal. Using a series cut crystal may give a frequency out of the crystal manufacturer's specifications. When in EC and ECIO mode, the device can have an external clock source drive the OSC1 pin (Figure2-3).

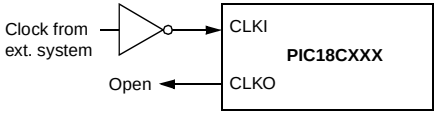
See Table 3-1 in the **"Reset"** section for time-out delays associated with crystal oscillators.

Figure 2-2: Crystal or Ceramic Resonator Operation (HS4, HS, XT or LP Oscillator Mode)



Note 1: A series resistor, ***Rs***, may be required for AT strip cut crystals.
2: The internal feedback resistor, R_F, is typically in the range of 2 to 10 MΩ.
3: See Table 2-2 and 2-3 for example values of *C1* and *C2*.

Figure 2-3: External Clock Input Operation (EC or ECIO Oscillator Modes)



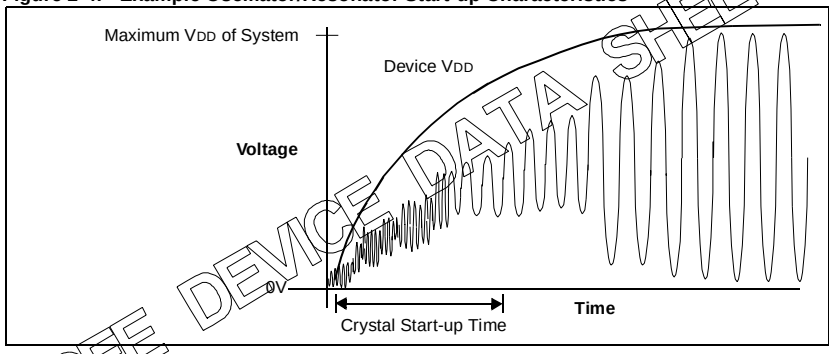
2.4.1 Oscillator/Resonator Start-up

As the device voltage increases from VSS, the oscillator will start its oscillations. The time required for the oscillator to start oscillating depends on many factors. These include:

- Crystal/resonator frequency
- Capacitor values used (***C1*** and ***C2*** in Figure2-2)
- Device VDD rise time
- System temperature
- Series resistor value and type if used (***Rs*** in Figure2-2)
- Oscillator mode selection of device (selects the gain of the internal oscillator inverter)
- Crystal quality
- Oscillator circuit layout
- System noise

Figure2-4 graphs an example oscillator/resonator start-up. The peak-to-peak voltage of the oscillator waveform can be quite low (less than 50% of device VDD), when the waveform is centered at VDD/2 (refer to parameters D033 and D043 in the **"Electrical Specifications"** section).

Figure 2-4: Example Oscillator/Resonator Start-up Characteristics



2.4.2 Component Selection

Figure2-2 is a diagram of the device's crystal or ceramic resonator circuitry. The resistance for the feedback resistor, ***RF***, is typically within the 2 to 10 MΩ range. This varies with device voltage, temperature and process variations. A series resistor, ***Rs***, may be required if an AT strip cut crystal is used. Be sure to include the device's operating voltage and the device's manufacturing process when determining resistor requirements. As you can see in Figure2-2, the connection to the device's internal logic is device dependent. See the applicable data sheet for device specifics. The typical values of capacitors (***C1***, ***C2***) are given in Table 2-2 and Table 2-3. Each device's data sheet will give the specific values that we test to at Microchip.

Table 2-2: Example Capacitor Selection for Ceramic Resonators

Ranges tested:			
Mode	**Frequency**	***C1*** **(1)**	***C2*** **(1)**
XT	455 kHz 2.0 MHz 4.0 MHz	TBD TBD TBD	TBD TBD TBD
HS	8.0 MHz 16.0 MHz	TBD TBD	TBD TBD

Resonators used:		
Frequency	**Manufacturer**	**Tolerance**
455 kHz	Panasonic EFO-A455K04B	±0.3%
2.0 MHz	Murata Erie CSA2.00MG	±0.5%
4.0 MHz	Murata Erie CSA4.00MG	±0.5%
8.0 MHz	Murata Erie CSA8.00MT	±0.5%
16.0 MHz	Murata Erie CSA16.00MX	±0.5%

Note 1: Recommended values of ***C1*** and ***C2*** are identical to the ranges tested above. Higher capacitance increases the stability of the oscillator but also increases the start-up time. These values are for design guidance only. Since each resonator has its own characteristics, the user should consult the resonator manufacturer for appropriate values of external components or verify oscillator performance.

2: All resonators tested required external capacitors.

Table 2-3: Example Capacitor Selection for Crystal Oscillator

Mode	Frequency	*C1* (1)	*C2* (1)
LP	32 kHz 200 kHz	TBD TBD	TBD TBD
XT	200 kHz 1 MHz 4 MHz	TBD TBD TBD	TBD TBD TBD
HS	4.0 MHz 8 MHz 20 MHz 25 MHz	TBD TBD TBD TBD	TBD TBD TBD TBD

Crystals used:

Frequency	Manufacturer	Tolerance
32.0 kHz	Epson C-001R32.768K-A	± 20 PPM
200 kHz	STD XTL 200.000 kHz	± 20 PPM
1.0 MHz	ECS ECS-10-13-1	± 50 PPM
4.0 MHz	ECS ECS-40-20-1	± 50 PPM
8.0 MHz	EPSON CA-301 8.000 M-C	± 30 PPM
20.0 MHz	EPSON CA-301 20.000 M-C	± 30 PPM

Note 1: Higher capacitance increases the stability of the oscillator, but also increases the start-up time. These values are for design guidance only. A series resistor, ***Rs***, may be required in HS mode, as well as XT mode, to avoid overdriving crystals with low drive level specification. Since each crystal has its own characteristics, the user should consult the crystal manufacturer for appropriate values of external components or verify oscillator performance.

2.4.3 Tuning the Oscillator Circuit

Since Microchip devices have wide operating ranges (frequency, voltage, and temperature; depending on the part and version ordered) and external components (crystals, capacitors,...) of varying quality and manufacture, validation of operation needs to be performed to ensure that the component selection will comply with the requirements of the application.

There are many factors that go into the selection and arrangement of these external components. These factors include:

- amplifier gain
- desired frequency
- resonant frequency(s) of the crystal
- temperature of operation
- supply voltage range
- start-up time
- stability
- crystal life
- power consumption
- simplification of the circuit
- use of standard components
- combination which results in fewest components

2.4.3.1 Determining Best Values for Crystals, Clock Mode, *C1, C2,* and *Rs*

The best method for selecting components is to apply a little knowledge and a lot of trial, measurement, and testing.

Crystals are usually selected by their parallel resonant frequency only, however other parameters may be important to your design, such as temperature or frequency tolerance. Application Note AN588 is an excellent reference if you would like to know more about crystal operation and their ordering information.

The PICmicro's internal oscillator circuit is a parallel oscillator circuit, which requires that a parallel resonant crystal be selected. The load capacitance is usually specified in the 20 pF to 32 pF range. The crystal will oscillate closest to the desired frequency with capacitance in this range. It may be necessary to sometimes alter these values a bit, as described later, in order to achieve other benefits.

Clock mode is primarily chosen by using the FOSC parameter specification (parameter 1A) in the device data sheet, based on frequency. Clock modes (except RC and EC) are simply gain selections; lower gain for lower frequencies, higher gain for higher frequencies. It is possible to select a higher or lower gain, if desired, based on the specific needs of the oscillator circuit.

C1* and *C2 should also be initially selected based on the load capacitance as suggested by the crystal manufacturer and the tables supplied in the device data sheet. The values given in the device data sheet can only be used as a starting point, since the crystal manufacturer, supply voltage, and other factors already mentioned may cause your circuit to differ from the one used in the factory characterization process.

Ideally, the capacitance is chosen so that it will oscillate at the highest temperature and lowest VDD that the circuit will be expected to perform under. High temperature and low VDD both have a limiting effect on the loop gain, such that if the circuit functions at these extremes, the designer can be more assured of proper operation at other temperatures and supply voltage combinations. The output sine wave should not be clipped in the highest gain environment (highest VDD and lowest temperature) and the sine output amplitude should be great enough in the lowest gain environment (lowest VDD and highest temperature) to cover the logic input requirements of the clock as listed in the device data sheet.

A method for improving start-up is to use a value of ***C2*** greater than ***C1***. This causes a greater phase shift across the crystal at power-up, which speeds oscillator start-up.

Besides loading the crystal for proper frequency response, these capacitors can have the effect of lowering loop gain if their value is increased. ***C2*** can be selected to affect the overall gain of the circuit. A higher ***C2*** can lower the gain if the crystal is being over driven (see also discussion on ***Rs***). Capacitance values that are too high can store and dump too much current through the crystal, so ***C1*** and ***C2*** should not become excessively large. Unfortunately, measuring the wattage through a crystal is tricky business, but if you do not stray too far from the suggested values, you should not have to be concerned with this.

A series resistor, ***Rs***, is added to the circuit if, after all other external components are selected to satisfaction, the crystal is still being overdriven. This can be determined by looking at the OSC2 pin, which is the driven pin, with an oscilloscope. Connecting the probe to the OSC1 pin will load the pin too much and negatively affect performance. Remember that a scope probe adds its own capacitance to the circuit, so this may have to be accounted for in your design, (i.e. if the circuit worked best with a ***C2*** of 20 pF and scope probe was 10 pF, a 30 pF capacitor may actually be called for). The output signal should not be clipping or squashed. Overdriving the crystal can also lead to the circuit jumping to a higher harmonic level or even crystal damage.

The OSC2 signal should be a clean sine wave that easily spans the input minimum and maximum of the clock input pin (4V to 5V peak to peak for a 5V V_{DD} is usually good). An easy way to set this is to again test the circuit at the minimum temperature and maximum V_{DD} that the design will be expected to perform in, then look at the output. This should be the maximum amplitude of the clock output. If there is clipping or the sine wave is squashing near V_{DD} and V_{SS} at the top and bottom, increasing load capacitors will risk too much current through the crystal or push the value too far from the manufacturer's load specification. Add a trimpot between the output pin and ***C2***, and adjust it until the sine wave is clean. Keeping it fairly close to maximum amplitude at the low temperature and high V_{DD} combination will assure this is the maximum amplitude the crystal will see and prevent overdriving. A series resistor, ***Rs***, of the closest standard value can now be inserted in place of the trimpot. If ***Rs*** is too high, perhaps more than 20k ohms, the input will be too isolated from the output, making the clock more susceptible to noise. If you find a value this high is needed to prevent overdriving the crystal, try increasing ***C2*** to compensate. Try to get a combination where ***Rs*** is around 10k or less and load capacitance is not too far from the 20 pF or 32 pF manufacturer specification.

2.4.3.1.1 Start-up

The most difficult time for the oscillator to start-up is when waking up from sleep. This is because the load capacitors have both partially charged to some quiescent value, and phase differential at wake-up is minimal. Thus, more time is required to achieve stable oscillation. Remember also that low voltage, high temperatures and the lower frequency clock modes also impose limitations on loop gain, which in turn affects start-up. Each of the following factors makes the start-up time worse:

- a low frequency design (with its low gain clock mode)
- a quiet environment (such as a battery operated device)
- operating in a shielded box (away from the noisy ***RF*** area)
- low voltage
- high temperature
- waking up from sleep.

Noise actually helps a design for oscillator start-up, since it helps “kick start” the oscillator.

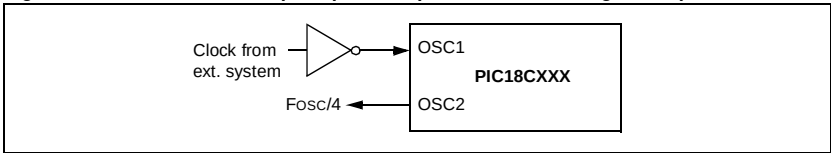
2.4.4 External Clock Input

Two of the oscillator modes use an external clock. These modes are EC and ECIO oscillator modes.

In the EC mode (Figure2-5), the OSC1 pin can be driven by CMOS drivers. In this mode, the OSC1/CLKI pin is hi-impedance and the OSC2/CLKO pin is the CLKO output (FOSC/4). The output is at a frequency of the selected oscillator divided by 4. This output clock is useful for testing or synchronization purposes. If the power-up timer is disabled, then there is no time-out after a POR, or else there will be a power-up timer. There is always a power-up time after a brown-out reset.

The feedback device between OSC1 and OSC2 is turned off to save current. There is no oscillator start-up time required after wake-up from sleep mode. If the power-up timer is disabled, then there is no time-out after a POR, or else (power-up timer enabled) there will be a power-up timer delay after POR. There is always a power-up timer after a brown-out reset.

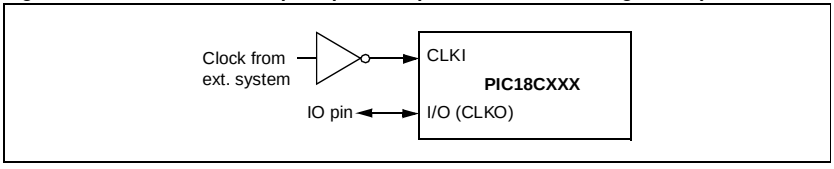
Figure 2-5: External Clock Input Operation (EC Oscillator Configuration)



In the ECIO mode (Figure2-6), the OSC1 pin can be driven by CMOS drivers. In this mode, the OSC1/CLKI pin is hi-impedance and the OSC2/CLKO is now multiplexed with a general purpose I/O pin.

The feedback device between OSC1 and OSC2 is turned off to save current. There is no oscillator start-up time required after wake-up from sleep mode. If the power-up timer is disabled, then there is no time-out after a POR, or else (power-up timer enabled) there will be a power-up timer delay after POR. There is always a power-up timer after a brown-out reset.

Figure 2-6: External Clock Input Operation (ECIO Oscillator Configuration)



2.4.5 External Crystal Oscillator Circuit for Device Clock

Sometimes more than one device needs to be clocked from a single crystal. Since Microchip does not recommend connecting other logic to the PICmicro's internal oscillator circuit, an external crystal oscillator circuit is recommended. Each device will then have an external clock source, and the number of devices that can be driven will depend on the buffer drive capability. This circuit is also useful when more than one device needs to operate synchronously to each other.

Either a prepackaged oscillator can be used or a simple oscillator circuit with TTL gates can be built. Prepackaged oscillators provide a wide operating range and better stability. A well-designed crystal oscillator will provide good performance with TTL gates. Two types of crystal oscillator circuits can be used; one with series resonance or one with parallel resonance.

Figure2-7 shows implementation of an external parallel resonant oscillator circuit. The circuit is designed to use the fundamental frequency of the crystal. The 74AS04 inverter performs the 180-degree phase shift that a parallel oscillator requires. The 4.7 kΩ resistor affects the circuit in three ways:

1. Provides negative feedback.
2. Biases the 74AS04 (#1) into the linear region.
3. Bounds the gain of the amplifier.

The 10 kΩ potentiometer is used to prevent overdriving of the crystal. It dissipates the power of the amplifier and allows the requirements of the crystal to be met.

Figure 2-7: External Parallel Resonant Crystal Oscillator Circuit

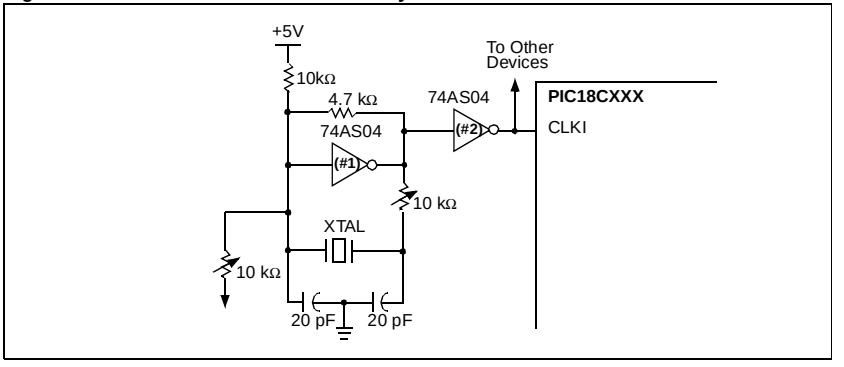
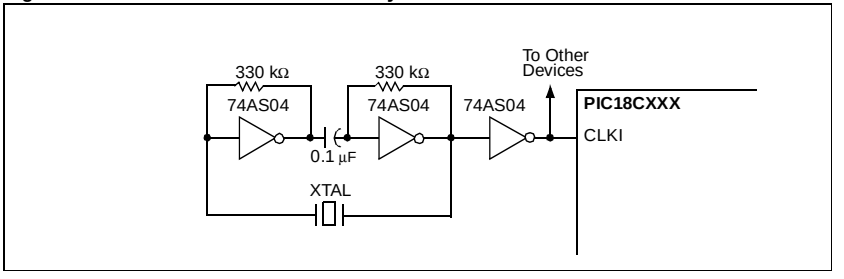


Figure2-8 shows an external series resonant oscillator circuit. This circuit is also designed to use the fundamental frequency of the crystal. The inverter performs a 180-degree phase shift in a series resonant oscillator circuit. The 330 kΩ resistors provide the negative feedback to bias the inverters in their linear region.

Figure 2-8: External Series Resonant Crystal Oscillator Circuit



When the device is clocked from an external clock source (as in Figure2-7 or Figure2-8) then the microcontroller's oscillator should be configured for EC or ECIO mode (Figure2-3).

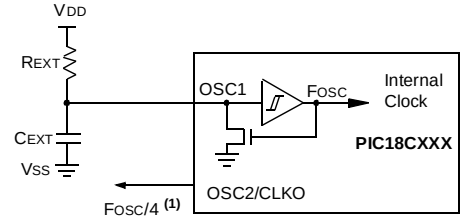
2.5 External RC Oscillator

For timing insensitive applications, the RC and RCIO device options offer additional cost savings. The RC oscillator frequency is a function of the:

- Supply voltage
- External resistor (REXT) values
- External capacitor (CEXT) values
- Operating temperature

In addition to this, the oscillator frequency will vary from unit to unit due to normal process parameter variation. Furthermore, the difference in lead frame capacitance between package types will also affect the oscillation frequency, especially for low CEXT values. The user also needs to take into account variation due to tolerance of external REXT and CEXT components used. Figure2-9 shows how the RC combination is connected. For REXT values below 2.2 kΩ, oscillator operation may become unstable, or stop completely. For very high REXT values (e.g. 1 MΩ), the oscillator becomes sensitive to noise, humidity and leakage. Thus, we recommend keeping REXT between 3 kΩ and 100 kΩ.

Figure 2-9: RC Oscillator Mode



Note 1: This output may also be configured as a general purpose I/O pin.

Although the oscillator will operate with no external capacitor (CEXT = 0 pF), we recommend using values above 20 pF for noise and stability reasons. With no or a small external capacitance, the oscillation frequency can vary dramatically due to changes in external capacitances, such as PCB trace capacitance and package lead frame capacitance.

See characterization data for RC frequency variation from part to part due to normal process variation. The variation is larger for larger resistance (since leakage current variation will affect RC frequency more for large R) and for smaller capacitance (since variation of input capacitance will affect RC frequency more).

See characterization data for the variation of oscillator frequency due to VDD for given REXT/CEXT values, as well as frequency variation due to operating temperature for given REXT, CEXT and VDD values.

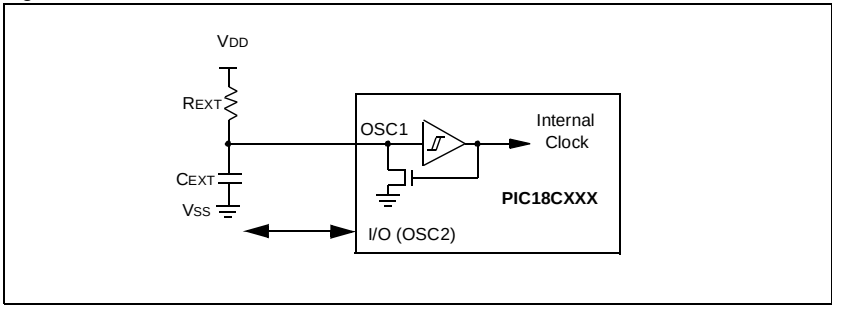
The oscillator frequency, divided by 4, is available on the OSC2/CLKO pin, and can be used for test purposes or to synchronize other logic (see **Figure 4-3: "Clock/Instruction Cycle"** in the **"Architecture"** section, for waveform).

2.5.1 RC Oscillator with I/O Enabled

The RCIO oscillator mode functions in the exact same manner as the RC oscillator mode. The only difference is that OSC2 pin does not output oscillator frequency divided by 4, but in this mode is configured as an I/O pin.

As in the RC mode, the user needs to take into account any variation of the clock frequency due to tolerance of external REXT and CEXT components used, process variation, voltage, and temperature. Figure2-10 shows how the RC with the I/O pin combination is connected.

Figure 2-10: RCIO Oscillator Mode



2.5.2 RC Start-up

As the device voltage increases, the RC will start its oscillations immediately after the pin voltage levels meet the input threshold specifications (parameters D032 and D042 in the **"Electrical Specifications"** section). The time required for the RC to start oscillating depends on many factors. These include:

- Resistor value used
- Capacitor value used
- Device VDD rise time
- System temperature

There is no oscillator start-up time (TOST) regardless of the source of reset or when sleep is terminated. If the power-up timer is disabled, then there is no time-out after a POR, or else (power-up timer enabled) there will be a power-up timer delay after POR. There is always a power-up time after a brown-out reset.

2.6 HS4 (HS oscillator with 4xPLL enabled)

A Phase Locked Loop (PLL) circuit is provided as a programmable option for users that want to multiply the frequency of the incoming crystal oscillator signal by 4. For an input clock frequency of 10 MHz, the internal clock frequency will be multiplied to 40 MHz. This is useful for customers who are concerned with EMI due to high frequency crystals.

The PLL can only be enabled when the oscillator configuration bits are programmed for HS4 mode (FOSC2:FOSC0 = ‘110’). If they are programmed for any other mode, the PLL is not enabled and the system clock will come directly from OSC1. The oscillator mode is specified during device programming.

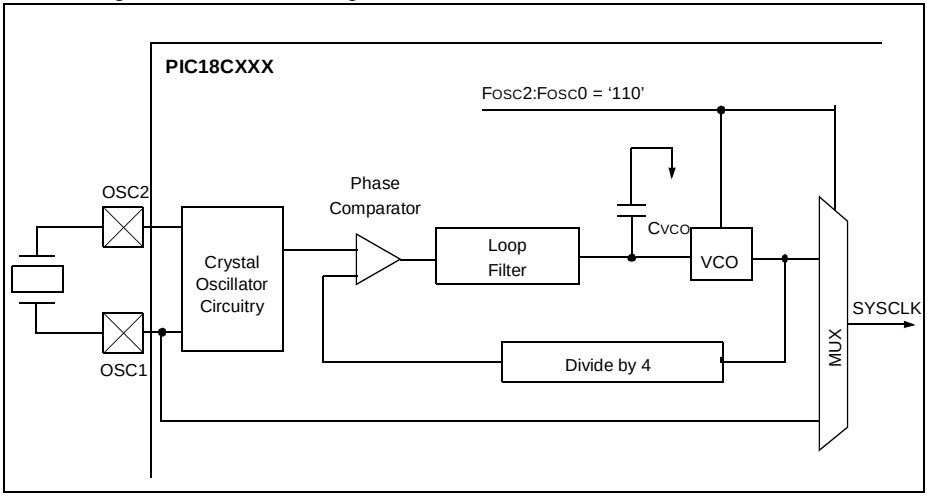
The PLL is divided into four basic parts (see Figure2-11):

- Phase comparator
- Loop filter
- VCO (Voltage Controlled Oscillator)
- Feedback divider

When in HS4 mode, the incoming clock is sampled by the phase comparator and is compared to PLL output clock divided by four. If the two are not in phase, the phase comparator drives an input to the loop filter to "pump" the voltage to the VCO, either up or down, depending upon whether the input clock was leading or lagging the output clock. This process continues until the incoming clock on OSC1 and the divide by 4 output clock of the VCO are in phase. The output clock is now "locked" in phase with the incoming clock, and its frequency is four times greater.

A PLL lock timer is used to ensure that the PLL has locked before device execution starts. The PLL lock timer has a time-out that is called TPLL. This delay is shown in Figure2-14.

Figure 2-11: PLL Block Diagram



2.7 Switching to Low Power Clock Source

This feature allows the clock source to switch from the default clock source that is selected by the FOSC2:FOSC0 bits to the Timer1 oscillator clock source. The availability of this feature is device dependent.

2.7.1 Switching Oscillator Mode Option

This feature is enabled by clearing the Oscillator System Clock Switch Enable ($\overline{\text{OSCSEN}}$) configuration bit. This provides the ability to switch to a low power execution mode if the alternate clock source (such as Timer1) is configured in oscillator mode with a low frequency (32 kHz, for example) crystal.

The enabling of the low power clock source is determined by the state of the SCS control bit in the Oscillator control register (OSCCON). (Register 2-1)

2.7.1.2 System Clock Switch Bit

The system clock switch bit, SCS (OSCCON) controls the switching of the oscillator source. It can be configured for either the Timer1 Oscillator clock source, or the default clock source (selected by the Fosc2:Fosc0 bits). When the SCS bit is set, it enables the Timer1 Oscillator clock source as the system clock. When the SCS bit is cleared, the system clock comes from the clock source specified by the Fosc2:Fosc0 bits. The SCS bit is cleared on all forms of reset.

Note:	The Timer1 oscillator must be enabled in order to switch the system clock source. The Timer1 oscillator is enabled by setting the T1OSCEN bit in the Timer1 Control Register (T1CON). If the Timer1 oscillator is not enabled, then any write to the SCS bit will be ignored, and the SCS bit will remain in the default state with the clock source coming from OSC1 or the PLL output.

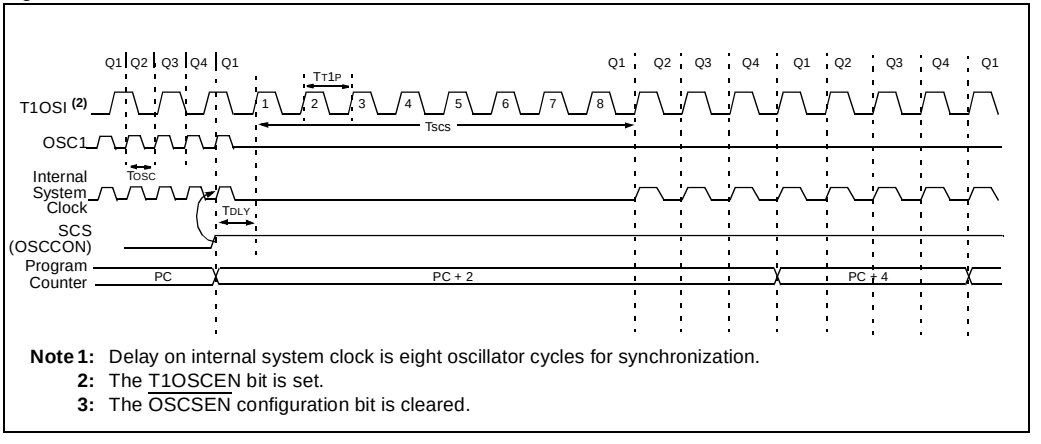
2.7.2 Oscillator Transitions

Switching from the default clock to the Timer1 Oscillator clock source is controlled as shown in the flow diagram (Figure2-16). This ensures a clean transition when switching oscillator clocks.

Circuitry is used to prevent "glitches" due to transitions when switching from the default clock source to the low power clock source and vice versa. Essentially, the circuitry waits for eight rising edges of the clock input to which the processor is switching. This ensures that the clock output pulse width will not be less than the shortest pulse width of the two clock sources. No additional delays are required when switching from the default clock source to the low power clock source.

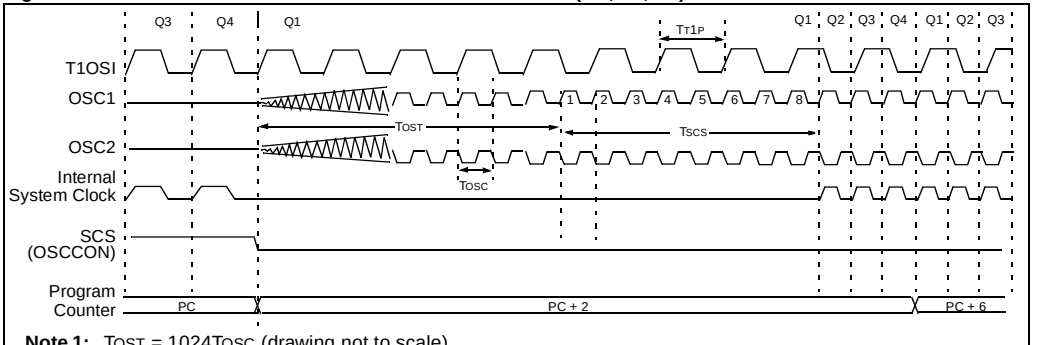
Figure2-12 through Figure2-15 show different transition waveforms when switching between the oscillators.

Figure 2-12: Transition From OSC1 to Timer1 Oscillator Waveform



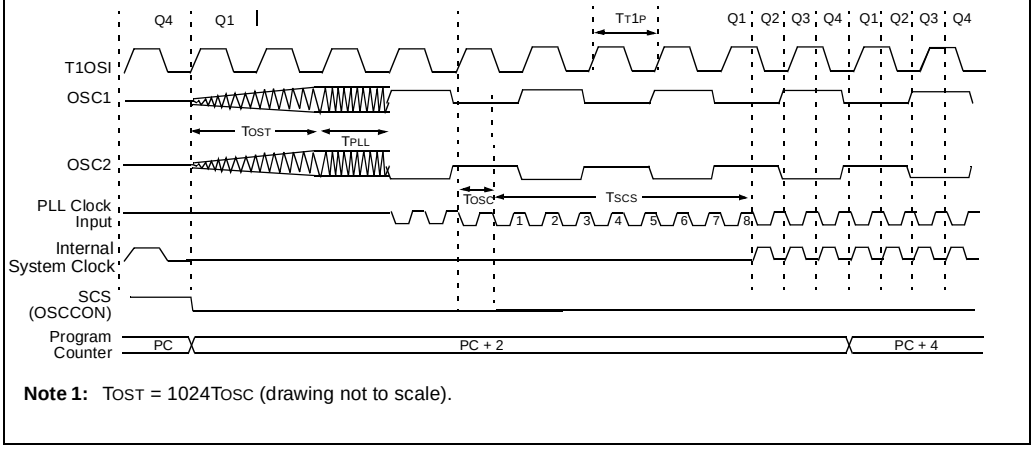
Note 1: Delay on internal system clock is eight oscillator cycles for synchronization.
2: The T1OSCEN bit is set.
3: The OSCSEN configuration bit is cleared.

Figure 2-13: Transition Between Timer1 and OSC1 Waveform (HS, XT, LP)



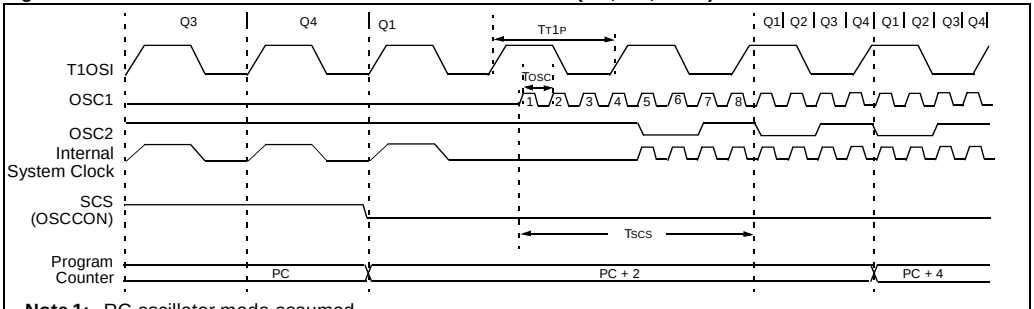
Note 1: TOST = 1024TOSC (drawing not to scale).

Figure 2-14: Transition Between Timer1 and OSC1 Waveform (HS4)



Note 1: TOST = 1024TOSC (drawing not to scale).

Figure 2-15: Transition Between Timer1 and OSC1 Waveform (RC, EC, ECIO)



Note 1: RC oscillator mode assumed.

Additional delays may occur before switching from the low power clock source back to the main oscillator. The sequence of events that take place will depend upon the main oscillator setting in the configuration register (the mode of the main oscillator).

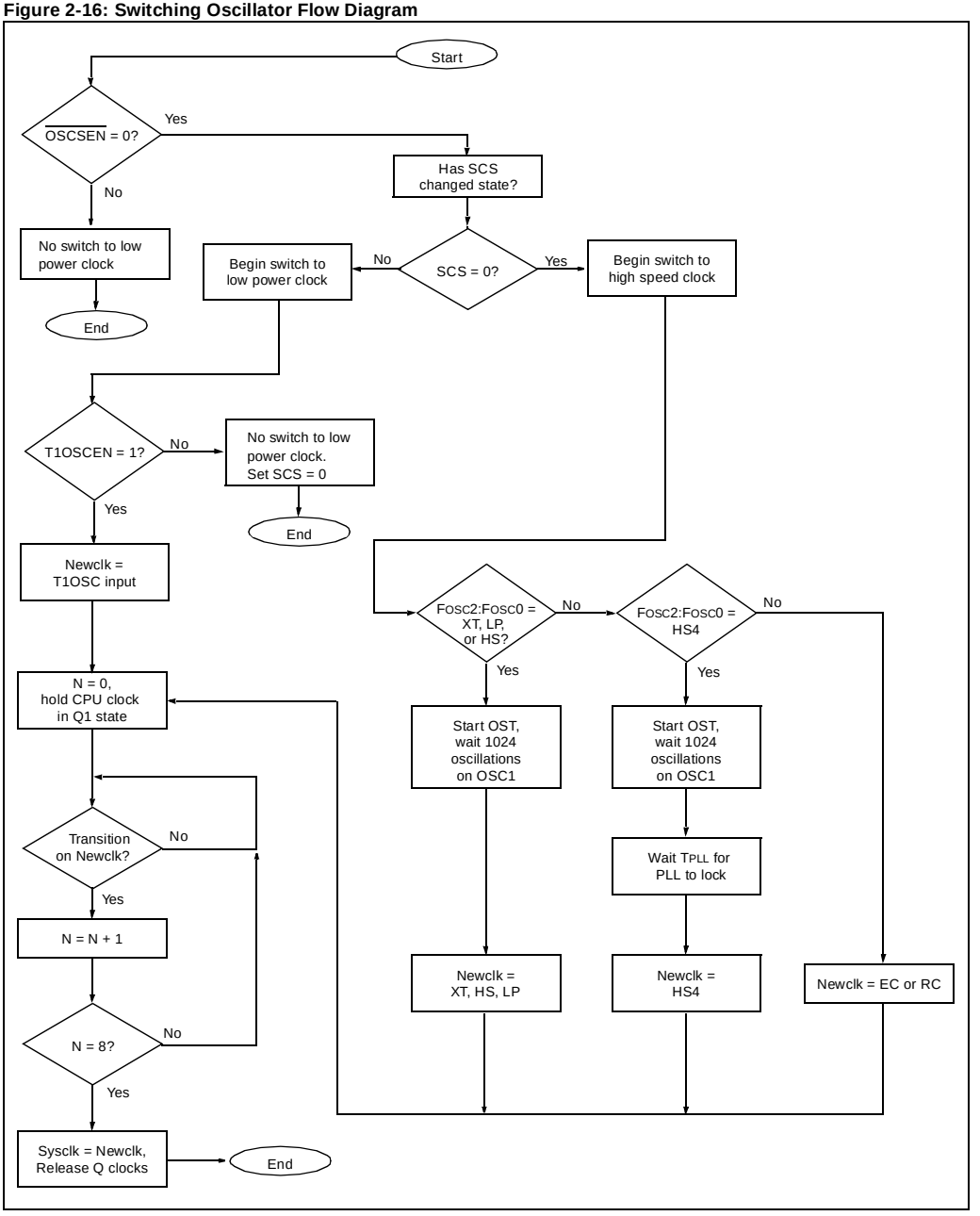
If the main oscillator is configured as a RC oscillator (RC, RCIO) or External Clock (EC, ECIO), then there is no oscillator start-up time. The transition from a low power clock to the main oscillator occurs after 8 clock cycles are counted on OSC1.

If the main oscillator is configured as a crystal (HS4, HS, XT or LP), then the transition will take place after an oscillator start-up time (TOST).

If the main oscillator is configured as a crystal with PLL (HS4) enabled, then the transition will take place after an oscillator start-up time (TOST) plus an additional PLL time-out, TPLL (see **"Electrical Specifications"** section, parameter 32). This is necessary because the crystal oscillator had been powered down until the time of the transition. In order to provide the system with a reliable clock when the change-over has occurred, the clock will not be released to the change-over circuit until the oscillator start-up time has expired. The additional TPLL time is required after oscillator start-up to allow the phase lock loop ample time to lock to the incoming oscillator frequency from OSC1.

A flow diagram for switching between a low power clock and the default oscillator clock is shown in Figure2-16.

Figure 2-16: Switching Oscillator Flow Diagram



2.8 Effects of Sleep Mode on the On-Chip Oscillator

When the device executes a SLEEP instruction, the on-chip clocks and oscillator are turned off and the device is held at the beginning of an instruction cycle (Q1 state). With the oscillator off, the OSC1 and OSC2 signals will stop oscillating. Since all the transistor switching currents have been removed, SLEEP mode achieves the lowest current consumption of the device (only leakage currents). Enabling any on-chip feature that will operate during SLEEP will increase the current consumed. The user can wake from SLEEP through external reset, Watchdog Timer Reset or through an interrupt.

See Table 3-1 in the **"Reset"** section for time-outs due to SLEEP and $\overline{\text{MCLR}}$ reset.

Table 2-4: OSC1 and OSC2 Pin States in Sleep Mode

OSC Mode	OSC1 Pin	OSC2 Pin
RC	Floating, external resistor should pull high	At logic low
RCIO	Floating, external resistor should pull high	Configured as I/O pin
ECIO	Floating	Configured as I/O pin
EC	Floating	At logic low
LP, XT and HS	Feedback inverter disabled at quiescent voltage level	Feedback inverter disabled at quiescent voltage level
HS4	Feedback inverter disabled at quiescent voltage level	Feedback inverter disabled at quiescent voltage level

2.9 Effects of Device Reset on the On-Chip Oscillator

Device resets have no effect on the on-chip crystal oscillator circuitry. The oscillator will continue to operate as it does under normal execution. While in RESET, the device logic is held at the Q1 state so that when the device exits RESET, it is at the beginning of an instruction cycle. The OSC2 pin, when used as the external clockout (RC, EC mode), will be held low during RESET, and as soon as the $\overline{\text{MCLR}}$ pin is at V_{IH} (input high voltage), the RC will start to oscillate.

See Table 3-1 in the **"Reset"** section for time-outs due to SLEEP and $\overline{\text{MCLR}}$ reset.

2.9.1 Power-up Delays

Power-up delays are controlled by two timers, so that no external reset circuitry is required for most applications. The delays ensure that the device is kept in RESET until the device power supply and clock are stable. For additional information on RESET operation, see the **"Reset"** section.

The Power-up Timer (PWRT) provides a fixed 72 ms delay on power-up due to POR or BOR, and keeps the part in RESET until the device power supply is stable. When a crystal is used (LP, XT, HS), the Oscillator Start-Up Timer (OST) keeps the chip in RESET until the PWRT timer delay has expired, allowing the crystal oscillator to stabilize on power up. The $\overline{\text{PWRTEN}}$ bit must be cleared for this time-out to occur.

When the PLL is enabled (HS4 oscillator mode), the Power-up Timer (PWRT) is used to keep the device in RESET for an extra nominal delay (T_{PLL}) above crystal mode. This delay ensures that the PLL is locked to the crystal frequency.

For additional information on RESET operation, see the **"Reset"** section.

2.10 Design Tips

Question 1: ***When looking at the OSC2 pin after power-up with an oscilloscope, there is no clock. What can cause this?***

Answer 1:

1. Executing a SLEEP instruction with no source for wake-up (such as, WDT, $\overline{\text{MCLR}}$, or an Interrupt). Verify that the code does not put the device to SLEEP without providing for wake-up. If it is possible, try waking it up with a low pulse on $\overline{\text{MCLR}}$. Powering up with $\overline{\text{MCLR}}$ held low will also give the crystal oscillator more time to start-up, but the Program Counter will not advance until the $\overline{\text{MCLR}}$ pin is high.
2. The wrong clock mode is selected for the desired frequency. For a blank device, the default oscillator is RCIO. Most parts come with the clock selected in the default RC mode, which will not start oscillation with a crystal or resonator. Verify that the clock mode has been programmed correctly.
3. The proper power-up sequence has not been followed. If a CMOS part is powered through an I/O pin prior to power-up, bad things can happen (latch up, improper start-up, etc.) It is also possible for brown-out conditions, noisy power lines at start-up, and slow V_{DD} rise times to cause problems. Try powering up the device with nothing connected to the I/O, and power-up with a known, good, fast-rise, power supply. Refer to the power-up information in the device data sheet for considerations on brown-out and power-up sequences.
4. The ***C1*** and ***C2*** capacitors attached to the crystal have not been connected properly or are not the correct values. Make sure all connections are correct. The device data sheet values for these components will usually get the oscillator running; however, they just might not be the optimal values for your design.

Question 2: ***The PICmicro device starts, but runs at a frequency much higher than the resonant frequency of the crystal.***

Answer 2:

The gain is too high for this oscillator circuit. Refer to subsection **2.4 "Crystal Oscillators/Ceramic Resonators"** to aid in the selection of ***C2*** (may need to be higher) ***Rs*** (may be needed) and clock mode (wrong mode may be selected). This is especially possible for low frequency crystals, like the common 32.768 kHz.

Question 3: ***The design runs fine, but the frequency is slightly off. What can be done to adjust this?***

Answer 3:

Changing the value of ***C1*** has some effect on the oscillator frequency. If a SERIES resonant crystal is used, it will resonate at a different frequency than a PARALLEL resonant crystal of the same frequency call-out. Ensure that you are using a PARALLEL resonant crystal.

Question 4: ***The board works fine, then suddenly quits or loses time.***

Answer 4:

Other than the obvious software checks that should be done to investigate losing time, it is possible that the amplitude of the oscillator output is not high enough to reliably trigger the oscillator input. Look at the ***C1*** and ***C2*** values and ensure that the the device configuration bits are correct for the desired oscillator mode.

Question 5: ***If I put an oscilloscope probe on an oscillator pin, I don't see what I expect. Why?***

Answer 5:

Remember that an oscilloscope probe has capacitance. Connecting the probe to the oscillator circuitry will modify the oscillator characteristics. Consider using a low capacitance (active) probe.

2.11 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced MCU family (that is they may be written for the Base-Line, Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the oscillator are:

Title	Application Note #
PICmicro Microcontrollers Oscillator Design Guide	AN588
Low Power Design using PICmicro Microcontrollers	AN606

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

2.12 Revision History

Revision A

This is the initial released revision of the Enhanced MCU oscillators description.

Section 3. Reset

HIGHLIGHTS

This section of the manual contains the following major topics:

3.1 Introduction

The reset logic is used to place the device into a known state. The source of the reset can be determined by reading the device status bits. The reset logic is designed with features that reduce system cost and increase system reliability.

Devices differentiate between various kinds of reset:

a) Power-on Reset (POR)
b) $\overline{\text{MCLR}}$ Reset during normal operation
c) $\overline{\text{MCLR}}$ Reset during SLEEP
d) WDT Reset (normal operation)
e) Programmable Brown-out Reset (BOR)
f) RESET Instruction
g) Stack Overflow Reset
h) Stack Underflow Reset

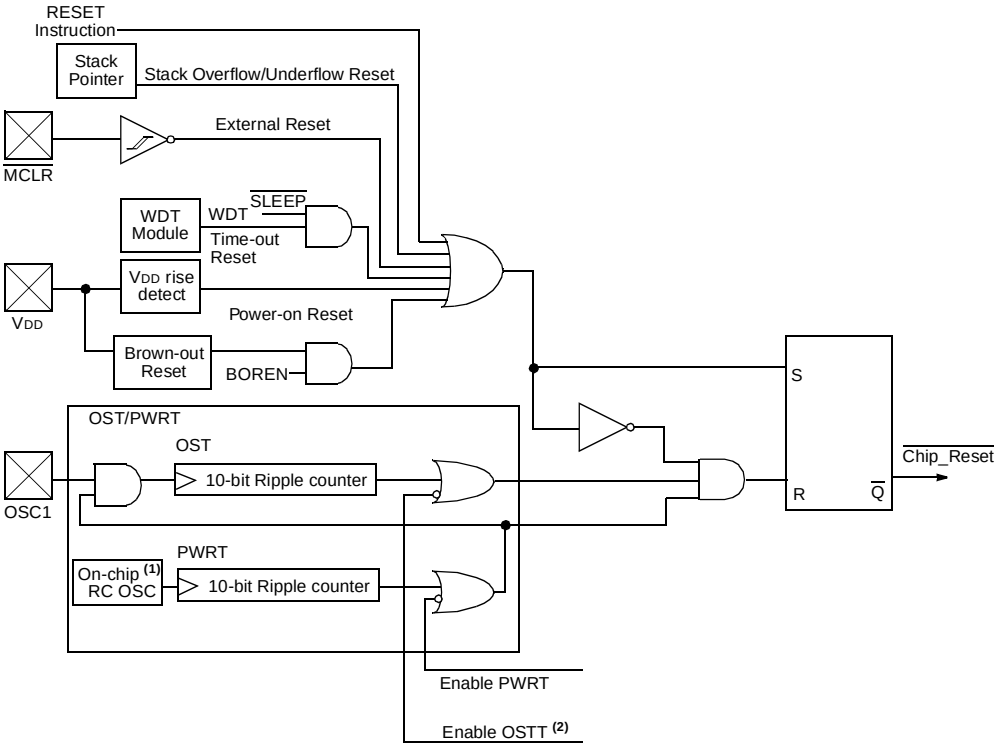
Most registers are unaffected by a reset; their status is unknown on POR and unchanged by all other resets. The other registers are forced to a “reset state” on Power-on Reset, $\overline{\text{MCLR}}$, WDT Reset, Brown-out Reset, $\overline{\text{MCLR}}$ Reset during SLEEP and by the RESET instruction.

Most registers are not affected by a WDT wake-up, since this is viewed as the resumption of normal operation. Status bits from the RCON register, $\overline{\text{RI}}$, $\overline{\text{TO}}$, $\overline{\text{PD}}$, $\overline{\text{POR}}$ and $\overline{\text{BOR}}$ are set or cleared differently in different reset situations as indicated in Table 3-3. These bits are used in software to determine the nature of the reset. See Table 3-4 for a full description of the reset states of all registers.

A simplified block diagram of the on-chip reset circuit is shown in Figure 3-1. This block diagram is a superset of reset features. To determine the features that are available on a specific device, please refer to the device’s Data Sheet.

Note:	While the Enhanced MCU is in a reset state, the internal phase clock is held at Q1 (beginning of an instruction cycle).

Figure 3-1: Simplified Block Diagram of On-chip Reset Circuit



Note 1: This is a separate oscillator from the RC oscillator of the CLKIN pin.
2: See Table 3-1 for time-out situations.

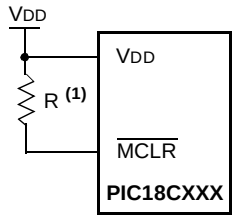
3.2 Resets and Delay Timers

The device has many sources for a device reset. Depending on the source of the reset, different delays may be initiated. These reset sources and the delays are discussed in the following sub-sections.

3.2.1 Power-on Reset (POR)

A Power-on Reset pulse is generated on-chip when VDD rise is detected. To take advantage of the POR, just tie the MCLR pin directly (or through a resistor) to VDD as shown in Figure 3-2. This will eliminate external RC components usually needed to create a Power-on Reset delay. A minimum rise time for VDD is required. See parameter D003 and parameter D004 in the **"Electrical Specifications"** section for details.

Figure 3-2: Using On-Chip POR

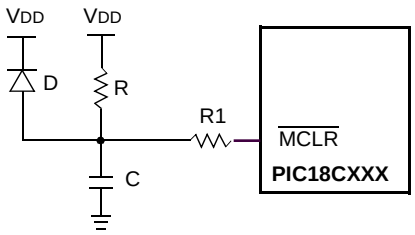


Note 1: The resistor is optional.

When the device exits the reset condition (begins normal operation), the device operating parameters (voltage, frequency, temperature, etc.) must be within their operating ranges, otherwise the device will not function correctly. Ensure the delay is long enough to get all operating parameters within specification.

Figure 3-3 shows a possible POR circuit for a slow power supply ramp up. The external Power-on Reset circuit is only required if the device would exit reset before the device VDD is in the valid operating range. The diode, D, helps discharge the capacitor quickly when VDD powers down.

Figure 3-3: External Power-on Reset Circuit (For Slow VDD Power-up)



Note 1: R < 40 kΩ is recommended to ensure that the voltage drop across R does not violate the device's electrical specification.

2: R1 = 100Ω to 1 kΩ will limit any current flowing into MCLR from external capacitor C in the event of MCLR/VPP pin breakdown due to Electrostatic Discharge (ESD) or Electrical Overstress (EOS).

3.2.2 Power-up Timer (PWRT)

The Power-up Timer provides a delay on Power-on Reset (POR) or Brown-out Reset (BOR). See parameter D033 in the **"Electrical Specifications"** section. The Power-up Timer operates on a dedicated internal RC oscillator. The device is kept in reset as long as the PWRT is active. The PWRT delay allows VDD to rise to an acceptable level. A configuration bit (PWRTEN) is provided to enable/disable the Power-up Timer.

Note:	Some devices require the Power-up Timer to be enabled when the Brown-out Reset circuitry is enabled. Please refer to the device data sheet for requirements.

The power-up time delay will vary from device to device due to VDD, temperature and process variations. See DC parameters for details.

3.2.3 Oscillator Start-up Timer (OST)

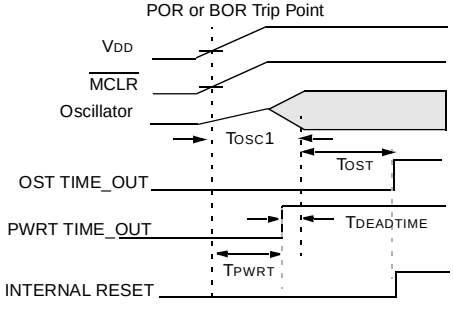
The Oscillator Start-Up Timer (OST) provides a 1024 oscillator cycle delay (from OSC1 input) (parameter 32) after the PWRT delay is over. This ensures that the crystal oscillator or resonator has started and is stable.

The OST time-out is invoked only for XT, LP and HS modes, on Power-on Reset, Brown-out Reset, wake-up from SLEEP, or on a transition from Timer1 input clock as the system clock to the oscillator as the system clock by clearing the SCS bit. The oscillator start-up timer is disabled for all resets and wake-ups in RC and EC modes. (See Table 3-1)

The OST counts the oscillator pulses on the OSC1/CLKIN pin. The counter only starts incrementing after the amplitude of the signal reaches the oscillator input thresholds. This delay allows the crystal oscillator or resonator to stabilize before the device exits the OST delay. The length of the time-out is a function of the crystal/resonator frequency.

Figure 3-4 shows the operation of the OST circuit in conjunction with the power-up timer. For low frequency crystals, this start-up time can become quite long. That is because the time it takes the low frequency oscillator to start oscillating is longer than the power-up timer's delay. The time from when the power-up timer times out to when the oscillator starts to oscillate is a dead time. There is no minimum or maximum time for this dead time (*TDEADTIME*), and is dependent on the time for the oscillator circuitry to have "good" oscillations.

Figure 3-4: Oscillator Start-up Time



TOSC1 = Time for the crystal oscillator to react to an oscillation level detectable by the Oscillator Start-up Timer (OST).

TOST = 1024TOSC.

3.2.3.1 PLL Lock Time-out

When the PLL is enabled, the time-out sequence following a Power-on Reset is different from other oscillator modes. A portion of the Power-up Timer is used to provide a fixed time-out that is sufficient for the PLL to lock to the main oscillator frequency. This PLL lock time-out TPLL (2ms nominal, Parameter 7 in the **"Electrical Specifications"** section) follows the Oscillator Start-up Time-out (OST).

3.2.4 Power-up Sequence

On power-up, the time-out sequence is as follows: First the internal POR is detected, then, if enabled, the PWRT time-out is invoked. After the PWRT time-out is over, the OST is activated. The total time-out will vary based on oscillator configuration and $\overline{\text{PWRTEN}}$ bit status. For example, in RC mode with the $\overline{\text{PWRTEN}}$ bit set (PWRT disabled), there will be no time-out at all. Figure 3-5, Figure 3-6 and Figure 3-7 depict time-out sequences.

Since the time-outs occur from the internal POR pulse, if $\overline{\text{MCLR}}$ is kept low long enough, the time-outs will expire. Bringing $\overline{\text{MCLR}}$ high will begin execution immediately (Figure 3-7). This is useful for testing purposes or to synchronize more than one device operating in parallel.

If the device voltage is not within the electrical specifications by the end of a time-out, the $\overline{\text{MCLR}}$/VPP pin must be held low until the voltage is within the device specification. The use of an external RC delay is sufficient for many of these applications.

On wake-up from sleep, the OST is activated for various oscillator configurations. When the PLL is activated in HS mode, an additional delay called TPLL (2 ms nominal) is added to the OST time-out to allow the necessary lock time for the PLL. See parameter D003 in the **"Electrical Specifications"** section for details.

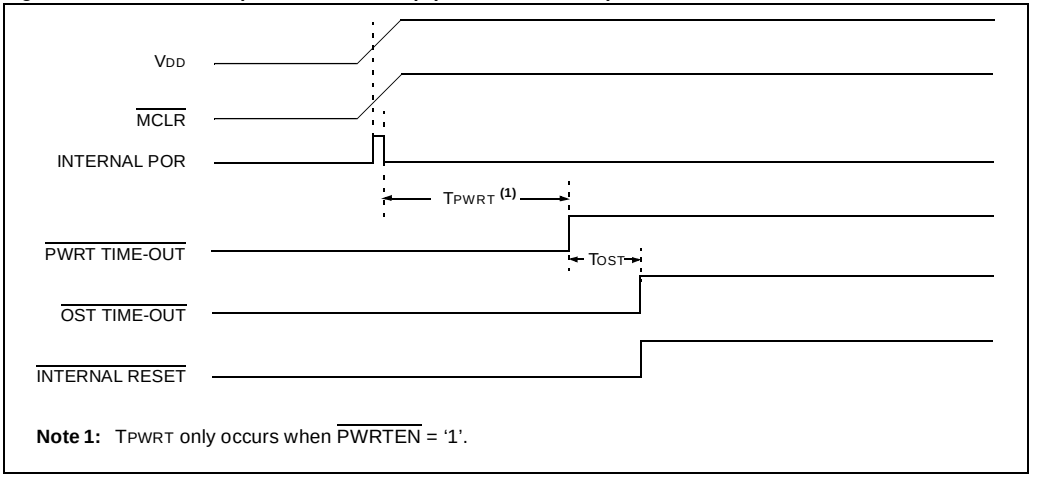
Table 3-1 shows the time-outs that occur in various situations, while Figure 3-5 through Figure 3-8 show four different cases that can happen on powering up the device.

Table 3-1: Time-out in Various Situations

Oscillator Configuration	Power-up (2) or Brown-Out (3)		Wake-up from SLEEP or Oscillator Switch
	$\overline{\text{PWRTEN}}$ = 0	$\overline{\text{PWRTEN}}$ = 1	
HS with PLL enabled (1)	72 ms + 1024Tosc + 2ms	1024Tosc + 2 ms	1024Tosc + 2 ms
HS, XT, LP	72 ms + 1024Tosc	1024Tosc	1024Tosc
EC	72 ms	—	—
External RC	72 ms	—	—

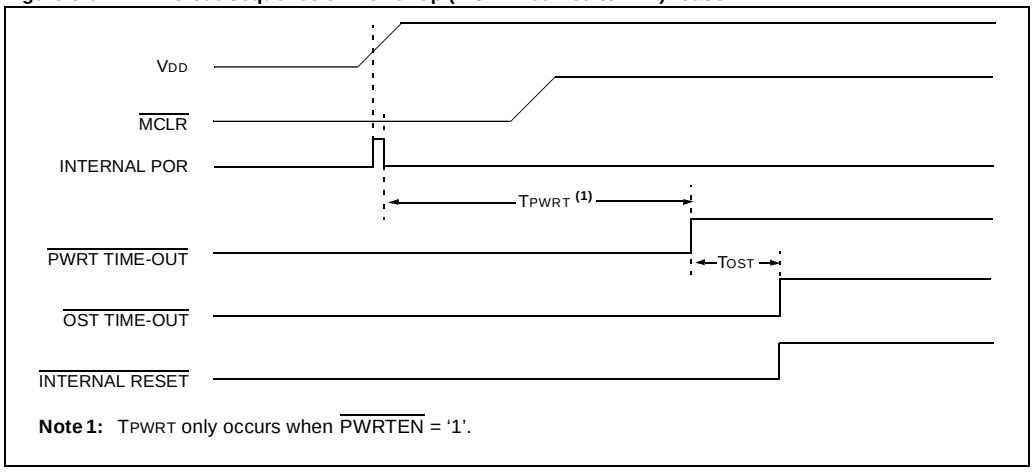
Note 1: 2 ms = Nominal time required for the PLL to lock. See the **"Electrical Specifications"** section.
2: 72 ms is the nominal power-up timer delay. See the **"Electrical Specifications"** section.
3: It is recommended that the power-up timer is enabled when using the Brown-out Reset module.

Figure 3-5: Time-out Sequence on Power-up ($\overline{\text{MCLR}}$ Tied to VDD)



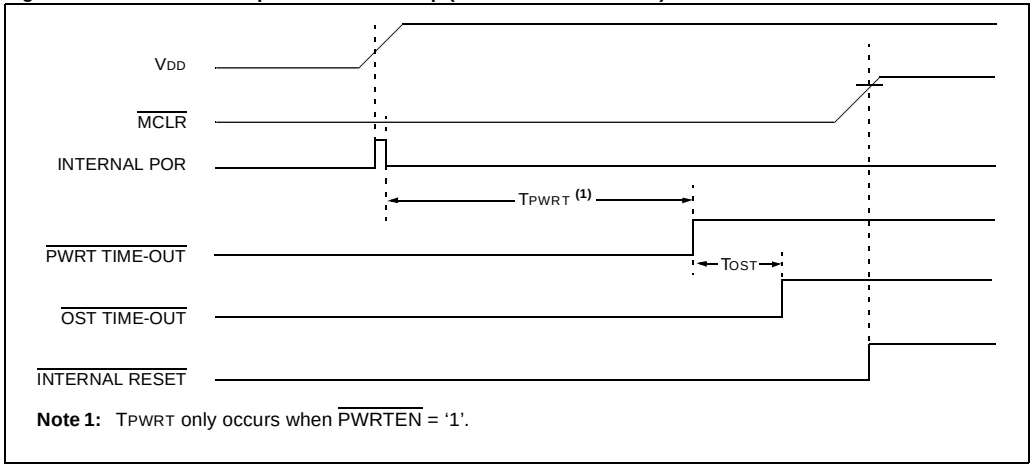
Note 1: TPWRT only occurs when $\overline{\text{PWRTEN}}$ = '1'.

Figure 3-6: Time-out Sequence on Power-up (MCLR not Tied to VDD): Case 1



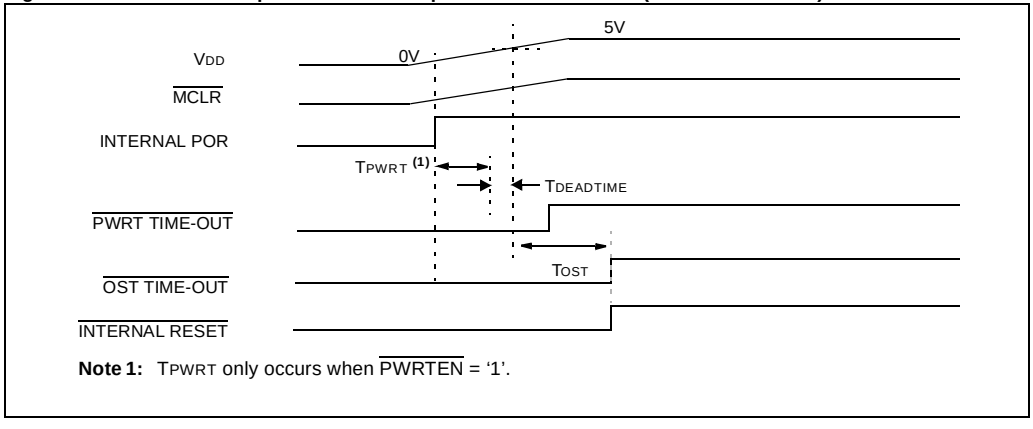
Note 1: TPWRT only occurs when PWRTEN = '1'.

Figure 3-7: Time-out Sequence on Power-up (MCLR not Tied to VDD): Case 2



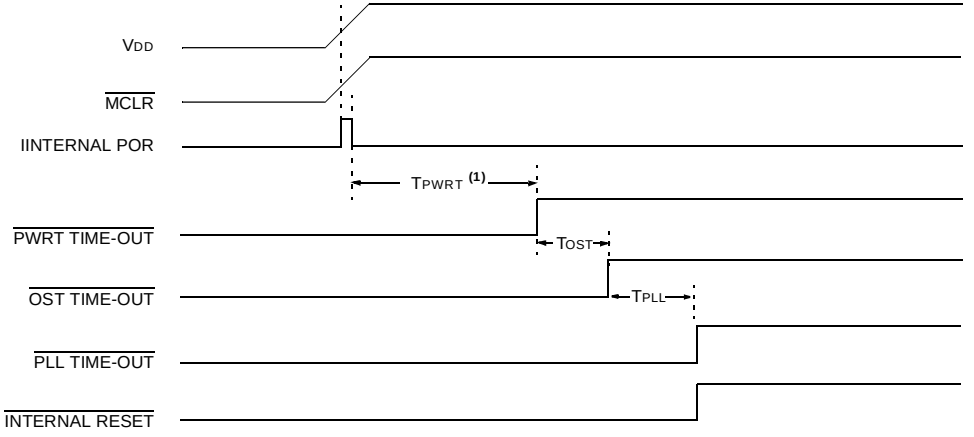
Note 1: TPWRT only occurs when PWRTEN = '1'.

Figure 3-8: Time-out Sequence on Power-up with Slow Rise Time (MCLR Tied to VDD)



Note 1: TPWRT only occurs when PWRTEN = '1'.

Figure 3-9: Time-out Sequence on POR w/ PLL Enabled ($\overline{MCLR}$ Tied to VDD)



TOST = 1024 clock cycles.

TPLL = PLL lock time.

Note 1: TPWRT only occurs when $\overline{PWRTEN}$ = '1'.

3.2.5 Brown-Out Reset (BOR)

On-chip Brown-out Reset circuitry places the device into reset when the device voltage (V_{DD}) falls below a trip point (V_{BOR}). This ensures that the device does not continue program execution outside the valid voltage operation range of the device. Brown-out resets are typically used in AC line applications (such as appliances) or large battery applications where large loads may be switched in (such as automotive).

Appliances encounter brown-out situations during plug-in and online voltage dip. Automotive electronics encounter brown-out when the ignition key is turned. In these application scenarios, the device voltage temporarily falls below the specified operating minimum.

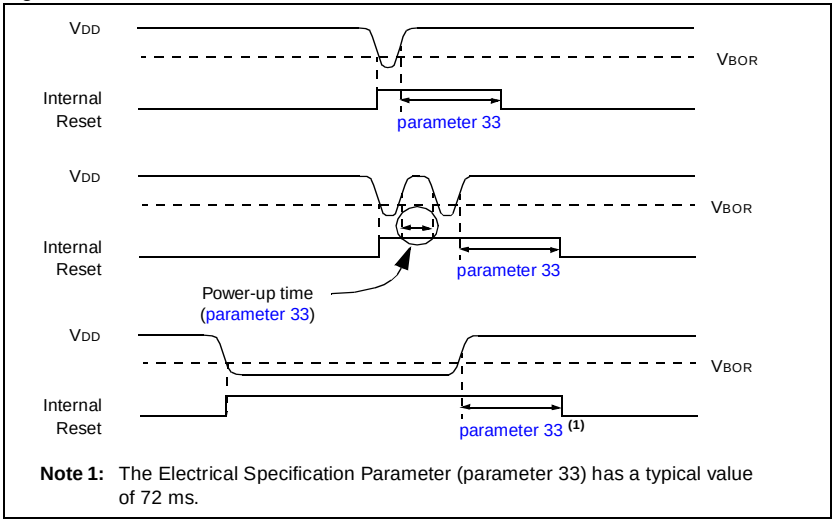
If the brown-out circuit meets the current consumption requirements of the system, it may also be used as a voltage supervisory function.

Note:	Before using the on-chip brown-out for a voltage supervisory function (monitor battery decay), please review the electrical specifications to ensure that they meet your requirements.

Figure 3-10 shows typical brown-out situations. The Brown-out Reset module is enabled by default. To disable the module, the BOREN configuration bit must be cleared at device programming.

Note 1:	It is recommended that the power-up timer be enabled when using the BOR module. The power-up timer is enabled by programming the PWRTEN configuration bit to '0'.
Note 2:	Some devices require the Power-up Timer to be enabled when the Brown-out Reset circuitry is enabled. Please refer to the device data sheet for requirements.

Figure 3-10: Brown-Out Situations



Note 1: The Electrical Specification Parameter (parameter 33) has a typical value of 72 ms.

3.2.5.1 BOR Operation

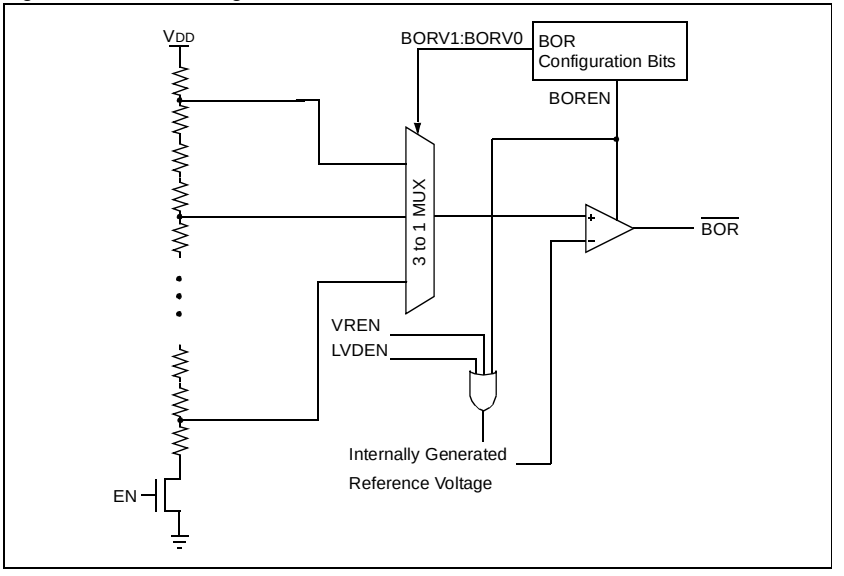
The BOREN configuration bit can disable (if clear/programmed) or enable (if set) the Brown-out Reset circuitry. If VDD falls below VBOR (parameter D005 in the **"Electrical Specifications"** section), for greater than the Brown-out Pulse Width Time (TBOR), parameter 35, the brown-out situation will reset the chip. A reset is not guaranteed to occur if VDD falls below VBOR for less than parameter 35.

The chip will remain in Brown-out Reset until VDD rises above VBOR. After which, the Power-up Timer is invoked and will keep the chip in reset an additional time delay (parameter 33). If VDD drops below VBOR while the Power-up Timer is running, the chip will go back into Reset and the Power-up Timer will be re-initialized. Once VDD rises above VBOR, the Power-up Timer will again start a time delay.

When the BOREN bit is set, all voltages below VBOR will hold the device in the reset state. This includes during the power-up sequence.

The brown-out trip point is user programmable at time of device programming. Figure 3-11 is a block diagram for the BOR circuit.

Figure 3-11: Block Diagram of BOR Circuit



The Brown-out Reset circuit has four available reset trip point voltages. The device selected determines which trip points make sense in an application. All devices have the trip points of 4.2V and 4.5V available. PIC18LCXXX devices add two more trip points. The first is 2.7V, while the second is dependent on the minimum operating voltage of that device. This means that the lowest trip point voltage will either be 2.5V or 1.8V. Table 3-2 shows the state of the configuration bits (BORV1:BORV0) and the BOR trip points that they select.

Table 3-2: Example BOR Trip Point Levels

BORV1:BORV0 Configuration Bits	Minimum Voltage Trip Point	Maximum Voltage Trip Point	Comment
1 1	1.8 V	1.86 V	PIC18LCXXX Devices (w/ V_{DDMIN} = 1.8V)
1 1	2.5 V	2.58 V	PIC18LCXXX Devices (w/ $V_{DDMIN} \geq$ 2.0V)
1 0	2.7 V	2.78 V	PIC18LCXXX Devices
0 1	4.2 V	4.33 V	All Devices
0 0	4.5 V	4.64 V	All Devices

Note: The minimum voltage at which the Brown-out Reset trip point can occur should be in the valid operating voltage range of the device.

The BOR is programmable to ensure that the BOR can be optimized to the voltage-frequency of the device, since the minimum device V_{DD} value will depend on the frequency of operation. For example, V_{DD} min. at 40 MHz may be 4.2V, whereas at 2 MHz it may be 1.8V.

3.2.5.2 Current Implications for BOR Operation

There are three components to the current consumption of the BOR operation. These are:

1. Current from Internal Reference Voltage
2. Current from BOR comparator
3. Current from resistor ladder

The Internal Reference Voltage is also used by the Low Voltage Detect circuitry and the A/D voltage references. The resistor ladder is also used by the Low Voltage Detect circuitry. If the Low Voltage Detect is enabled, then only the additional current of the comparator is added for enabling the BOR feature.

When the module is enabled, the BOR comparator and voltage divider are enabled and consume static current. The **"Electrical Specifications"** section parameter 32 gives the current specification.

The Brown-out Comparator circuit consumes current when enabled. To eliminate this current consumption, the Brown-out Reset can be disabled by programming the Brown-out Reset Enable configuration bit (BOREN) to '0'.

3.2.5.3 BOR Initialization

The BOR module must be enabled and programmed through the device configuration bits. These include BOREN, which enables or disables the module, and BORV1:BORV0, which set the BOR voltage.

3.2.5.4 External Brown-Out Reset Circuits

There are some applications where the device's programmable Brown-out Reset trip point levels may still not be at the desired level for the application.

Figure 3-12 shows a circuit for external brown-out protection using the MCP100 device.

Figure 3-13 and Figure 3-14 are two examples of external circuitry that may be implemented. Each option needs to be evaluated to determine if they match the requirements of the application.

Figure 3-12: External Brown-Out Protection Using the MCP100

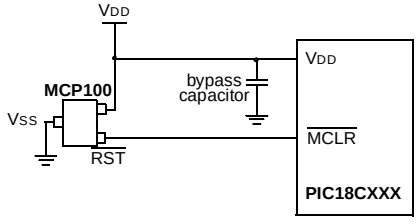
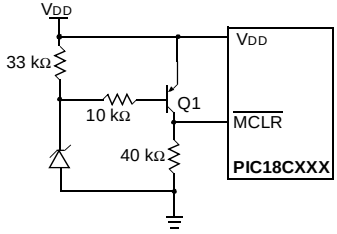
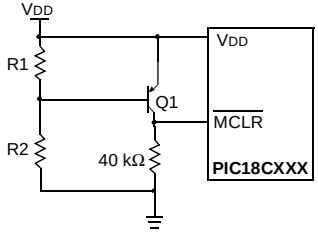


Figure 3-13: External Brown-Out Protection Circuit 1



Note 1: Internal Brown-out Reset circuitry should be disabled when using this circuit.
2: Resistors should be adjusted for the characteristics of the transistor.
3: This circuit will activate reset when VDD goes below (Vz + 0.7V) where Vz = Zener voltage.

Figure 3-14: External Brown-Out Protection Circuit 2



Note 1: This circuit is less expensive, but less accurate. Transistor Q1 turns off when VDD is below a certain level such that:

$$V_{DD} \cdot \frac{R1}{R1 + R2} = 0.7V$$

2: Internal Brown-out Reset circuitry should be disabled when using this circuit.
3: Resistors should be adjusted for the characteristics of the transistor.

3.3 Registers and Status Bit Values

Table 3-3 shows the significance of the device status bits and the initialization conditions for the RCON register. Table 3-4 shows the reset conditions for the Special Function Registers.

Register 3-1 shows the bits of the RCON register and Table 3-3 shows the initialization values.

Register 3-1: RCON Register Bits and Positions

R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-u
IPEN	LWRT	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

Table 3-3: Status Bits, Their Significance, and the Initialization Condition for RCON Register

Condition	Program Counter	RCON Register	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$	STKFUL	STKUNF
Power-on Reset	0000h	00-1 1100	1	1	1	0	u	u	u
$\overline{MCLR}$ Reset during normal operation	0000h	00-u uuuu	u	u	u	u	u	u	u
Software Reset during normal operation	0000h	0u-0 uuuu	0	u	u	u	u	u	u
Stack Overflow Reset during normal operation	0000h	0u-u uu11	u	u	u	u	u	u	1
Stack Underflow Reset during normal operation	0000h	0u-u uu11	u	u	u	u	u	1	u
$\overline{MCLR}$ Reset during SLEEP	0000h	00-u 10uu	u	1	0	u	u	u	u
WDT Reset	0000h	0u-u 01uu	1	0	1	u	u	u	u
WDT Wake-up	PC + 2	uu-u 00uu	u	0	0	u	u	u	u
Brown-out Reset	0000h	0u-1 11u0	1	1	1	1	0	u	u
Interrupt Wake-up from SLEEP	PC + 2 (1)	uu-u 00uu	u	1	0	u	u	u	u

Legend: u = unchanged, x = unknown, - = unimplemented bit read as '0'.

Note 1: When the wake-up is due to an interrupt and the GIEH or GIEL bits are set, the PC is loaded with the interrupt vector (0008h or 0018h).

Table 3-4: Initialization Conditions for SFR Registers

Register	Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets WDT Reset Reset Instruction Stack Resets	Wake-up via WDT or Interrupt
TOSU	---0 0000	---0 0000	---0 uuuu (3)
TOSH	0000 0000	0000 0000	uuuu uuuu (3)
TOSL	0000 0000	0000 0000	uuuu uuuu (3)
STKPTR	00-0 0000	00-0 0000	uu-u uuuu (3)
PCLATU	---0 0000	---0 0000	---u uuuu
PCLATH	0000 0000	0000 0000	uuuu uuuu
PCL	0000 0000	0000 0000	PC + 2 (2)
TBLPTRU	--00 0000	--00 0000	--uu uuuu
TBLPTRH	0000 0000	0000 0000	uuuu uuuu
TBLPTRL	0000 0000	0000 0000	uuuu uuuu
TABLAT	0000 0000	0000 0000	uuuu uuuu
PRODH	xxxx xxxx	uuuu uuuu	uuuu uuuu
PRODL	xxxx xxxx	uuuu uuuu	uuuu uuuu
INTCON	0000 000x	0000 000u	uuuu uuuu (1)
INTCON2	1111 -1-1	1111 -1-1	uuuu -u-u (1)
INTCON3	11-0 0-00	11-0 0-00	uu-u u-uu (1)
INDF0	N/A	N/A	N/A
POSTINC0	N/A	N/A	N/A
POSTDEC0	N/A	N/A	N/A
PREINC0	N/A	N/A	N/A
PLUSW0	N/A	N/A	N/A
FSR0H	---- 0000	---- 0000	---- uuuu
FSR0L	xxxx xxxx	uuuu uuuu	uuuu uuuu
WREG	xxxx xxxx	uuuu uuuu	uuuu uuuu
INDF1	N/A	N/A	N/A
POSTINC1	N/A	N/A	N/A
POSTDEC1	N/A	N/A	N/A
PREINC1	N/A	N/A	N/A

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition.

Note 1: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the PC is loaded with the interrupt vector (0008h or 0018h).

3: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the TOSU, TOSH and TOSL are updated with the current value of the PC. The STKPTR is modified to point to the next location in the hardware stack.

4: The long write enable is only reset on a POR or $\overline{\text{MCLR}}$ reset.

5: The bits in the PIR, PIE, and IPR registers are device dependent. Their function and location may change from device to device.

Table 3-4: Initialization Conditions for SFR Registers (Continued)

Register	Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets WDT Reset Reset Instruction Stack Resets	Wake-up via WDT or Interrupt
PLUSW1	N/A	N/A	N/A
FSR1H	---- 0000	---- 0000	---- uuuu
FSR1L	xxxx xxxx	uuuu uuuu	uuuu uuuu
BSR	---- 0000	---- 0000	---- uuuu
INDF2	N/A	N/A	N/A
POSTINC2	N/A	N/A	N/A
POSTDEC2	N/A	N/A	N/A
PREINC2	N/A	N/A	N/A
PLUSW2	N/A	N/A	N/A
FSR2H	---- 0000	---- 0000	---- uuuu
FSR2L	xxxx xxxx	uuuu uuuu	uuuu uuuu
STATUS	---x xxxx	---u uuuu	---u uuuu
TMR0H	xxxx xxxx	uuuu uuuu	uuuu uuuu
TMR0L	xxxx xxxx	uuuu uuuu	uuuu uuuu
T0CON	1111 1111	1111 1111	uuuu uuuu
OSCCON	---- ---0	---- ---0	---- ---u
LVDCON	--00 0101	--00 0101	--uu uuuu
WDTCON	---- ---0	---- ---0	---- ---u
RCON (4)	00-1 11q0	00-1 qquu	uu-u qquu
TMR1H	xxxx xxxx	uuuu uuuu	uuuu uuuu
TMR1L	xxxx xxxx	uuuu uuuu	uuuu uuuu
T1CON	0-00 0000	u-uu uuuu	u-uu uuuu
TMR2	xxxx xxxx	uuuu uuuu	uuuu uuuu
PR2	1111 1111	1111 1111	1111 1111
T2CON	-000 0000	-000 0000	-uuu uuuu
SSPBUF	xxxx xxxx	uuuu uuuu	uuuu uuuu
SSPADD	0000 0000	0000 0000	uuuu uuuu
SSPSTAT	0000 0000	0000 0000	uuuu uuuu
SSPCON1	0000 0000	0000 0000	uuuu uuuu
SSPCON2	0000 0000	0000 0000	uuuu uuuu

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition.

Note 1: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the PC is loaded with the interrupt vector (0008h or 0018h).

3: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the TOSU, TOSH and TOSL are updated with the current value of the PC. The STKPTR is modified to point to the next location in the hardware stack.

4: The long write enable is only reset on a POR or $\overline{\text{MCLR}}$ reset.

5: The bits in the PIR, PIE, and IPR registers are device dependent. Their function and location may change from device to device.

Table 3-4: Initialization Conditions for SFR Registers (Continued)

Register	Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets WDT Reset Reset Instruction Stack Resets	Wake-up via WDT or Interrupt
ADRESH	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADRESL	xxxx xxxx	uuuu uuuu	uuuu uuuu
ADCON0	0000 0000	0000 0000	uuuu uuuu
ADCON1	--0- 0000	--0- 0000	--u- uuuu
CCPR1H	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR1L	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP1CON	--00 0000	--00 0000	--uu uuuu
CCPR2H	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCPR2L	xxxx xxxx	uuuu uuuu	uuuu uuuu
CCP2CON	--00 0000	--00 0000	--uu uuuu
TMR3H	xxxx xxxx	uuuu uuuu	uuuu uuuu
TMR3L	xxxx xxxx	uuuu uuuu	uuuu uuuu
T3CON	0000 0000	uuuu uuuu	uuuu uuuu
SPBRG	xxxx xxxx	uuuu uuuu	uuuu uuuu
RCREG	xxxx xxxx	uuuu uuuu	uuuu uuuu
TXREG	xxxx xxxx	uuuu uuuu	uuuu uuuu
TXSTA	0000 -01x	0000 -01u	uuuu -uuu
RCSTA	0000 000x	0000 000u	uuuu uuuu
IPR2 **(5)**	1	1	u
PIR2 **(5)**	0	0	u **(1)**
PIE2 **(5)**	0	0	u
IPR1 **(5)**	1	1	u
PIR1 **(5)**	0	0	u **(1)**
PIE1 **(5)**	0	0	u

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition.

Note 1: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the PC is loaded with the interrupt vector (0008h or 0018h).

3: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the TOSU, TOSH and TOSL are updated with the current value of the PC. The STKPTR is modified to point to the next location in the hardware stack.

4: The long write enable is only reset on a POR or $\overline{\text{MCLR}}$ reset.

5: The bits in the PIR, PIE, and IPR registers are device dependent. Their function and location may change from device to device.

Table 3-4: Initialization Conditions for SFR Registers (Continued)

Register	Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets WDT Reset Reset Instruction Stack Resets	Wake-up via WDT or Interrupt
TRISE	0000 -111	0000 -111	uuuu -uuu
TRISD	1111 1111	1111 1111	uuuu uuuu
TRISC	1111 1111	1111 1111	uuuu uuuu
TRISB	1111 1111	1111 1111	uuuu uuuu
TRIS	-111 1111	-111 1111	-uuu uuuu
LATE	---- -xxx	---- -uuu	---- -uuu
LATD	xxxx xxxx	uuuu uuuu	uuuu uuuu
LATC	xxxx xxxx	uuuu uuuu	uuuu uuuu
LATB	xxxx xxxx	uuuu uuuu	uuuu uuuu
LATA	-xxx xxxx	-uuu uuuu	-uuu uuuu
PORTE	---- -000	---- -000	---- -uuu
PORTD	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTC	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTB	xxxx xxxx	uuuu uuuu	uuuu uuuu
PORTA	-x0x 0000	-u0u 0000	-uuu uuuu

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0', q = value depends on condition.

Note 1: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).

2: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the PC is loaded with the interrupt vector (0008h or 0018h).

3: When the wake-up is due to an interrupt and the GIEL or GIEH bit is set, the TOSU, TOSH and TOSL are updated with the current value of the PC. The STKPTR is modified to point to the next location in the hardware stack.

4: The long write enable is only reset on a POR or $\overline{\text{MCLR}}$ reset.

5: The bits in the PIR, PIE, and IPR registers are device dependent. Their function and location may change from device to device.

3.3.1 Reset Control (RCON) Register

The Reset Control (RCON) register contains flag bits to allow differentiation between resets. The Reset Control register has seven bits.

The $\overline{\text{POR}}$ (Power-on Reset) bit is cleared on a Power-on Reset and is unaffected otherwise. The user sets this bit following a Power-on Reset. On subsequent resets, if the $\overline{\text{POR}}$ bit is clear (= '0'), it will indicate that a Power-on Reset must have occurred.

Note:	The state of the $\overline{\text{BOR}}$ bit is unknown on Power-on Reset. It must be set by the user and checked on subsequent resets to see if the $\overline{\text{BOR}}$ bit is clear, indicating a brown-out has occurred. The $\overline{\text{BOR}}$ status bit is a "don't care" and is not necessarily predictable if the brown-out circuit is disabled (by clearing the BOREN bit in the Configuration register).

The power-down bit ($\overline{\text{PD}}$) provides indication if the device was placed into sleep mode. It is set by a power-up, a CLRWDT instruction or by user software. The $\overline{\text{PD}}$ bit is cleared when the SLEEP instruction is executed or by user software.

Register 3-2: RCON Register

R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-u
IPEN	LWRT	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
bit 7							bit 0

bit 7 **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts

bit 6 **LWRT:** Long Write Enable bit
1 = Enable Table Writes to internal program memory
Once this bit is set, it can only be cleared by a POR or $\overline{\text{MCLR}}$ reset.
0 = Disable Table Writes to internal program memory; Table Writes only to external program memory.

bit 5 **Unimplemented:** Read as '0'

bit 4 **$\overline{\text{RI}}$:** Reset Instruction Flag bit
1 = The RESET instruction was not invoked
0 = The RESET instruction was executed
(must be set in software after the RESET instruction is executed)

bit 3 **$\overline{\text{TO}}$:** Time-out bit
1 = After power-up, CLRWDT instruction or SLEEP instruction
0 = A WDT time-out occurred

bit 2 **$\overline{\text{PD}}$:** Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 1 **$\overline{\text{POR}}$:** Power-on Reset Flag bit
1 = A Power-on Reset has not occurred
0 = A Power-on Reset occurred
(must be set in software after a Power-on Reset occurs)

bit 0 **$\overline{\text{BOR}}$:** Brown-out Reset Flag bit
1 = A Brown-out Reset has not occurred
0 = A Brown-out Reset occurred
(must be set in software after a Brown-out Reset or Power-on Reset occurs)

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

3.4 Design Tips

Question 1: ***With windowed devices, my system resets and operates properly. With an OTP device, my system does not operate properly.***

Answer 1:

The most common reason for this is that the windowed device has not had its window covered. The background light causes the device to power-up in a different state than would typically be seen in a device where no light is present. In most cases, all the General Purpose RAM and Special Function Registers were not initialized by the application software.

3.5 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is, they may be written for the Base-Line, the Mid-Range or High-End families), but the concepts are pertinent and could be used (with modification and possible limitations). The current application notes related to Resets are:

Title	Application Note #
Power-up Trouble Shooting	AN607
Power-up Considerations	AN522

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

3.6 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Reset description.

Section 4. Architecture

HIGHLIGHTS

This section of the manual contains the following major topics:

4.1 Introduction

The high performance of the PIC18CXXX devices can be attributed to a number of architectural features commonly found in RISC microprocessors. These include:

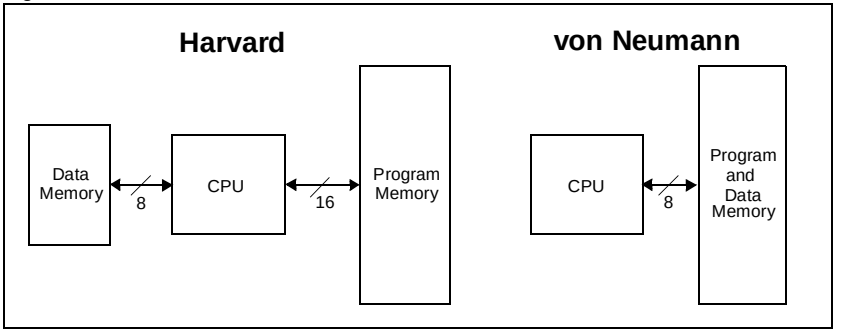
- Harvard architecture
- Long Word Instructions
- Single Word Instructions
- Single Cycle Instructions
- Instruction Pipelining
- Reduced Instruction Set
- Register File Architecture
- Orthogonal (Symmetric) Instructions

Figure 4-2 shows a general block diagram for PIC18CXXX devices.

Harvard Architecture:

Harvard architecture has the program memory and data memory as separate memories which are accessed from separate buses. This improves bandwidth over traditional von Neumann architecture in which program and data are fetched from the same memory using the same bus. To execute an instruction, a von Neumann machine must make one or more (generally more) accesses across the 8-bit bus to fetch the instruction. Then data may need to be fetched, operated on and possibly written. As can be seen from this description, the bus can become extremely congested. With a Harvard architecture, the instruction is fetched in a single instruction cycle (all 16 bits). While the program memory is being accessed, the data memory is on an independent bus and can be read and written. These separated busses allow one instruction to execute, while the next instruction is fetched. A comparison of Harvard and von Neumann architectures is shown in Figure 4-1.

Figure 4-1: Harvard vs. von Neumann Block Architectures



Long Word Instructions:

Long word instructions have a wider (more bits) instruction bus than the 8-bit data memory bus. This is possible because the two buses are separate. This allows instructions to be sized differently than the 8-bit wide data word and allows a more efficient use of the program memory, since the program memory width is optimized to the architectural requirements.

Single Word Instructions:

Single word instruction opcodes are 16-bits wide making it possible to have all but a few instructions be single word instructions. A 16-bit wide program memory access bus fetches a 16-bit instruction in a single cycle. With single word instructions, the number of words of program memory locations equals the number of instructions for the device. This means that all locations are valid instructions.

Typically in the von Neumann architecture, most instructions are multi-byte. In general, a device with 4 Kbytes of program memory would allow approximately 2K of instructions. This 2:1 ratio is generalized and dependent on the application code. Since each instruction may take multiple bytes, there is no assurance that each location is a valid instruction.

Double Word Instructions:

Some operations require more information then can be stored in the 16 bits of a program memory location. These operations require a double word instruction, and are therefore 32-bits wide. Instructions that require this second instruction word are:

- Memory to memory move instruction (12 bits for each RAM address)
  - `MOVFF  SourceReg, DestReg`
- Literal value to FSR move instruction (12 bits for data and 2 bits for FSR to load)
  - `LFSR  FSR#, Address`
- Call and goto operations (20 bits for address)
  - `CALL  Address`
  - `GOTO  Address`

The first word indicates to the CPU that the next program memory location is the additional information for this instruction and not an instruction. If the CPU tries to execute the second word of an instruction (due to a software modified PC pointing to that location as an instruction), the fetched data is executed as a `NOP`.

Double word instruction execution is not split between the two TCY cycles by an interrupt request. That is, when an interrupt request occurs during the execution of a double word instruction, the execution of the instruction is completed before the processor vectors to the interrupt address. The interrupt latency is preserved.

Instruction Pipeline:

The instruction pipeline is a two-stage pipeline that overlaps the fetch and execution of instructions. The fetch of the instruction takes one TCY, while the execution takes another TCY. However, due to the overlap of the fetch of current instruction and execution of previous instruction, an instruction is fetched and another instruction is executed every TCY.

Single Cycle Instructions:

With the program memory bus being 16-bits wide, the entire instruction is fetched in a single machine cycle (TCY), except for double word instructions. The instruction contains all the information required and is executed in a single cycle. There may be a one cycle delay in execution if the result of the instruction modified the contents of the program counter. This requires the pipeline to be flushed and a new instruction to be fetched.

Two Cycle Instructions:

Double word instructions require two cycles to execute, since all the required information is in the 32 bits.

Reduced Instruction Set:

When an instruction set is well designed and highly orthogonal (symmetric), fewer instructions are required to perform all needed tasks. With fewer instructions, the whole set can be more rapidly learned.

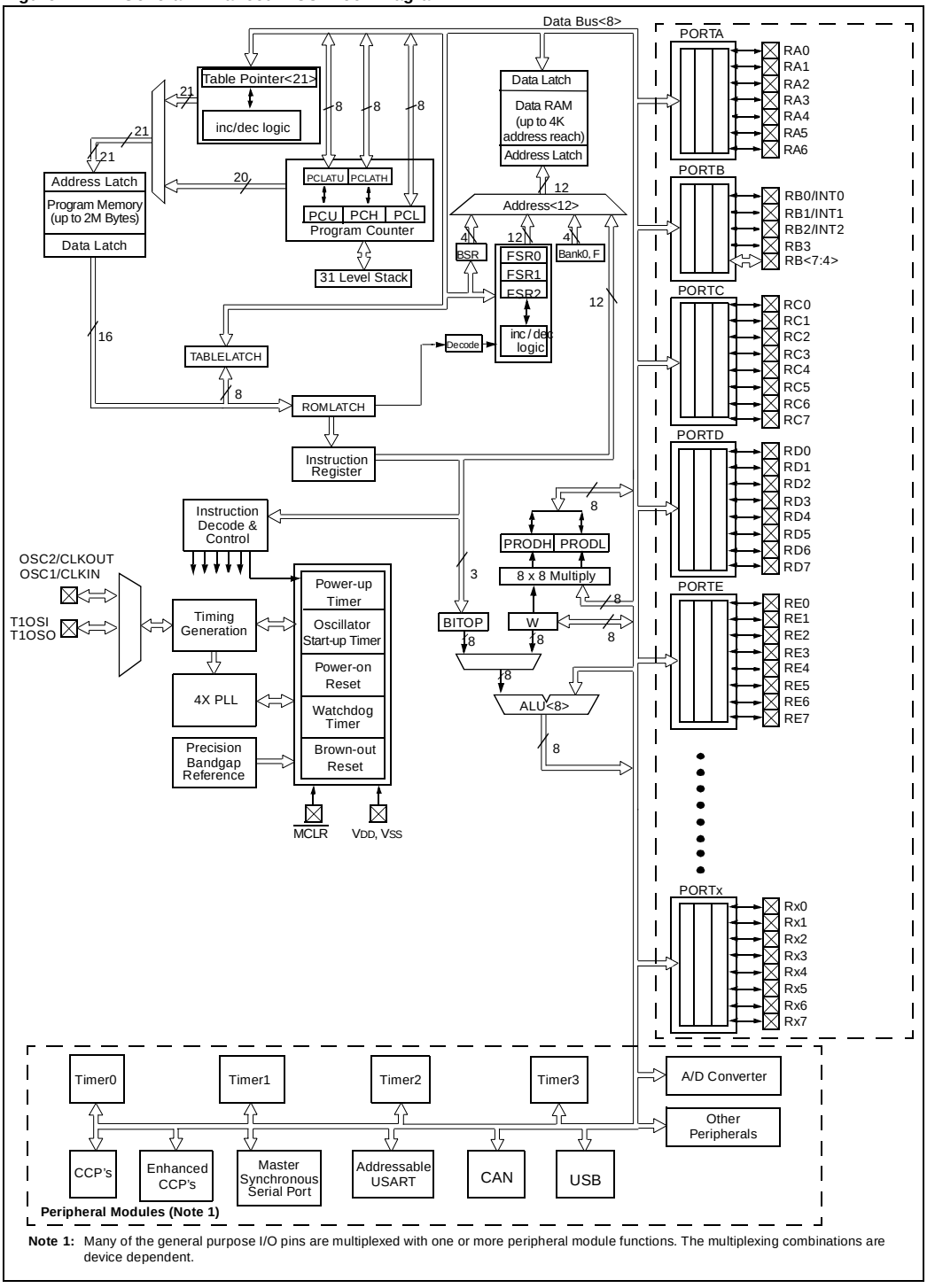
Register File Architecture:

The register files/data memory can be directly or indirectly addressed. All special function registers, including the program counter, are mapped in the data memory.

Orthogonal (Symmetric) Instructions:

Orthogonal instructions make it possible to carry out any operation on any register using any addressing mode. This symmetrical nature and lack of “special instructions” make programming simple yet efficient. In addition, the learning curve is reduced significantly. The Enhanced MCU instruction set uses only three non-register oriented instructions, which are used for two of the cores features. One is the `SLEEP` instruction, which places the device into the lowest power use mode. The second is the `CLRWDT` instruction, which verifies the chip is operating properly by preventing the on-chip Watchdog Timer (WDT) from overflowing and resetting the device. The third is the `RESET` instruction, which resets the device.

Figure 4-2: General Enhanced MCU Block Diagram

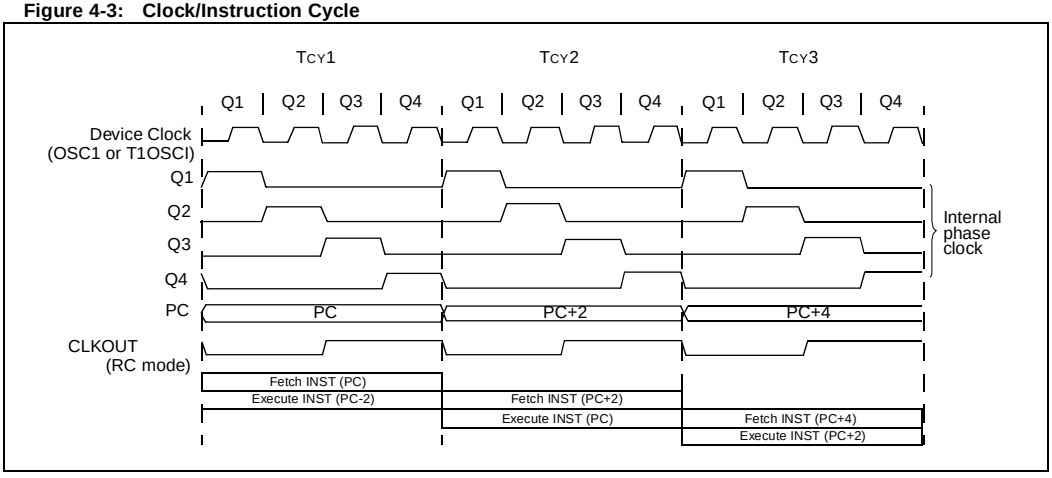


Note 1: Many of the general purpose I/O pins are multiplexed with one or more peripheral module functions. The multiplexing combinations are device dependent.

4.2 Clocking Scheme/Instruction Cycle

The clock input is internally divided by four to generate four non-overlapping quadrature clocks, namely Q1, Q2, Q3 and Q4. Internally, the program counter is incremented every Q1, and the instruction is fetched from the program memory and latched into the instruction register in Q4. The instruction is decoded and executed during the following Q1 through Q4. The clocks and instruction execution flow are illustrated in Figure 4-3 and Example 4-1.

Figure 4-3: Clock/Instruction Cycle

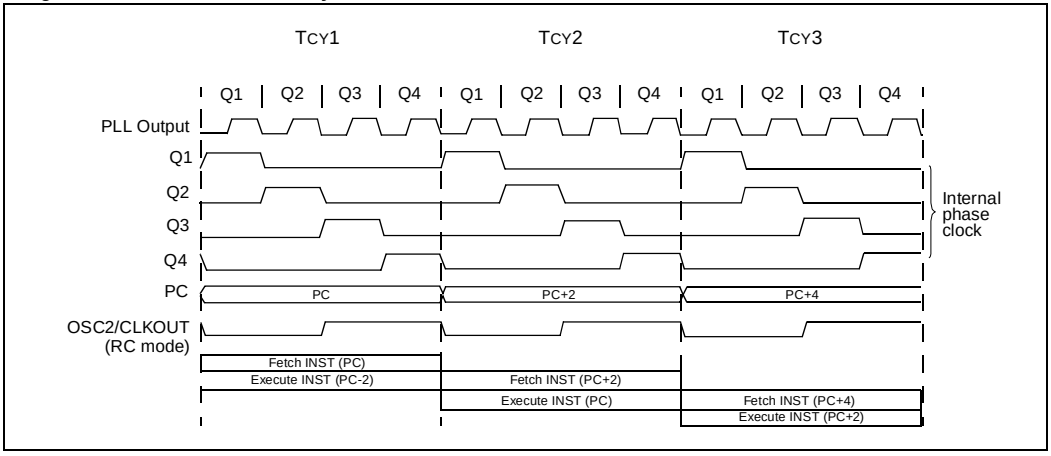


4.2.1 Phase Lock Loop (PLL)

The clock input is multiplied by four by the PLL. Therefore, when it is internally divided by four, it provides an instruction cycle that is the same frequency as the external clock frequency. Four non-overlapping quadrature clocks, namely Q1, Q2, Q3 and Q4 are still generated internally.

Internally, the program counter (PC) is incremented every Q1, and the instruction is fetched from the program memory and latched into the instruction register in Q4. The instruction is decoded and executed during the following Q1 through Q4. The clocks and instruction execution flow are illustrated in Figure 4-4 and Example 4-1.

Figure 4-4: Clock/Instruction Cycle with PLL



4.3 Instruction Flow/Pipelining

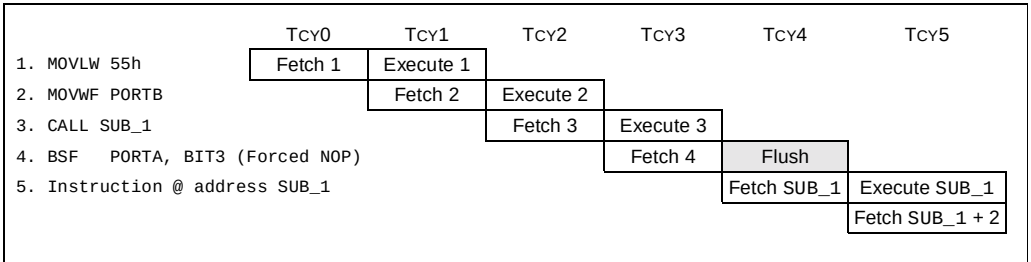
An "Instruction Cycle" consists of four Q cycles (Q1, Q2, Q3 and Q4). Fetch takes one instruction cycle, while decode and execute takes another instruction cycle. However, due to pipelining, each instruction effectively executes in one cycle. If an instruction causes the program counter to change (e.g. GOTO instruction), then an extra cycle is required to complete the instruction (See Example 4-1).

The instruction **fetch** begins with the program counter incrementing in Q1.

In the **execution** cycle, the fetched instruction is latched into the "Instruction Register (IR)" in cycle Q1. This instruction is then decoded and executed during the Q2, Q3 and Q4 cycles. Data memory is read during Q2 (operand read) and written during Q4 (destination write).

Example 4-1 shows the operation of the two stage pipeline for the instruction sequence shown. At time TCY0, the first instruction is fetched from program memory. During TCY1, the first instruction executes, while the second instruction is fetched. During TCY2, the second instruction executes, while the third instruction is fetched. During TCY3, the fourth instruction is fetched, while the third instruction (CALL SUB_1) is executed. When the third instruction completes execution, the CPU forces the address of instruction four onto the Stack and then changes the Program Counter (PC) to the address of SUB_1. This means that the instruction that was fetched during TCY3 needs to be "flushed" from the pipeline. During TCY4, instruction four is flushed (executed as a NOP) and the instruction at address SUB_1 is fetched. Finally during TCY5, instruction five is executed and the instruction at address SUB_1 + 2 is fetched.

Example 4-1: Instruction Pipeline Flow



Most instructions are single cycle. Program branches take two cycles, since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

4.4 I/O Descriptions

Table 4-1 gives a brief description of device pins and the functions that may be multiplexed to a port pin. Multiple functions may exist on one port pin. When multiplexing occurs, the peripheral module's functional requirements may force an override of the data direction (TRIS bit) of the port pin (such as in the A/D and Comparator modules).

Table 4-1: I/O Descriptions

Pin Name	Pin Type	Buffer Type	Description
A19	O	—	System bus address line 19
A18	O	—	System bus address line 18
A17	O	—	System bus address line 17
A16	O	—	System bus address line 16
AD15	I/O	TTL	System bus address/data line 15
AD14	I/O	TTL	System bus address/data line 14
AD13	I/O	TTL	System bus address/data line 13
AD12	I/O	TTL	System bus address/data line 12
AD11	I/O	TTL	System bus address/data line 11
AD10	I/O	TTL	System bus address/data line 10
AD9	I/O	TTL	System bus address/data line 9
AD8	I/O	TTL	System bus address/data line 8
AD7	I/O	TTL	System bus address/data line 7
AD6	I/O	TTL	System bus address/data line 6
AD5	I/O	TTL	System bus address/data line 5
AD4	I/O	TTL	System bus address/data line 4
AD3	I/O	TTL	System bus address/data line 3
AD2	I/O	TTL	System bus address/data line 2
AD1	I/O	TTL	System bus address/data line 1
AD0	I/O	TTL	System bus address/data line 0
ALE	O	—	System bus address latch enable strobe
			Analog Input Channels
AN0	I	Analog	
AN1	I	Analog	
AN2	I	Analog	
AN3	I	Analog	
AN4	I	Analog	
AN5	I	Analog	
AN6	I	Analog	
AN7	I	Analog	
AN8	I	Analog	
AN9	I	Analog	
AN10	I	Analog	
AN11	I	Analog	
AN12	I	Analog	
AN13	I	Analog	
AN14	I	Analog	
AN15	I	Analog	
AVDD	P	P	Analog Power

Legend: TTL = TTL-compatible input
CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
O = output
PU = Weak internal pull-up
I = input
Analog = Analog input or output
P = Power

Table 4-1: I/O Descriptions (Continued)

Pin Name	Pin Type	Buffer Type	Description
AVSS	P	P	Analog Ground
BA0	O	—	System bus byte address 0
CANRX	I	ST	CAN bus receive pin
CANTX0	O	—	CAN bus transmit
CANTX1	O	—	CAN bus complimentary transmit or CAN bus bit time clock
CCP1	I/O	ST	Capture1 input/Compare1 output/PWM1 output
CCP2	I/O	ST	Capture2 input/Compare2 output/PWM2 output.
CK	I/O	ST	USART Synchronous Clock, always associated with TX pin function (See related TX, RX, DT)
CLKI	I	ST/CMOS	External clock source input. Always associated with pin function OSC1. (See related OSC1/CLKIN, OSC2/CLKOUT pins)
CLKO	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate. Always associated with OSC2 pin function. (See related OSC2, OSC1)
CMPA	O	—	Comparator A output
CMPB	O	—	Comparator B output
$\overline{CS}$	I	TTL	Chip select control for parallel slave port (See related $\overline{RD}$ and $\overline{WR}$)
CVREF	O	Analog	Comparator voltage reference output
DT	I/O	ST	USART Synchronous Data. Always associated RX pin function. (See related RX, TX, CK)

Legend: TTL = TTL-compatible input
ST = Schmitt Trigger input with CMOS levels
PU = Weak internal pull-up
Analog = Analog input or output
CMOS = CMOS compatible input or output
O = output
I = input
P = Power

Table 4-1: I/O Descriptions (Continued)

Pin Name	Pin Type	Buffer Type	Description
INT0	I	ST	External Interrupt0
INT1	I	ST	External Interrupt1
INT2	I	ST	External Interrupt2
$\overline{LB}$	O	—	System bus low byte strobe
LVDIN	I	Analog	Low voltage detect input
$\overline{MCLR}$	I/P	ST	Master clear (reset) input or programming voltage input. This pin is an active low reset to the device.
NC	—	—	These pins should be left unconnected.
$\overline{OE}$	O	—	System bus output enable strobe
OSC1	I	ST/CMOS	Oscillator crystal input or external clock source input. ST buffer when configured in RC mode. CMOS otherwise.
OSC2	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT, which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
PSP0	I/O	TTL	Parallel Slave Port for interfacing to a microprocessor port. These pins have TTL input buffers when PSP module is enabled.
PSP1	I/O	TTL	
PSP2	I/O	TTL	
PSP3	I/O	TTL	
PSP4	I/O	TTL	
PSP5	I/O	TTL	
PSP6	I/O	TTL	
PSP7	I/O	TTL	
			PORTA is a bi-directional I/O port.
RA0	I/O	TTL	
RA1	I/O	TTL	
RA2	I/O	TTL	
RA3	I/O	TTL	
RA4	I/O	ST	RA4 is an open drain when configured as output.
RA5	I/O	TTL	
RA6	I/O	TTL	
			PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs.
RB0	I/O	TTL	
RB1	I/O	TTL	
RB2	I/O	TTL	
RB3	I/O	TTL	
RB4	I/O	TTL	Interrupt on change pin.
RB5	I/O	TTL	Interrupt on change pin.
RB6	I/O	TTL/ST	Interrupt on change pin. Serial programming clock. TTL input buffer as general purpose I/O, Schmitt Trigger input buffer when used as the serial programming clock.
RB7	I/O	TTL/ST	Interrupt on change pin. Serial programming data. TTL input buffer as general purpose I/O, Schmitt Trigger input buffer when used as the serial programming data.

Legend: TTL = TTL-compatible input
ST = Schmitt Trigger input with CMOS levels
PU = Weak internal pull-up
Analog = Analog input or output
CMOS = CMOS compatible input or output
O = output
I = input
P = Power

Table 4-1: I/O Descriptions (Continued)

Pin Name	Pin Type	Buffer Type	Description
			PORTC is a bi-directional I/O port.
RC0	I/O	ST	
RC1	I/O	ST	
RC2	I/O	ST	
RC3	I/O	ST	
RC4	I/O	ST	
RC5	I/O	ST	
RC6	I/O	ST	
RC7	I/O	ST	
$\overline{RD}$	I	TTL	Read control for parallel slave port. (See also $\overline{WR}$ and $\overline{CS}$ pins.)
			PORTD is a bi-directional I/O port.
RD0	I/O	ST	
RD1	I/O	ST	
RD2	I/O	ST	
RD3	I/O	ST	
RD4	I/O	ST	
RD5	I/O	ST	
RD6	I/O	ST	
RD7	I/O	ST	
			PORTE is a bi-directional I/O port.
RE0	I/O	ST	
RE1	I/O	ST	
RE2	I/O	ST	
RE3	I/O	ST	
RE4	I/O	ST	
RE5	I/O	ST	
RE6	I/O	ST	
RE7	I/O	ST	
			PORTF is a digital input
RF0	I/O	ST	
RF1	I/O	ST	
RF2	I/O	ST	
RF3	I/O	ST	
RF4	I/O	ST	
RF5	I/O	ST	
RF6	I/O	ST	
RF7	I/O	ST	

Legend: TTL = TTL-compatible input
ST = Schmitt Trigger input with CMOS levels
PU = Weak internal pull-up
Analog = Analog input or output
CMOS = CMOS compatible input or output
O = output
I = input
P = Power

Table 4-1: I/O Descriptions (Continued)

Pin Name	Pin Type	Buffer Type	Description
			PORTG is a digital input
RG0	I/O	ST	
RG1	I/O	ST	
RG2	I/O	ST	
RG3	I/O	ST	
RG4	I/O	ST	
RG5	I/O	ST	
RG6	I/O	ST	
RG7	I/O	ST	
			PORTH is a digital input
RH0	I/O	ST	
RH1	I/O	ST	
RH2	I/O	ST	
RH3	I/O	ST	
RH4	I/O	ST	
RH5	I/O	ST	
RH6	I/O	ST	
RH7	I/O	ST	
			PORTJ is a digital input
RJ0	I/O	ST	
RJ1	I/O	ST	
RJ2	I/O	ST	
RJ3	I/O	ST	
RJ4	I/O	ST	
RJ5	I/O	ST	
RJ6	I/O	ST	
RJ7	I/O	ST	
			PORTK is a digital input
RK0	I/O	ST	
RK1	I/O	ST	
RK2	I/O	ST	
RK3	I/O	ST	
RK4	I/O	ST	
RK5	I/O	ST	
RK6	I/O	ST	
RK7	I/O	ST	

Legend: TTL = TTL-compatible input
ST = Schmitt Trigger input with CMOS levels
PU = Weak internal pull-up
Analog = Analog input or output
CMOS = CMOS compatible input or output
O = output
I = input
P = Power

Table 4-1: I/O Descriptions (Continued)

Pin Name	Pin Type	Buffer Type	Description
			PORTL is a digital input
RL0	I/O	ST	
RL1	I/O	ST	
RL2	I/O	ST	
RL3	I/O	ST	
RL4	I/O	ST	
RL5	I/O	ST	
RL6	I/O	ST	
RL7	I/O	ST	
RX	I	ST	USART Asynchronous Receive
SCL	I/O	ST	Synchronous serial clock input/output for I^2C mode.
SCLA	I/O	ST	Synchronous serial clock for I^2C interface.
SCLB	I/O	ST	Synchronous serial clock for I^2C interface.
SDA	I/O	ST	$I^2C^{™}$ Data I/O
SDAA	I/O	ST	Synchronous serial data I/O for I^2C interface
SDAB	I/O	ST	Synchronous serial data I/O for I^2C interface
SCK	I/O	ST	Synchronous serial clock input/output for SPI mode.
SDI	I	ST	SPI Data In
SDO	O	—	SPI Data Out (SPI mode)
$\overline{SS}$	I	ST	SPI Slave Select input
T0CKI	I	ST	Timer0 external clock input
T1CKI	I	ST	Timer1 external clock input
T1OSO	O	CMOS	Timer1 oscillator output
T1OSI	I	CMOS	Timer1 oscillator input
TX	O	—	USART Asynchronous Transmit (See related RX)
$\overline{UB}$	O	—	System bus upper byte strobe

Legend: TTL = TTL-compatible input
CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
O = output
PU = Weak internal pull-up
I = input
Analog = Analog input or output
P = Power

Table 4-1: I/O Descriptions (Continued)

Pin Name	Pin Type	Buffer Type	Description
VREF	I	Analog	Analog High Voltage Reference input. DR reference voltage output on devices with comparators.
VREF+	I	Analog	Analog High Voltage Reference input. Usually multiplexed onto an analog pin.
VREF-	I	Analog	Analog Low Voltage Reference input. Usually multiplexed onto an analog pin.
VSS	P	—	Ground reference for logic and I/O pins.
VDD	P	—	Positive supply for logic and I/O pins.
VPP	P	—	Programming voltage input
$\overline{WR}$	I	TTL	Write control for parallel slave port (See $\overline{CS}$ and $\overline{RD}$ pins also).
$\overline{WRL}$	O	—	System bus write low byte strobe
$\overline{WRH}$	O	—	System bus write high byte strobe

Legend: TTL = TTL-compatible input
ST = Schmitt Trigger input with CMOS levels
PU = Weak internal pull-up
Analog = Analog input or output
CMOS = CMOS compatible input or output
O = output
I = input
P = Power

4.5 Design Tips

No related design tips at this time.

4.6 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is, they may be written for the Base-Line, the Mid-Range, or High-End families), but the concepts are pertinent and could be used (with modification and possible limitations). The current application notes related to Architecture are:

Title	Application Note #

No related application notes at this time.

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

4.7 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Architecture description.

Section 5. CPU and ALU

HIGHLIGHTS

This section of the manual contains the following major topics:

5.1 Introduction

The Central Processing Unit (CPU) is responsible for using the information in the program memory (instructions) to control the operation of the device. Many of these instructions operate on data memory. To operate on data memory, the Arithmetic Logical Unit (ALU) is required. In addition to performing arithmetical and logical operations, the ALU controls the state of the status bits, which are found in the STATUS register. The result of some instructions force status bits to a value depending on the state of the result.

The machine codes that the CPU recognizes are shown in Table 5-1, as well as the instruction mnemonics that the MPASM uses to generate these codes.

Table 5-1: PIC18CXXX Instruction Set

Mnemonic, Operands		Description	Cycles (4)	16-Bit Instruction Word MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d, a	Add WREG and f	1	0010 01da ffff ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010 00da ffff ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001 101a ffff ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110 11da ffff ffff	Z, N	2
COMF	f, d, a	Complement f	1	0001 11da ffff ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110 001a ffff ffff	None	4
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110 010a ffff ffff	None	4
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110 000a ffff ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000 01da ffff ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010 11da ffff ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100 11da ffff ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010 10da ffff ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011 11da ffff ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100 10da ffff ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001 00da ffff ffff	Z	1, 2
MOVF	f, d, a	Move f	1	0101 00da ffff ffff	Z	
MOVFF	f_s, f_d	Move f_s (source) to f_d (destination)	2	1100 ffff ffff ffff 1111 ffff ffff ffff	None	
MOVWF	f, a	Move WREG to f	1	0110 111a ffff ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000 001a ffff ffff	None	
NEGF	f, a	Negate f	1	0110 110a ffff ffff	C, DC, Z, OV, N	1, 2
RLCF	f, d, a	Rotate Left f through Carry	1	0011 01da ffff ffff	C, DC, Z, N	
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100 01da ffff ffff	C, DC, Z, N	1, 2
RRCF	f, d, a	Rotate Right f through Carry	1	0011 00da ffff ffff	C, DC, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100 00da ffff ffff	C, DC, Z, N	
SETF	f, a	Set f	1	0110 100a ffff ffff	None	
SUBFWB	f, d, a	Subtract f from WREG with	1	0101 01da ffff ffff	C, DC, Z, OV, N	1, 2
SUBWF	f, d, a	Subtract WREG from f	1	0101 11da ffff ffff	C, DC, Z, OV, N	
SUBWFB	f, d, a	Subtract WREG from f with	1	0101 10da ffff ffff	C, DC, Z, OV, N	1, 2
SWAPF	f, d, a	Swap nibbles in f	1	0011 10da ffff ffff	None	4
TSTFSZ	f, a	Test f, skip if 0	1 (2 or 3)	0110 011a ffff ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001 10da ffff ffff	Z, N	
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b, a	Bit Clear f	1	1001 bbba ffff ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000 bbba ffff ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011 bbba ffff ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010 bbba ffff ffff	None	3, 4
BTG	f, d, a	Bit Toggle f	1	0111 bbba ffff ffff	None	1, 2

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

4: Some instructions are 2 word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word retrieves the information embedded in these 16 bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

Table 5-1: PIC18CXXX Instruction Set (Continued)

Mnemonic, Operands		Description	Cycles (4)	16-Bit Instruction Word MSb LSb	Status Affected	Notes
CONTROL OPERATIONS						
BC	n	Branch if Carry	1 (2)	1110 0010 nnnn nnnn	None	
BN	n	Branch if Negative	1 (2)	1110 0110 nnnn nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110 0011 nnnn nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110 0111 nnnn nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110 0101 nnnn nnnn	None	
BNZ	n	Branch if Not Zero	2	1110 0001 nnnn nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110 0100 nnnn nnnn	None	
BRA	n	Branch Unconditionally	1 (2)	1101 0nnn nnnn nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110 0000 nnnn nnnn	None	
CALL	n, s	Call subroutine 1st word	2	1110 110s kkkk kkkk	None	
		2nd word		1111 kkkk kkkk kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000 0000 0000 0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000 0000 0000 0111	C	
GOTO	n	Go to address 1st word	2	1110 1111 kkkk kkkk	None	
		2nd word		1111 kkkk kkkk kkkk		
NOP	—	No Operation	1	0000 0000 0000 0000	None	
NOP	—	No Operation (4)	1	1111 xxxx xxxx xxxx	None	
POP	—	Pop top of return stack (TOS)	1	0000 0000 0000 0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000 0000 0000 0101	None	
RCALL	n	Relative Call	2	1101 1nnn nnnn nnnn	None	
RESET		Software device RESET	1	0000 0000 1111 1111	All	
RETFIE	s	Return from interrupt enable	2	0000 0000 0001 000s	GIEH, GIEL	
RETLW	k	Return with literal in WREG	2	0000 1100 kkkk kkkk	None	
RETURN	s	Return from Subroutine	2	0000 0000 0001 001s	None	
SLEEP	—	Go into standby mode	1	0000 0000 0000 0011	$\overline{TO}$, $\overline{PD}$	
TBLRD	m	Table Read * → mm = 00	2	0000 0000 0000 10mm	None	
		*+ → mm = 01				
		*- → mm = 10				
		+* → mm = 11				5
TBLWT	m	Table Write * → mm = 00	2	0000 0000 0000 11mm	None	
		*+ → mm = 01				
		*- → mm = 10				
		+* → mm = 11				

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

4: Some instructions are 2 word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word retrieves the information embedded in these 16 bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

Table 5-1: PIC18CXXX Instruction Set (Continued)

Mnemonic, Operands		Description	Cycles (4)	16-Bit Instruction Word MSb LSb	Status Affected	Notes
LITERAL OPERATIONS						
ADDLW	k	Add literal and WREG	1	0000 1111 kkkk kkkk	C, DC, Z, OV, N	
ANDLW	k	AND literal with WREG	1	0000 1011 kkkk kkkk	Z, N	
IORLW	k	Inclusive OR literal with WREG	1	0000 1001 kkkk kkkk	Z, N	
MOVLB	k	Move literal to BSR<3:0>	1	0000 0001 0000 kkkk	None	
LFSR	f, k	Move literal (12-bit) to FSRx	2	1110 1110 00ff kkkk	None	
		2nd word		1111 0000 kkkk kkkk		
MOVLW	k	Move literal to WREG	1	0000 1110 kkkk kkkk	None	
MULLW	k	Multiply literal with WREG	1	0000 1101 kkkk kkkk	None	
RETLW	k	Return with literal in WREG	2	0000 1100 kkkk kkkk	None	
SUBLW	k	Subtract WREG from literal	1	0000 1000 kkkk kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR literal with WREG	1	0000 1010 kkkk kkkk	Z, N	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

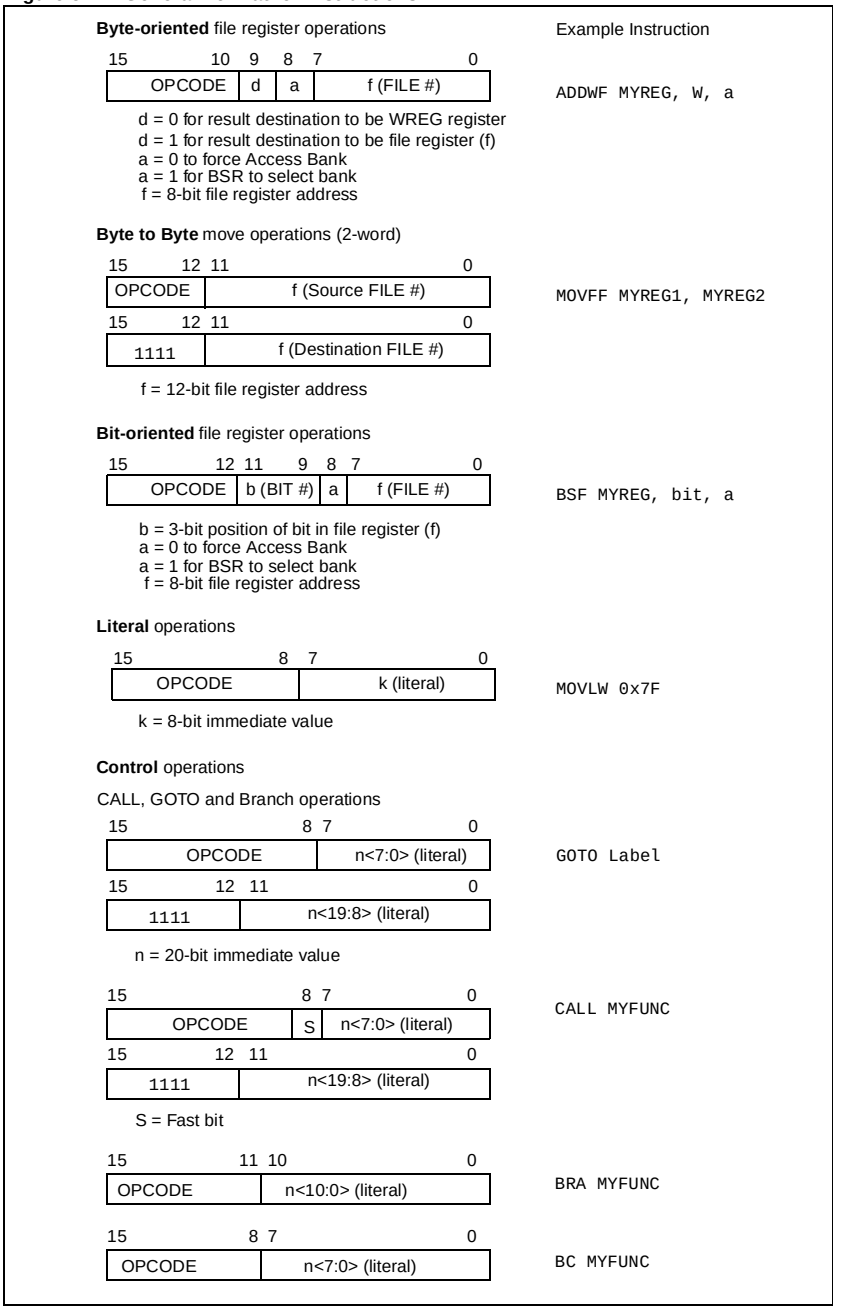
4: Some instructions are 2 word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word retrieves the information embedded in these 16 bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

5.2 General Instruction Format

The Enhanced family instructions can be broken down into five general formats as shown in Figure 5-1. As can be seen, the opcode for the instruction varies from 4 bits to 8 bits. This variable opcode size is what allows 77 instructions to be implemented.

Figure 5-1: General Format for Instructions



5.3 Central Processing Unit (CPU)

The CPU can be thought of as the “brains” of the device. It is responsible for fetching the correct instruction for execution, decoding that instruction and then executing that instruction.

The CPU sometimes works in conjunction with the ALU to complete the execution of the instruction (in arithmetic and logical operations).

The CPU controls the program memory address bus, the data memory address bus and accesses to the stack.

5.4 Instruction Clock

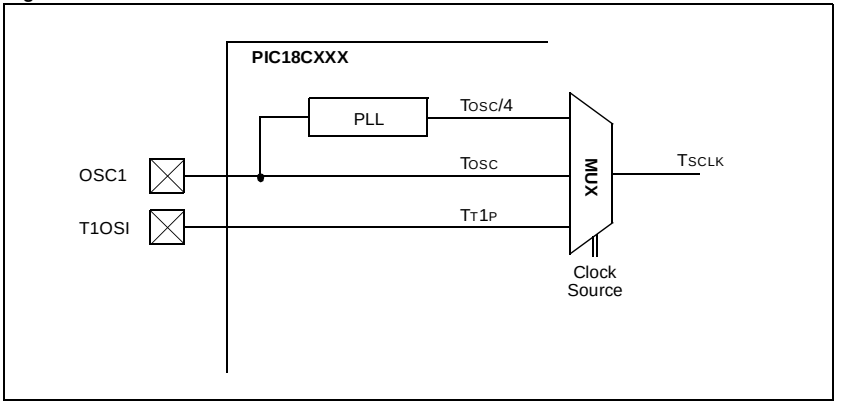
There are three oscillator clock sources from which the device can operate. These are

1. External System Clock (TOSC)
2. Phase Lock Loop (PLL)
3. Timer1 Oscillator (TT1P)

Figure 5-2 shows these clock inputs and the device clock output (TSCLK). TSCLK is the system clock. Four TSCLK cycles are an instruction cycle (TCY).

The external system clock (TOSC) goes into the device and is input into a multiplexer and a 4 x Phase Lock Loop (PLL). The output of the PLL also enters the multiplexer and has a name TOSC/4. Some devices may also have an alternate oscillator called Timer1 oscillator (see **"Timer1"** section), which can provide another system clock. Timer1 has a cycle time called TT1P. This clock source also enters into the multiplexer.

Figure 5-2: Device Clock Sources



Each instruction cycle (TCY) is comprised of four Q cycles (Q1-Q4). The Q cycle time is the same as the system clock cycle time (TSCLK). The Q cycles provide the timing/designation for the Decode, Read, Process Data, Write, etc., of each instruction cycle.

The four Q cycles that make up an instruction cycle (TCY) are shown in Figure 5-3. The relationship of the Q cycles to the instruction cycle can be generalized as:

Q1: Instruction Decode Cycle or forced No Operation (NOP)

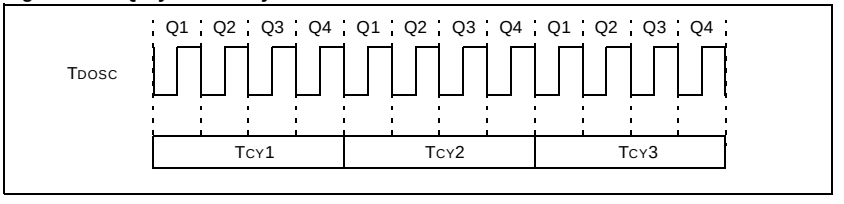
Q2: Instruction Read Data Cycle or No Operation (NOP)

Q3: Process the Data

Q4: Instruction Write Data Cycle or No Operation (NOP)

Each instruction description will show a detailed Q cycle operation for the instruction.

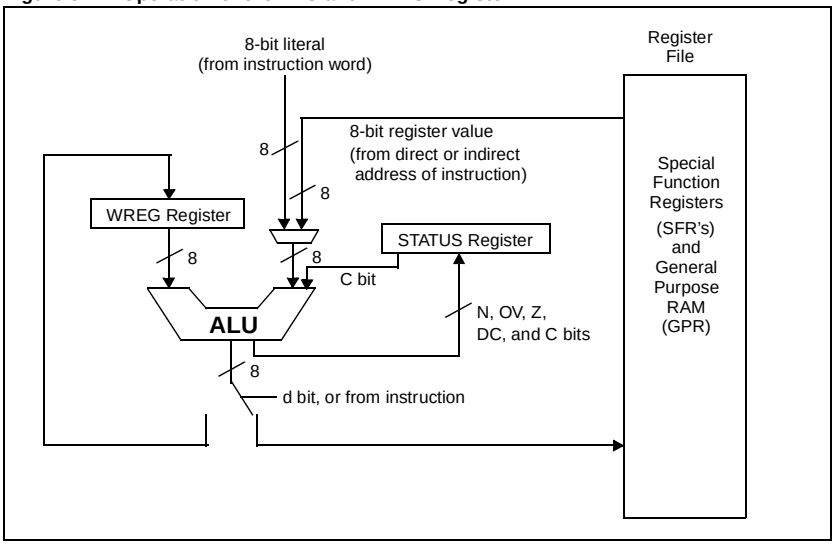
Figure 5-3: Q Cycle Activity



5.5 Arithmetic Logical Unit (ALU)

PICmicro devices contain an 8-bit ALU and an 8-bit working register (WREG). The ALU is a general purpose arithmetic and logical unit. It performs arithmetic and Boolean functions between the data in the working register and any register file. The WREG register is directly addressable and in the SFR memory map.

Figure 5-4: Operation of the ALU and WREG Register



The ALU is 8-bits wide and is capable of addition, subtraction, multiplication, shift and logical operations. Unless otherwise mentioned, arithmetic operations are two's complement in nature. In two-operand instructions, typically one operand is the working register (WREG register). The other operand is a file register or an immediate constant. In single operand instructions, the operand is either the WREG register or a file register.

The 8x8 multiplier operates in a single cycle, placing the 16-bit result in the PRODH:PRODL register pair.

Depending on the instruction executed, the ALU may affect the values of the Carry (C), Digit Carry (DC), Zero (Z), Overflow (OV), and Negative (N) bits in the STATUS register. The C and DC bits operate as a borrow bit and a digitborrow out bit, respectively, in subtraction. See the SUBLW and SUBWF instructions in the **"Instruction Set"** section for examples.

5.5.1 Signed Math

Signed arithmetic is comprised of a magnitude and a sign bit. The overflow bit indicates if the magnitude overflows and causes the sign bit to change state when the result of an 8-bit signed operation is greater than 127 (7Fh) or less than -128 (80h).

Signed math can have greater than 7-bit values (magnitude), if more than one byte is used. The overflow bit only operates on bit6 (MSb of magnitude) and bit7 (sign bit) of each byte value in the ALU. That is, the overflow bit is not useful if trying to implement signed math where the magnitude, for example, is 11 bits.

If the signed math values are greater than 7 bits (such as 15, 24 or 31 bits), the algorithm must ensure that the low order bytes of the signed value ignore the overflow status bit.

Example 5-1 shows two cases of doing signed arithmetic. The Carry (C) bit and the Overflow (OV) bit are the most important status bits for signed math operations.

Example 5-1: 8-bit Math Addition

Case 1:

Hex Value	Signed Values	Unsigned Values
FFh	-1	255
+ 01h	+ 1	+ 1
= 00h	= 0 (FEh)	= 256 → 00h
C bit = 1	C bit = 1	C bit = 1
OV bit = 0	OV bit = 0	OV bit = 0
DC bit = 1	DC bit = 1	DC bit = 1
Z bit = 1	Z bit = 1	Z bit = 1
N bit = 0	N bit = 0	N bit = 0

Case 2:

Hex Value	Signed Values	Unsigned Values
7Fh	127	127
+ 01h	+ 1	+ 1
= 80h	= 128 → 00h	= 128
C bit = 0	C bit = 0	C bit = 0
OV bit = 1	OV bit = 1	OV bit = 1
DC bit = 1	DC bit = 1	DC bit = 1
Z bit = 0	Z bit = 0	Z bit = 0
N bit = 1	N bit = 1	N bit = 1

The Negative bit is used to indicate if the MSb of the result is set or cleared.

5.6 STATUS Register

The STATUS register, shown in Register 5-1, contains the arithmetic status of the ALU. The STATUS register can be the destination for any instruction, as with any other register. If the STATUS register is the destination for an instruction that affects the Z, DC, C, OV or N bits, then the write to these five bits is disabled. These bits are set or cleared according to the device logic. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper three bits and set the Z bit. This leaves the STATUS register as `000u u1uu` (where u = unchanged).

It is recommended, therefore, that only `BCF, BSF, SWAPF, MOVFF,` and `MOVWF` instructions are used to alter the STATUS register, because these instructions do not affect the Z, C, DC, OV or N bits of the STATUS register. For other instructions, not affecting any status bits, see Table 5-1.

Note 1: The C and DC bits operate as a $\overline{\text{borrow}}$ and $\overline{\text{digitborrow}}$ bit, respectively, in subtraction.

Register 5-1: STATUS Register

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC	C
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **N:** Negative bit
This bit is used for signed arithmetic (2's complement). It indicates whether the result was negative, (ALU MSb = 1).
1 = Result was negative
0 = Result was positive

bit 3 **OV:** Overflow bit
This bit is used for signed arithmetic (2's complement). It indicates an overflow of the 7-bit magnitude, which causes the sign bit (bit7) to change state.
1 = Overflow occurred for signed arithmetic (in this arithmetic operation)
0 = No overflow occurred

bit 2 **Z:** Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC:** Digit carry/$\overline{\text{borrow}}$ bit
For ADDWF, ADDLW, SUBLW, and SUBWF instructions
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

Note: For $\overline{\text{borrow}}$, the polarity is reversed. A subtraction is executed by adding the 2's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the bit4 or bit3 of the source register.

bit 0 **C:** Carry/$\overline{\text{borrow}}$ bit
For ADDWF, ADDLW, SUBLW, and SUBWF instructions
1 = A carry-out from the most significant bit of the result occurred
0 = No carry-out from the most significant bit of the result occurred

Note: For $\overline{\text{borrow}}$, the polarity is reversed. A subtraction is executed by adding the 2's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

5.6.1 RCON Register

The Reset Control (RCON) register contains flag bit(s) that allow the user to differentiate between the device resets.

Note 1:	If the BOREN configuration bit is set, $\overline{BOR}$ is '0' on Power-on Reset. If the BOREN configuration bit is clear, $\overline{BOR}$ is unknown on Power-on Reset. The $\overline{BOR}$ status bit is a "don't care" and is not necessarily predictable if the brown-out circuit is disabled (the BOREN configuration bit is clear).
2:	It is recommended that the $\overline{POR}$ bit be set after a Power-on Reset has been detected, so that subsequent Power-on Resets may be detected.

Register 5-2: RCON Register

R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
IPEN	LWRT	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

bit 7 **IPEN**: Interrupt Priority Enable bit
This bit reflects the value of the MPEEN configuration bit.
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16CXXX compatibility mode)

bit 6 **LWRT:** Long Write Enable
1 = Enable Table Writes to internal program memory
Once this bit is set, it can only be cleared by a POR or $\overline{MCLR}$ reset.
0 = Disable Table Writes to internal program memory; Table Writes only to external program memory

bit 5 **Unimplemented:** Read as '0'

bit 4 **$\overline{RI}$:** Reset Instruction Flag bit
1 = The Reset instruction was not executed to cause the device reset
0 = The Reset instruction was executed
(must be set in software after a Brown-out Reset occurs)

bit 3 **$\overline{TO}$:** Watchdog Time-out Flag bit
1 = After Power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 2 **$\overline{PD}$**: Power-down Detection Flag bit
1 = After Power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 1 **$\overline{POR}$**: Power-on Reset Status bit
1 = A Power-on Reset has not occurred
0 = A Power-on Reset occurred
(After a Power-on Reset occurs, this bit must be set in software to detect subsequent occurrences of Power-on Reset.

bit 0 **$\overline{BOR}$**: Brown-out Reset Status bit
1 = A Brown-out Reset has not occurred
0 = A Brown-out Reset occurred
(must be set in software after a Brown-out Reset occurs)

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

5.7 Design Tips

Question 1: ***My program algorithm does not seem to function correctly.***

Answer 1:

There are many possible reasons for this. A couple of possibilities are:

1. The destination of the instruction may be specifying the WREG register (d = 0) instead of the file register (d = 1).
2. The access bit may be specifying the Virtual RAM Bank instead of the desired bank of RAM.

When possible, the use of an In-Circuit Emulator (such as MPLAB-ICE) or a simulator (such as MPLAB-SIM) can assist in locating the reason for the unexpected execution flow.

Question 2: ***I cannot seem to modify the STATUS register flags.***

Answer 2:

If the STATUS register is the destination for an instruction that affects the Z, DC, C, OV or N bits, then the write to those bits is disabled. These bits are set or cleared based on device logic. Therefore, to modify bits in the STATUS register, it is recommended to use the BCF and BSF instructions.

5.8 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is they may be written for the Base-Line, the Mid-Range or High-End families), but the concepts are pertinent and could be used (with modification and possible limitations). The current application notes related to the CPU or the ALU are:

Title	Application Note #
IEEE 754 Compliant Floating Point Routines	AN575
Fixed Point Routines	AN617
Floating Point Math Functions	AN660

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

5.9 Revision History

Revision A

This is the initial released revision of the Enhanced MCU CPU and ALU description.

Section 6. Hardware 8x8 Multiplier

HIGHLIGHTS

This section of the manual contains the following major topics:

6.1 Introduction

An 8 x 8 hardware multiplier is included in the ALU of the devices. By making the multiplication a hardware operation, it completes in a single instruction cycle. This is an unsigned multiplication that gives a 16-bit result. The result is stored into the 16-bit Product register (PRODH:PRODL). The multiplier does not affect any flags in the ALUSTA register.

Making the 8 x 8 multiplier execute in a single cycle gives the following advantages:

- Higher computational throughput
- Reduces code size requirements for multiplication algorithms

The performance increase allows the device to be used in applications previously reserved for Digital Signal Processors.

Table 6-1 shows a performance comparison between devices using the single cycle hardware multiplier and performing the same function without the hardware multiplier.

Table 6-1: Performance Comparison

Routine	Multiply Method	Program Memory (Words)	Cycles (Max)	Time		
				@ 40 MHz	@ 10 MHz	@ 4 MHz
8 x 8 unsigned	Without hardware multiplier	13	69	6.9 µs	27.6 µs	69 µs
	Hardware multiply	1	1	100 ns	400 ns	1 µs
8 x 8 signed	Without hardware multiplier	33	91	9.1 µs	36.4 µs	91 µs
	Hardware multiply	6	6	600 ns	2.4 µs	6 µs
16 x 16 unsigned	Without hardware multiplier	21	242	24.2 µs	96.8 µs	242 µs
	Hardware multiply	24	24	2.4 µs	9.6 µs	24 µs
16 x 16 signed	Without hardware multiplier	52	254	25.4 µs	102.6 µs	254 µs
	Hardware multiply	36	36	3.6 µs	14.4 µs	36 µs

6.2 Operation

Example 6-1 shows the sequence to do an 8 x 8 unsigned multiply. Only one instruction is required when one argument of the multiply is already loaded in the WREG register.

Example 6-2 shows the sequence to do an 8 x 8 signed multiply. To account for the sign bits of the arguments, each argument's most significant bit (MSb) is tested and the appropriate subtractions are done.

Example 6-1: 8 x 8 Unsigned Multiply Routine

```
    MOVFF    ARG1, WREG  ;
    MULWF    ARG2        ; ARG1 * ARG2 ->
                         ;   PRODH:PRODL
```

Example 6-2: 8 x 8 Signed Multiply Routine

```
    MOVFF    ARG1, WREG
    MULWF    ARG2        ; ARG1 * ARG2 ->
                         ;   PRODH:PRODL
    BTFSC    ARG2, SB    ; Test Sign Bit
    SUBWF    PRODH, F    ; PRODH = PRODH
                         ;         - ARG1
    MOVFF    ARG2, WREG
    BTFSC    ARG1, SB    ; Test Sign Bit
    SUBWF    PRODH, F    ; PRODH = PRODH
                         ;         - ARG2
```

Example 6-3 shows the sequence to do a 16 x 16 unsigned multiply. Equation 6-1 shows the algorithm that is used. The 32-bit result is stored in four registers, RES3, RES2, RES1 and RES0.

Equation 6-1: 16 x 16 Unsigned Multiplication Algorithm

$$
\begin{aligned}
RES3:RES2:RES1:RES0 &= ARG1H:ARG1L \bullet ARG2H:ARG2L \\
&= (ARG1H \bullet ARG2H \bullet 2^{16})+ \\
&\quad (ARG1H \bullet ARG2L \bullet 2^{8})+ \\
&\quad (ARG1L \bullet ARG2H \bullet 2^{8})+ \\
&\quad (ARG1L \bullet ARG2L)
\end{aligned}
$$

Example 6-3: 16 x 16 Unsigned Multiply Routine

```
   MOVFF    ARG1L, WREG
   MULWF    ARG2L       ; ARG1L * ARG2L ->
                        ;   PRODH:PRODL
   MOVFF    PRODH, RES1 ;
   MOVFF    PRODL, RES0 ;
;
   MOVFF    ARG1H, WREG
   MULWF    ARG2H       ; ARG1H * ARG2H ->
                        ;   PRODH:PRODL
   MOVFF    PRODH, RES3 ;
   MOVFF    PRODL, RES2 ;
;
   MOVFF    ARG1L, WREG
   MULWF    ARG2H       ; ARG1L * ARG2H ->
                        ;   PRODH:PRODL
   MOVFF    PRODL, WREG ;
   ADDWF    RES1, F     ; Add cross
   MOVFF    PRODH, WREG ;    products
   ADDWFC   RES2, F     ;
   CLRF     WREG, F     ;
   ADDWFC   RES3, F     ;
;
   MOVFF    ARG1H, WREG ;
   MULWF    ARG2L       ; ARG1H * ARG2L ->
                        ;   PRODH:PRODL
   MOVFF    PRODL, WREG ;
   ADDWF    RES1, F     ; Add cross
   MOVFF    PRODH, WREG ;    products
   ADDWFC   RES2, F     ;
   CLRF     WREG, F     ;
   ADDWFC   RES3, F     ;
```

Example 6-4 shows the sequence to do a 16 x 16 signed multiply. Equation 6-2 shows the algorithm used. The 32-bit result is stored in four registers, RES3, RES2, RES1 and RES0. To account for the sign bits of the arguments, each argument pairs' most significant bit (MSb) is tested and the appropriate subtractions are done.

Equation 6-2: 16 x 16 Signed Multiplication Algorithm

$$
\begin{aligned}
&RES3:RES2:RES1:RES0 \\
&\quad = ARG1H:ARG1L \bullet ARG2H:ARG2L \\
&\quad = (ARG1H \bullet ARG2H \bullet 2^{16}) + \\
&\qquad (ARG1H \bullet ARG2L \bullet 2^{8}) + \\
&\qquad (ARG1L \bullet ARG2H \bullet 2^{8}) + \\
&\qquad (ARG1L \bullet ARG2L) + \\
&\qquad (-1 \bullet ARG2H\langle 7\rangle \bullet ARG1H:ARG1L \bullet 2^{16}) + \\
&\qquad (-1 \bullet ARG1H\langle 7\rangle \bullet ARG2H:ARG2L \bullet 2^{16})
\end{aligned}
$$

Example 6-4: 16 x 16 Signed Multiply Routine

```
   MOVFF    ARG1L, WREG
   MULWF    ARG2L       ; ARG1L * ARG2L ->
                        ;   PRODH:PRODL
   MOVFF    PRODH, RES1 ;
   MOVFF    PRODL, RES0 ;
;
   MOVFF    ARG1H, WREG
   MULWF    ARG2H       ; ARG1H * ARG2H ->
                        ;   PRODH:PRODL
   MOVFF    PRODH, RES3 ;
   MOVFF    PRODL, RES2 ;
;
   MOVFF    ARG1L, WREG
   MULWF    ARG2H       ; ARG1L * ARG2H ->
                        ;   PRODH:PRODL
   MOVFF    PRODL, WREG ;
   ADDWF    RES1, F     ; Add cross
   MOVFF    PRODH, WREG ;    products
   ADDWFC   RES2, F     ;
   CLRF     WREG, F     ;
   ADDWFC   RES3, F     ;
;
   MOVFF    ARG1H, WREG ;
   MULWF    ARG2L       ; ARG1H * ARG2L ->
                        ;   PRODH:PRODL
   MOVFF    PRODL, WREG ;
   ADDWF    RES1, F     ; Add cross
   MOVFF    PRODH, WREG ;    products
   ADDWFC   RES2, F     ;
   CLRF     WREG, F     ;
   ADDWFC   RES3, F     ;
;
   BTFSS    ARG2H, 7    ; ARG2H:ARG2L neg?
   GOTO     SIGN_ARG1   ; no, check ARG1
   MOVFF    ARG1L, WREG ;
   SUBWF    RES2        ;
   MOVFF    ARG1H, WREG ;
   SUBWFB   RES3
;
SIGN_ARG1
   BTFSS    ARG1H, 7    ; ARG1H:ARG1L neg?
   GOTO     CONT_CODE   ; no, done
   MOVFF    ARG2L, WREG ;
   SUBWF    RES2        ;
   MOVFF    ARG2H, WREG ;
   SUBWFB   RES3
;
CONT_CODE
     :
```

6.3 Design Tips

None.

6.4 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is, they may be written for the Base-Line, the Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the H/W Multiplier modules are:

Title	Application Note #
IEEE 754 Compliant Floating Point Routines	AN575
Fixed Point Routines	AN617
Floating Point Math Functions	AN660

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

6.5 Revision History

Revision A

This is the initial released revision of the Enhanced MCE Hardware Multiplier module description.

Section 7. Memory Organization

HIGHLIGHTS

This section of the manual contains the following major topics:

7.1 Introduction

There are two memory blocks in the memory map; program memory and data memory. Each block has its own bus, so that access to each block can occur during the same instruction cycle.

The data memory can further be broken down into General Purpose RAM and the Special Function Registers (SFRs). The operation of the SFRs that control the "core" are described here. The SFRs used to control the peripheral modules are described in the section discussing each individual peripheral module.

In addition, there are other registers used that are neither part of the program nor data memory spaces. These registers are not directly addressable and include:

- return address stack
- fast return stack

Table 7-1 shows the program memory space used depending on the memory allocated, and Table 7-2 shows the data memory space used.

Table 7-1: PIC18CXXX Program Memory Ranges

Program Memory	Program Memory Address Range
1K x 8	0000h - 3FFh
2K x 8	0000h - 7FFh
4K x 8	0000h - FFFh
8K x 8	0000h - 1FFFh
12K x 8	0000h - 2FFFh
16K x 8	0000h - 3FFFh
24K x 8	0000h - 5FFFh
32K x 8	0000h - 7FFFh
48K x 8	0000h - BFFFh
64K x 8	0000h - FFFFh
96K x 8	0000h - 17FFFh
128K x 8	0000h - 1FFFFh
160K x 8	0000h - 27FFFh
192K x 8	0000h - 2FFFFh
256K x 8	0000h - 3FFFFh
384K x 8	0000h - 5FFFFh
512K x 8	0000h - 7FFFFh
768K x 8	0000h - BFFFFh
1024K x 8	0000h - FFFFFh
1536K x 8	0000h - 17FFFFh
2048K x 8	0000h - 1FFFFFh

Table 7-2: PIC18CXXX Data Memory Ranges

Data Memory	Banks
64	0, 15
128	0, 15
256	0, 15
512	0-1, 15
640	0-2, 15
768	0-2, 15
1024	0-3, 15
1280	0-4, 15
1536	0-5,15
1792	0-6, 15
2048	0-7, 15
2304	0-8, 15
2560	0-9, 15
2816	0-10, 15
3072	0-11, 15
3328	0-12, 15
3584	0-13,15
3840	0-14,15
3968	0-15

7.2 Program Memory

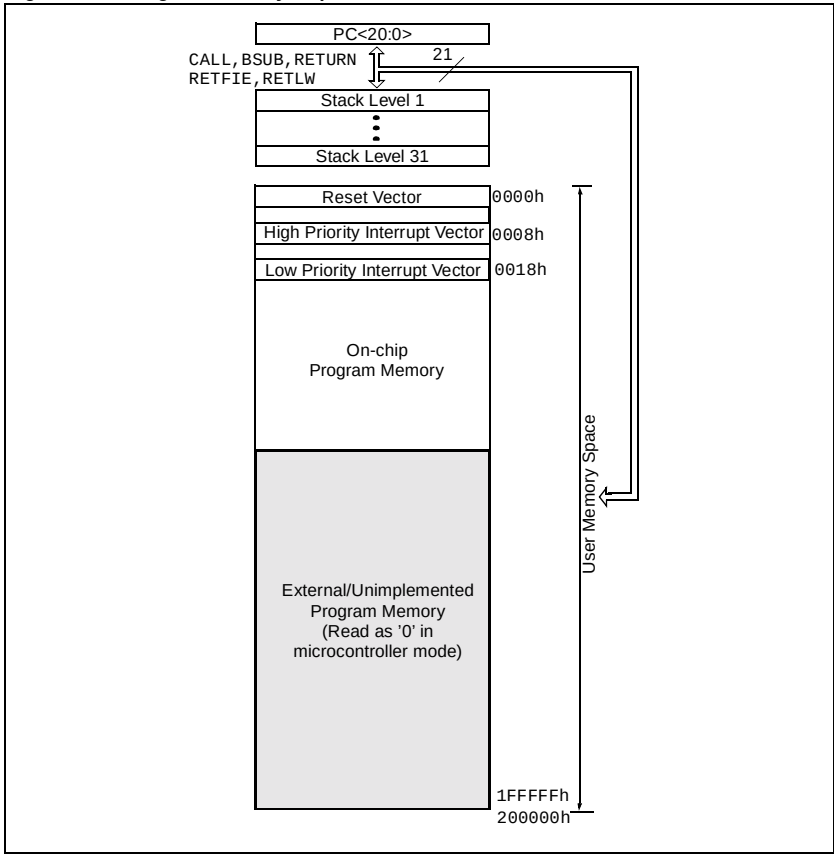
Enhanced MCU devices have a 21-bit program counter capable of addressing 2 Mbytes (1Mwords) of program memory space. The program memory contains instructions for execution and data tables for storing fixed data. Data tables may be written once using table write instructions and read as required, using the table read instructions.

The program space is implemented as a single contiguous block. The reset vector is at address 000000h, the high priority interrupt vector is at address 000008h, and the low priority interrupt vector is at address 000018h (Figure 7-1).

CALL and GOTO instructions can address any location in the memory map, while the BRA and RCALL instructions have a limited program memory reach (+1024, -1023 program memory word locations). To allow the CALL and GOTO instructions to contain the entire address, it requires that these instructions use 2 program memory words (2 word instruction).

Instructions are also available to move information between the data memory and the program memory areas. These are called table operations. Table operations work with byte entities. This is discussed in detail in the **"Table Read/Table Write"** section.

Figure 7-1: Program Memory Map and Stack for PIC18CXXX



7.2.1 Reset Vector

On any Enhanced MCU device, a reset forces the Program Counter (PC) to address 0h. This is known as the “Reset Vector Address”, since this is the address that program execution will branch to when a device reset occurs.

Any reset will also clear the contents of the PCLATU and PCLATH registers.

7.2.2 Interrupt Vectors

Two interrupt vectors are implemented; one for interrupts programmed as high priority and the other for the interrupts programmed as low priority. The vector addresses are 08h for high priority interrupts and 18h for low priority interrupts. If the interrupt priority is not used, all interrupts are treated as high priority.

When an interrupt is acknowledged, the PC is forced to address 0008h or 0018h. This is known as the “Interrupt Vector Address”. When the PC is forced to the interrupt vector, the PCLATU and PCLATH registers are not modified. Once in the service interrupt routine (ISR), before any write to the PC, the PCLATH register should be written with the value that will specify the desired location in program memory. Before the PCLATH register is modified by the Interrupt Service Routine (ISR), the contents of the PCLATH may need to be saved so it can be restored before returning from the ISR.

7.2.3 Calibration Information

Some devices have calibration information stored in their program memory. This information is programmed by Microchip when the device is under final test. The use of these values allows the application to achieve better results. The calibration information is typically at the end of program memory. These bytes can be accessed with the table read instructions.

Note:	For windowed devices, write down all calibration values **BEFORE** erasing. This allows the device’s calibration values to be restored when the device is re-programmed. When possible, writing the values on the package is recommended.

7.2.4 Instructions in Program Memory

The program memory is addressed in bytes. Instructions are stored as two bytes or four bytes in program memory. The least significant byte of an instruction word is always stored in a program memory location with an even address (LSb = '0'). Figure 7-2 shows an example of how instruction words are stored in the program memory. To maintain alignment with instruction boundaries, the PC increments in steps of 2 and the LSb will always read '0'.

The CALL and GOTO instructions have an absolute program memory address embedded into the instruction. Since instructions are always stored on word boundaries, the data contained in the instruction is a word address. The word address is written to PC<20:1>, which accesses the desired byte address in program memory. Instruction #2 in Figure 7-2 shows how the instruction "GOTO 000006h' is encoded in the program memory. Program branch instructions which encode a relative address offset operate in the same manner. The offset value stored in a branch instruction represents the number of single word instructions that the PC will be offset by. The **"Instruction Set"** section provides further details of the instruction set.

Figure 7-2: Instructions in Program Memory

			High Byte	Low Byte	Word Address ↓
	Program Memory Byte Locations →				000000h
					000002h
					000004h
					000006h
Instruction 1:	MOVLW	055h	0Fh	55h	000008h
Instruction 2:	GOTO	000006h	EFh	03h	00000Ah
			F0h	00h	00000Ch
Instruction 3:	MOVFF	123h, 456h	C1h	23h	00000Eh
			F4h	56h	000010h
					000012h
					000014h

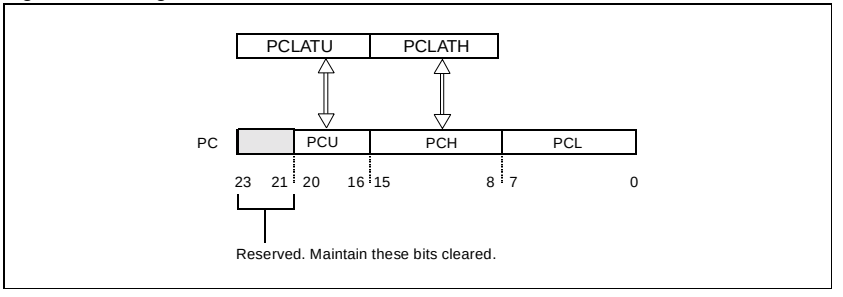
7.3 Program Counter (PC)

The Program Counter (PC) specifies the address of the instruction to fetch for execution. The PC is 21-bits wide and addresses each byte (rather than words) in the program memory. The low byte is called the PCL register (PC<7:0>). This register is readable and writable. The high byte is called the PCH register (PC<15:8>). This register is not directly readable or writable. Updates to the PCH register may be performed through the PCLATH register.

The upper byte is called the PCU register (PC<20:16>). The PCU register is not directly readable or writable. Updates to the PCU register may be performed through the PCLATU register. The PC structure is PCU<4:0>:PCH<7:0>:PCL<7:0> and is equivalent to PC<20:0>.

Figure 7-3 shows the interaction of the PCU, PCH, and PCL registers with the PCLATU and PCLATH registers.

Figure 7-3: Program Counter Structure



The low byte of the PC (PCL<7:0>) is mapped in the data memory. PCL is readable and writable just as is any other register. PCU and PCH are the upper and high bytes of the PC respectively, and are not directly addressable. Registers PCLATU<4:0> (PC upper latch) and PCLATH<7:0> (PC high latch) are used as holding latches for the high bytes of the PCU and PCH, and are mapped into data memory. The user can read and write PCH through PCLATH and PCU through PCLATU. Any time PCL is read, the current contents of PCH and PCU are transferred to PCLATH and PCLATU, respectively. Any time PCL is written to, the contents of PCLATH and PCLATU are transferred to PCH and PCU, respectively.

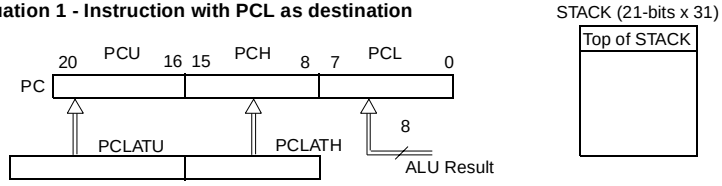
Note:	The values in PCLATU and PCLATH do not always reflect the current value in PCU and PCH. When needing to modify the current Program Counter (PC) value, first read the PCL register to update the values in the PCLATU and PCLATH registers.

The PC addresses bytes rather than words in the program memory. Because the PC must access the instructions in program memory on an even byte boundary, the LSb of the PC is a forced '0' and the PC increments by two for each instruction. The LSb bit of the PCL is readable but not writable. Any write to the LSb is ignored.

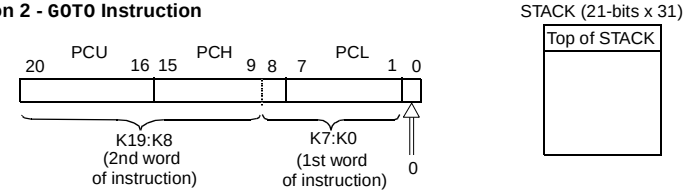
Figure 7-4 shows the four situations for the loading of the PC. Situation 1 shows how the PC is loaded on a write to PCL (PCLATH<4:0> → PCH). Situation 2 shows how the PC is loaded during a GOTO instruction (PCLATH<4:3> → PCH). Situation 4 shows how the PC is loaded during a CALL instruction (PCLATH<4:3> → PCH), with the PC loaded (PUSHed) onto the Top of Stack. Situation 6 shows how the PC is loaded during one of the return instructions where the PC is loaded (POPed) from the Top of Stack.

Figure 7-4: Loading of PC In Different Situations

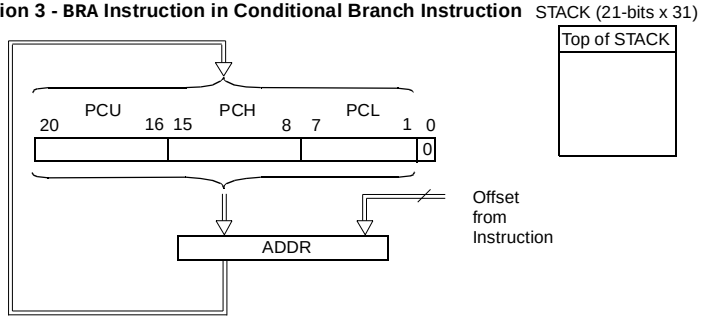
Situation 1 - Instruction with PCL as destination



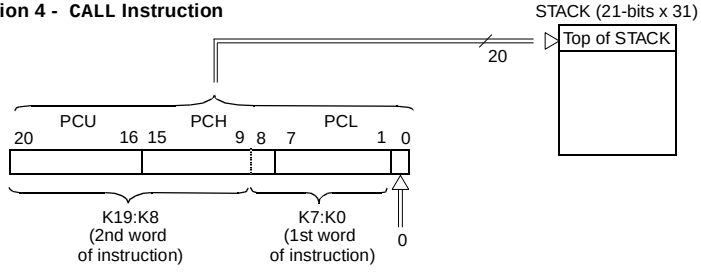
Situation 2 - GOTO Instruction



Situation 3 - BRA Instruction in Conditional Branch Instruction

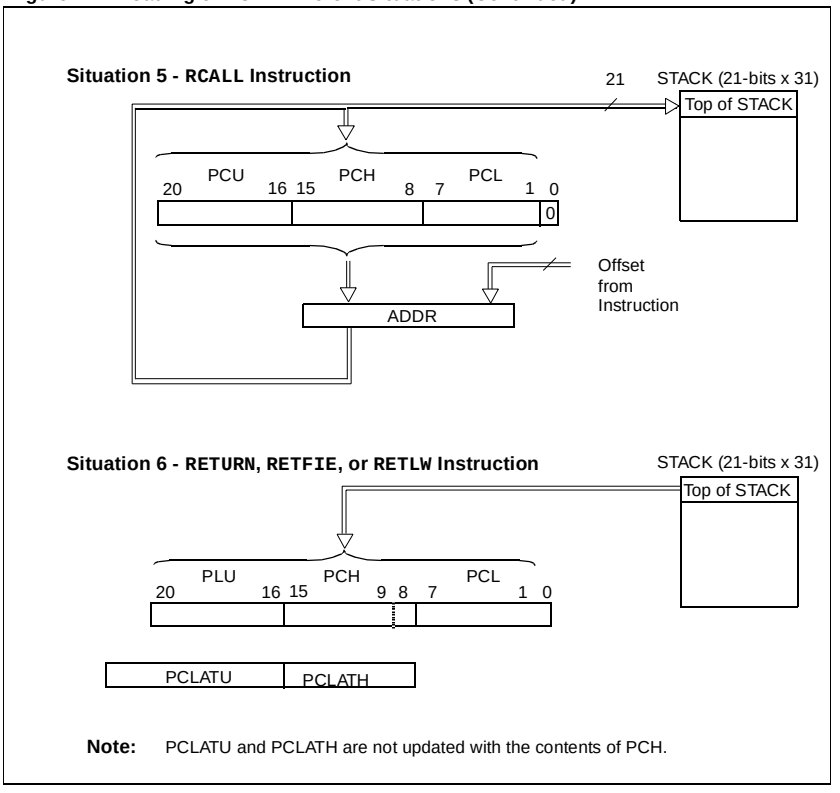


Situation 4 - CALL Instruction



Note: PCLATU and PCLATH are not updated with the contents of PCH.

Figure 7-4: Loading of PC In Different Situations (Continued)



Note: PCLATU and PCLATH are not updated with the contents of PCH.

7.3.1 Computed GOTO

A computed GOTO is accomplished by adding an offset to the program counter (ADDWF PCL). When doing a table read using a computed GOTO method, care should be exercised if the table location crosses a PCL memory boundary (each 256 byte block) and the PCH memory boundary (each 64Kbyte block).

A lookup table can be formed with an ADDWF PCL instruction and a group of RETLW 0xnn instructions. WREG is loaded with an offset into the table before executing a call to that table. The first instruction of the called routine is the ADDWF PCL instruction. The next instruction executed will be one of the RETLW 0xnn instructions that returns the value 0xnn to the calling function.

Since the Program Counter is a byte counter (instead of a word counter), adds to the PCL allow a table size of 128 entries before the PCLATH needs to be modified.

Note:	Since the Program Counter is 21-bits, the uppercase PCLATU register may also need to be modified when doing computed gotos.

In this method of storing tables in PIC18CXXX devices, only one data value may be stored in each instruction location, and room on the return address stack is required. A better method of storing data in program memory is through the use of table reads and writes. Two bytes of data can now be stored in each instruction location.

Note:	Any write to the Program Counter (PCL) will cause the contents of PCLATU and PCLATH to be loaded into PCU and PCH, respectively.

7.4 Lookup Tables

Look-up tables instructions are implemented two ways in the PIC18CXXX devices. The computed goto is compatible with the PIC16CXXX and PIC17CXXX parts. Code written for those devices will run on the PIC18CXXX devices with minor modifications.

Table read instructions are implemented on the PIC17CXXX and PIC18CXXX devices. However, table operations on the PIC18CXXX work differently than on the PIC17CXXX.

7.4.1 Table Reads/Table Writes

Lookup table data may be stored 2 bytes per program word. By using TBLPTR and TABLAT, data may be retrieved from program memory one byte at a time as required.

Table writes to program memory can be executed as many times as desired. Remember that the technology of the program memory determines the outcome of the table write. Table writes to EPROM memory allow the program memory cell to go from a '1' state to a '0' state, but not the other direction. FLASH memory allows the cell to go from a '1' to a '0' and a '0' to a '1' (though typically a program memory word or block location is always written).

Example 7-1: PIC18CXXX Table Lookup

```
      MOVLW  BYTE_COUNT        ; Load the Byte Count value
      MOVWF  CNTR              ;    into CNTR
;
;;    MOVLW  UPPER(TBL_ADDR)   ; Load the Table Address
;;    MOVWF  TBLPTRU           ;   (on POR TBLPTRU = 0, so
;;                             ;   loading TBLPTRU is not
;;                             ;   required for conversions)
      MOVLW  HIGH(TBL_ADDR)    ; Load the Table Address
      MOVWF  TBLPTRH           ;
      MOVLW  LOW(TBL_ADDR)     ;
      MOVWF  TBLPTRL           ;
      LOOP1  TBLRD*+           ; Read value into TABLAT,
                               ;   Increment TBLPTR
      MOVFF  TABLAT, POSTINC0  ; Copy byte to RAM @ FSR0
                               ;   Increment FSR0
      DECFSZ CNTR              ; Read Byte Count locations
      GOTO   LOOP1             ; Read next Byte
```

Example 7-2: PIC17CXXX Table Lookup

```
      MOVLW  WORD_COUNT        ; Load the Word Count value
      MOVWF  CNTR              ;    into CNTR
;
      MOVLW  HIGH(TBL_ADDR)    ; Load the Table Address
      MOVWF  TBLPTRH           ;
      MOVLW  LOW(TBL_ADDR)     ;
      MOVWF  TBLPTRL           ;
      TABLRD 0, 1, DUMMY       ; Dummy read,
                               ;   Updates TABLATH
                               ;   Increments TBLPTR
      LOOP1 TLRD 1, INDF0      ; Read HI byte in TABLATH
      TABLRD 0, 1, INDF0       ; Read LO byte in TABLATL,
                               ;   update TABLATH:TABLATL,
                               ;   and increment TBLPTR
      DECFSZ CNTR              ; Read Word Count locations
      GOTO LOOP1               ; Read next word
```

Example 7-3: PIC16CXXX Table Lookup

```
         CLRF   CNTR              ;
TABLELP
         MOVF   CNTR, W           ; Place value in
                                  ;   WREG register
         CALL   TABLE1
         MOVWF  INDF
         INCF   FSR
         INCF   CNTR
         BTFSS  CNTR, 3           ;   CNTR = 00001000b?
         GOTO   TABLE_LP
         :
         :
TABLE1
         ADDWF  PCL               ; Enusure that table does
                                  ;   not cross 256 byte
                                  ;   page boundary.
         RETLW  'G'
         RETLW  'O'
         RETLW  ' '
         RETLW  'M'
         RETLW  'C'
         RETLW  'H'
         RETLW  'P'
         RETLW  '!'
```

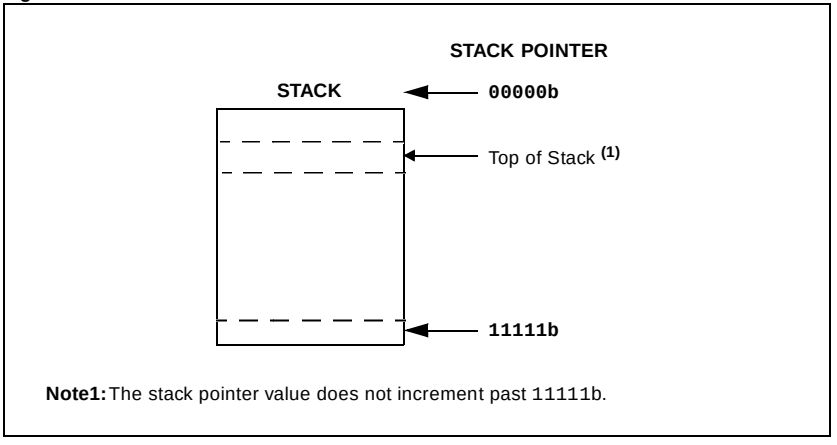
7.5 Stack

The stack allows a combination of up to 31 program calls and interrupts to occur. The stack contains the return address from this branch in program execution.

Enhanced MCU devices have an 31-level deep x 21-bit wide hardware stack. The stack space is not part of either program or data space and the stack pointer is not readable nor writable. The PC is PUSHed onto the stack when a CALL instruction is executed or an interrupt causes a branch. The stack is POPed in the event of a RETURN, RETLW or a RETFIE instruction execution. PCLATH is not modified when the stack is PUSHed or POPed.

After the PC is PUSHed onto the stack 31 times (without POPing any values off the stack), the 32nd PUSH over-writes the value from the 31st PUSH and sets the STKFUL bit while the STKPTR remains at 11111b. The 33rd PUSH overwrites the 32nd PUSH (and so on) while STKPTR remains 11111b. When the stack overflow enable bit is enabled a device reset will occur.

Figure 7-5: Stack Modification



Note1: The stack pointer value does not increment past 11111b.

Whenever the program branches, the return address is saved to the stack. Such branches include CALL, RCALL, or an interrupt. The stack pointer is incremented and PC<20:1> is PUSHed onto the return stack. PC<0> is always assumed to be 0. When a branch return is executed, the top of the stack is POPed to the Program Counter and the stack pointer is decremented. PCLATU and PCLATH are not affected during these branches.

The PC is word incremented by 2 after each instruction fetch during Q1 unless:

- Modified by a GOTO, CALL, RCALL, RETURN, RETLW, RETFIE, or branch instruction
- Modified by an interrupt response
- Due to a write to PCL by an instruction

Skips are equivalent to a forced NOP cycle at the skipped address.

7.6 Data Memory Organization

Data memory is made up of the Special Function Registers (SFR) area and the General Purpose Registers (GPR) area. The SFRs are used for control and status of the microcontroller and peripheral functions, while GPRs are the general area for user data storage and scratch pad operations.

Each register has a 12-bit address. This allows up to 4096 bytes of data memory. This memory is partitioned into 16 banks of 256 bytes that contain the General Purpose Registers (GPRs) and Special Function Registers (SFRs).

The data memory can be banked for both the GPR and SFR areas. Banking requires the use of BSR Register. Figure 7-6 shows the data memory map organizations, while Table 7-2 shows which banks will be used depending on the memory size of the devices.

SFRs start at the last location of Bank 15 (0xFFF) and work up. Once the SFR space ends, any lower locations in that bank may be implemented as GPRs. GPRs start at the first location of Bank 0 (0h) and work down. Any read of an unimplemented location will read as '0's.

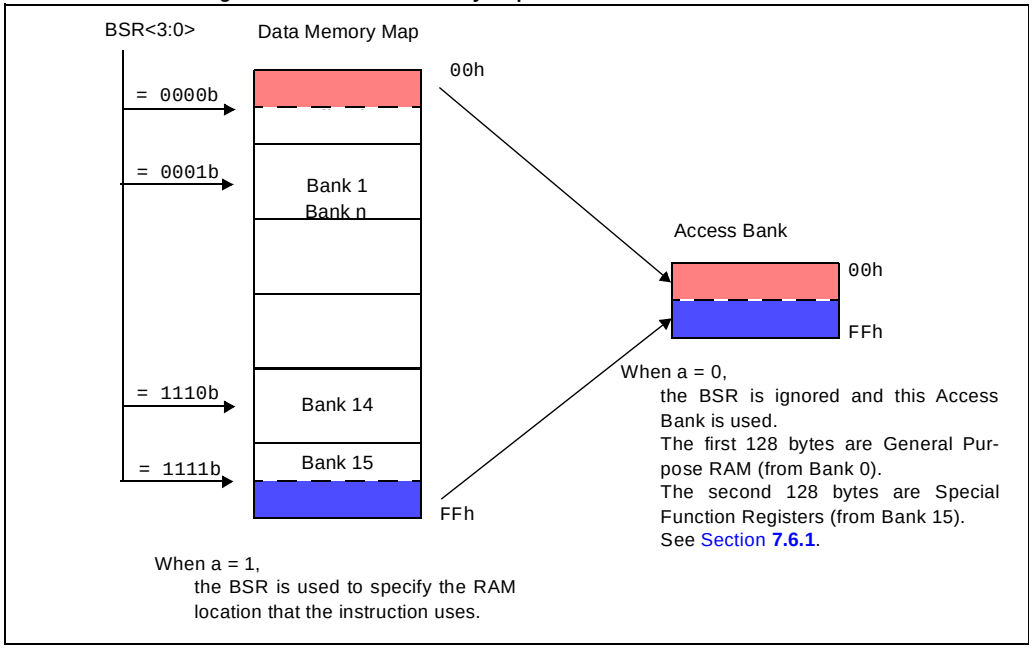
The Instruction set and architecture allows operations across all banks. To move values from one register to another register, the MOVFF instruction can be used. This is a two word / two cycle instruction.

The entire data memory can be accessed either directly or indirectly. Direct addressing may require the use of the BSR register. Indirect addressing requires the use of the File Select Registers (FSRs). Each FSR holds a 12-bit value that can access any location in the Data Memory map.

To ensure that commonly used registers (SFRs and select GPRs) can be accessed in a single cycle, regardless of the current BSR values, an Access Bank is implemented. This is explained in Section **7.6.1**.

The GPR area is banked to allow greater than 256 bytes of general purpose RAM to be addressed. SFRs are for the registers that control the peripheral and core functions.

Figure 7-6: The Data Memory Map and the Access Bank



7.6.1 Access Bank

The Access Bank is an architectural enhancement that is very useful for C compiler code optimization. The techniques used by the C compiler may also be useful for programs written in assembly.

This data memory region can be used for:

- Intermediate computational values
- Local variables of subroutines
- Faster context saving/switching of variables
- Common variables
- Faster evaluation/control of SFRs (no banking)

The Access Bank is comprised of the upper portion of Bank 15 (SFRs) and the lower portion of Bank 0 (GPR). These two sections will be referred to as Access RAM High and Access RAM Low, respectively. Figure 7-6 indicates the Access RAM areas. The actual size of memory used from Bank 0 and Bank 15 depends on the specific device. When appropriate, devices will use 128 bytes from Bank 0 (GPR) and 128 bytes from Bank 15 (SFR). In larger devices with more SFRs, the GPR Access bank size may be reduced to allocate that space to SFRs Access space.

A bit in the instruction word specifies if the operation is to occur in the bank specified by the BSR register or in the Access Bank. This bit is denoted by the 'a' bit (for access bit).

When forced in the Access Bank (a = '0'), the last address in Access RAM Low is followed by the first address in Access RAM High. Access RAM High maps the Special Function Registers so that these registers can be accessed without any software overhead. This is useful for testing status flags, modifying control bits, software stacks, and context saving of registers.

7.6.2 General Purpose Registers (GPR)

Enhanced MCU devices may have banked memory in the GPR area. GPRs are not initialized by a Power-on Reset and are unchanged on all other resets.

The register file can be accessed either directly, or indirectly, using the File Select Register (FSR). Some devices have areas that are shared across the data memory banks, so a read/write to that area will appear as the same location (value), regardless of the current bank. We refer to this area as the Common RAM.

Data RAM is available for use as GPR registers by all instructions. Most banks of data memory contain only GPR registers starting with bank 0. The top half of bank 15 (0xF80 to 0xFFF) contains SFRs.

Each data memory bank has 256 locations and can be addressed using an 8-bit address.

7.6.3 Special Function Registers

The SFRs are used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM.

The SFRs can be classified into two sets; those associated with the “core” function and those related to the peripheral functions. Those registers related to the “core” are described in this section, while those related to the operation of the peripheral features are described in the section of that peripheral feature.

The SFRs are typically distributed among the peripherals whose functions they control.

If the SFRs do not use all the available locations on a particular device, the unused locations will be unimplemented and read as '0's. In devices that have a high integration of features, some of the SFRs may be in banks other than bank 15. See Figure 7-7 for addresses for the SFRs. As new devices are introduced with new SFRs, this register map will be updated. Please refer to the device data sheet for that device’s register map.

Figure 7-7: Special Function Register Map

FFFh	TOSU	FDFh	INDF2 (2)	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2 (2)	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2 (2)	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2 (2)	FBCh	CCPR2H	F9Ch	—
FFBh	PCLATU	FDBh	PLUSW2 (2)	FBBh	CCPR2L	F9Bh	—
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	—
FF9h	PCL	FD9h	FSR2L	FB9h	—	F99h	—
FF8h	TBLPTRU	FD8h	STATUS	FB8h	—	F98h	—
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	—	F97h	—
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	—	F96h	TRISE
FF5h	TABLAT	FD5h	T0CON	FB5h	—	F95h	TRISD
FF4h	PRODH	FD4h	—	FB4h	—	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	LVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	—
FF0h	INTCON3	FD0h	RCON	FB0h	—	F90h	—
FEFh	INDF0 (2)	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—
FEEh	POSTINC0 (2)	FCEh	TMR1L	FAEh	RCREG	F8Eh	—
FEDh	POSTDEC0 (2)	FCDh	T1CON	FADh	TXREG	F8Dh	LATE
FECh	PREINC0 (2)	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD
FEBh	PLUSW0 (2)	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh	—	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	—	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	—	F88h	—
FE7h	INDF1 (2)	FC7h	SSPSTAT	FA7h	—	F87h	—
FE6h	POSTINC1 (2)	FC6h	SSPCON1	FA6h	—	F86h	—
FE5h	POSTDEC1 (2)	FC5h	SSPCON2	FA5h	—	F85h	—
FE4h	PREINC1 (2)	FC4h	ADRESH	FA4h	—	F84h	PORTE
FE3h	PLUSW1 (2)	FC3h	ADRESL	FA3h	—	F83h	PORTD
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	—	FA0h	PIE2	F80h	PORTA

Note 1: Unimplemented-registers are read as ’0’.
2: This is not a physical register.

7.6.4 Bank Select Register (BSR)

The need for a large general purpose memory space dictated a general purpose RAM banking scheme. A Special Function Register (named BSR) selects the currently active general purpose RAM bank. Only the lower middle of the BSR register (BSR<3:0>) is used. This allows access to potentially 16 banks. Direct long addressing mode is limited to 12-bit addresses. This also allows accesses to any of the 16 banks. BSR<7:4> will always read 0's, and writes have no effect. All data memory is implemented as static RAM.

A MOVLB bank instruction has been provided in the instruction set to assist in selecting banks.

If the currently selected bank is not implemented, any read will return all '0's, and all writes are ignored and the STATUS register bits will be set/cleared as appropriate for the instruction performed.

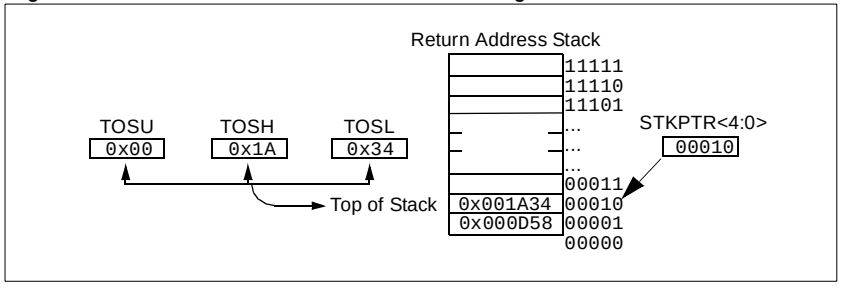
7.7 Return Address Stack

The Return Address Stack allows any combination of up to 31 program calls and interrupts to occur. The PC (Program Counter) is PUSHed onto the stack when a CALL or RCALL instruction is executed or an interrupt is acknowledged. The PC value is pulled off the stack on a RETURN, RETLW, or a RETFIE instruction. PCLATU and PCLATH are not affected by any of the return instructions.

The stack operates as a 31 word by 21-bit RAM and a 5-bit stack pointer (STKPTR), with the stack pointer initialized to 00000b after all resets. There is no RAM associated with stack pointer location 00000b. This is only a reset value. During a CALL type instruction causing a PUSH onto the stack, the stack pointer is first incremented and the RAM location pointed to by the stack pointer is written with the contents of the PC. During a RETURN type instruction causing a POP from the stack, the contents of the RAM location pointed to by the STKPTR register are transferred to the PC and then the stack pointer is decremented.

The stack space is not part of either program or data space and the stack pointer is neither readable nor writable. The address on the top of the stack is readable and writable through SFR registers. Data can also be PUSHed to or POPed from the stack using the top-of-stack SFRs. Status bits indicate if the stack pointer attempts to exceed the 31 levels provided. The stack does not wrap when the stack is PUSHed greater than 31 times.

Figure 7-8: Return Address Stack and Associated Registers



7.7.1 Top-Of-Stack Access

The Top-of-Stack (TOS) is readable and writable. Three register locations, TOSU, TOSH and TOSL, hold the contents of the stack location pointed to by the STKPTR register. This allows users to implement a software stack if necessary. After a CALL, RCALL or interrupt, the software can read the PUSHed value by reading the TOSU, TOSH and TOSL registers. These values can be placed on a user defined software stack. At return time, the software can replace the TOSU, TOSH and TOSL and do a return.

The user must disable the global interrupt enable bits during this time to prevent inadvertent stack operations.

Note: The user must disable interrupts when manipulating the stack.

7.7.2 PUSH and POP Instructions

Since the Top-of-Stack (TOS) is readable and writable, the ability to PUSH values onto the stack and pull values off the stack without disturbing normal program execution is a desirable option. To PUSH the current PC value onto the stack, a PUSH instruction can be executed. This will increment the stack pointer and load the current PC value onto the stack. TOSU, TOSH, and TOSL can then be modified to place data on the stack instead of a return address. This data may be a new return address. This may be done in the operation of a Real Time Operating System (RTOS).

The ability to pull the TOS value off of the stack and replace it with the value that was previously PUSHed onto the stack, without disturbing normal execution, is achieved by using the POP instruction. The POP instruction discards the current TOS by decrementing the stack pointer. The previous value PUSHed onto the stack then becomes the TOS value.

Example 7-4: Using the PUSH Instruction

```
MOVLW  Dummy_TOSU          ;
MOVWF  TOSU                ;
MOVLW  Dummy_TOSH          ;
MOVWF  TOSH                ;
MOVLW  Dummy_TOSL          ;
MOVWF  TOSL                ;
PUSH                       ;
```

Example 7-5: Using the POP Instruction

```
MOVFF  TOSU, PREINC1       ;
MOVFF  TOSH, PREINC1       ;
MOVFF  TOSL, PREINC1       ;
POP                        ;
```

7.7.3 Return Stack Pointer (STKPTR)

The STKPTR register contains the return stack pointer value and the overflow and underflow bits.

The stack overflow bit (STKFUL) and underflow bit (STKUNF) allow software verification of a stack condition. The STKFUL and STKUNF bits are cleared only after a POR reset.

After the PC is PUSHed onto the stack 31 times (without POPing any values off the stack), the 32nd PUSH over-writes the value from the 31st PUSH and sets the STKFUL bit, while the STKPTR remains at 11111b. The 33rd PUSH overwrites the 32nd PUSH (and so on), while STKPTR remains 11111b.

After the stack is POPed enough times to unload the stack, the next POP will return a value of zero to the PC and set the STKUNF bit while the STKPTR remains at 00000b. The next POP returns zero again (and so on), while STKPTR remains 00000b.

Note:	Returning a zero to the PC on an underflow has the effect of vectoring the program to the reset vector, where the stack conditions can be verified and appropriate actions can be taken.

The stack pointer can be accessed through the STKPTR register. The user may read and write the stack pointer values. This can be used by a Real Time Operating System (RTOS) for return stack maintenance. Figure 7-8 shows the STKPTR register. The value of the stack pointer will be 0 through 31. At reset, the stack pointer value will be 0. The stack pointer will increment when PUSHing and will decrement when POPing.

7.7.4 Stack Full/Underflow Resets

At the user's option, the overflow and underflow can cause a device reset to interrupt the program code. The reset is enabled with a configuration bit, STVREN. When the STVREN bit is disabled, a full or underflow condition will set the appropriate STKFUL or STKUNF bit but not cause a reset. When the STVREN bit is enabled, a overflow or underflow will set the appropriate STKFUL or STKUNF bit and then cause a device reset very similar in nature to the WDT reset. In either case, the STKFUL or STKUNF bits are only cleared by user software or a POR reset.

7.7.5 Fast Register Stack

A "fast interrupt return" option is available for interrupts. A fast register stack is provided for the STATUS, WREG, and BSR registers and are only one in depth. The stack is neither readable nor writable and is loaded with the current value of the corresponding register when the processor vectors for an interrupt. The values in the registers are then loaded back into the working registers if the fast return instruction is used to return from the interrupt.

Low or high priority interrupt PUSHes values into the stack registers. If both low and high priority interrupts are enabled, the stack registers cannot be used reliably for low priority interrupts. A high priority interrupt, if one occurs while servicing a low priority interrupt, will overwrite the stack registers stored by the low priority interrupt.

If high priority interrupts are not disabled during low priority interrupts, users must save the key registers in software during a low priority interrupt.

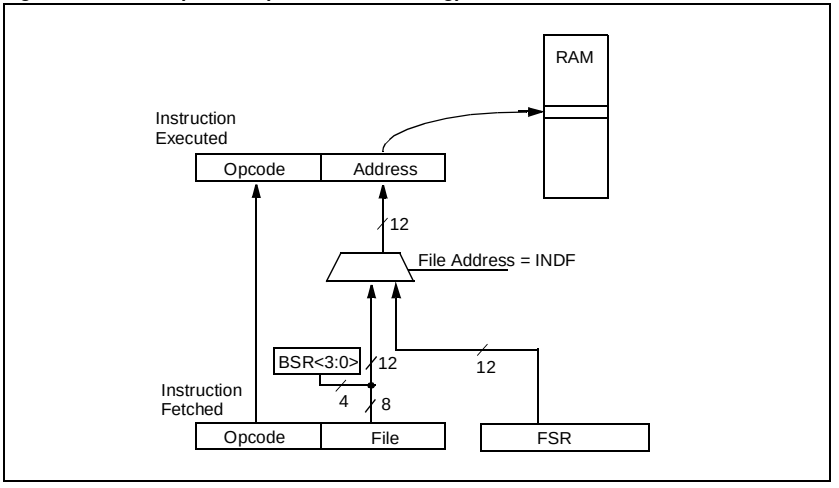
If no interrupts are used, the fast register stack can be used to restore the STATUS, WREG and BSR registers at the end of a subroutine call. To use the fast register stack for a subroutine call, a fast call instruction must be executed.

7.7.6 Indirect Addressing, INDF, and FSR Registers

Indirect addressing is a mode of addressing data memory where the data memory address in the instruction is not fixed. An SFR register is used as a pointer to the data memory location that is to be read or written. Since this pointer is in RAM, the contents can be modified by the program. This can be useful for data tables in the data memory and for software stacks. Figure 7-9 shows the operation of indirect addressing. This shows the moving of the value to the data memory address specified by the value of the FSR register.

Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself indirectly (FSR = '0') will read 00h. Writing to the INDF register indirectly results in a no-operation (although status bits may be affected). The FSR register contains a 12-bit address, which is shown in Figure 7-9.

Figure 7-9: FSR Operation (Indirect Addressing)



There are three indirect addressing registers. To address the entire Data Memory space (4096 bytes), these registers are 12 bits wide. To store the 12 bits of addressing information, two 8-bit registers are required. These indirect addressing registers are:

1. FSR0: composed of FSR0H:FSR0L
2. FSR1: composed of FSR1H:FSR1L
3. FSR2: composed of FSR2H:FSR2L

In addition, there are registers INDF0, INDF1 and INDF2, which are not physically implemented. Reading or writing to these registers activates indirect addressing, with the value in the corresponding FSR register being the address of the data.

If an instruction writes a value to INDF0, the value will be written to the address pointed to by FSR0H:FSR0L. A read from INDF1 actually reads the data from the address pointed to by FSR1H:FSR1L. INDFn can be used in code anywhere an operand can be used.

If the INDF0, INDF1 or INDF2 register is read indirectly via an FSR, all '0's are read (Zero bit is set). Similarly, if INDF0, INDF1or INDF2 is written to indirectly, the operation will be equivalent to a NOP, and the STATUS bits are not affected.

7.7.6.1 Indirect Addressing Operation

Each FSR register has an INDF register plus four addresses associated with it. The same INDFn, and FSRnH:FSRnL registers are used, but depending on the INDFn address selected, the FSRnH:FSRnL registers may be modified.

When a data access is done to the one of the five INDFn locations, the address selected will configure the FSRn register to:

- Do nothing to FSRn after an indirect access (no change) - INDFn
- Auto-decrement FSRn after an indirect access (post-decrement) - POSTDECn
- Auto-increment FSRn after an indirect access (post-increment) - POSTINCn
- Auto-increment FSRn before an indirect access (pre-increment) - PREINCn
- Use the value in the WREG register as an offset to FSRn. Do not modify the value of the WREG or the FSRn register after an indirect access (no change) - PLUSWn

When using the auto-increment or auto-decrement features, the effect on the FSR is not reflected in the STATUS register. For example, if the indirect address causes the FSR to equal '0', the Z bit will not be set.

Incrementing or decrementing an FSR affects all 12 bits. That is, when FSRnL overflows from an increment, FSRnH will be incremented automatically.

Adding these features allows the FSRn to be used as a stack pointer in addition to its uses for table operations in data memory.

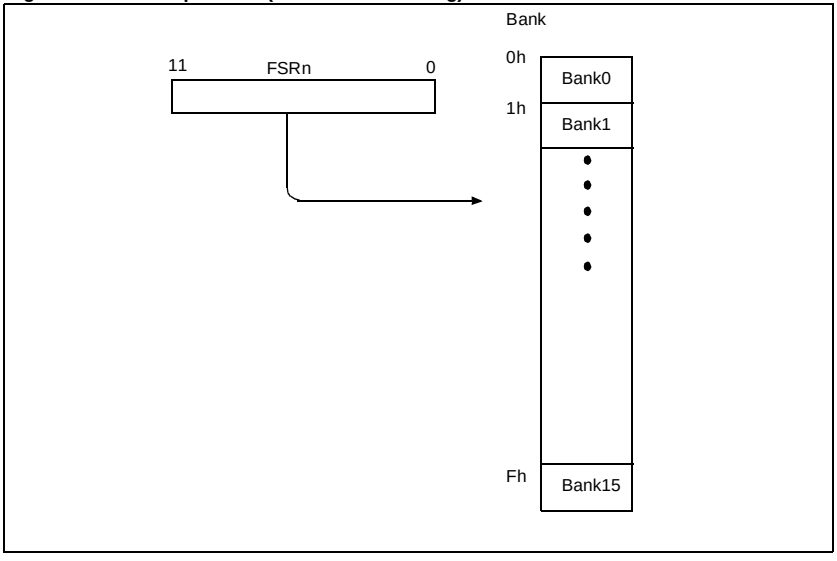
Each FSR has an address associated with it that performs an indexed indirect access. When a data access to this INDFn location (PLUSWn) occurs, the FSRn is configured to add the signed value in the WREG register and the value in FSR to form the address before an indirect access. The FSR value is not changed.

Note:	Accessing the PLUSWn address causes indexed indirect access. The addressed register is the addition of the value in the FSRn register and the **SIGNED** value in the WREG register.

If an FSR register contains a value that points to one of the INDFn, an indirect read will read 00h (Zero bit is set), while an indirect write will be equivalent to a NOP (STATUS bits are not affected).

If an indirect addressing operation is done where the target address is an FSRnH or FSRnL register, the write operation will dominate over the pre- or post-increment/decrement functions. Figure 7-10 shows the interaction of the FSR register and the data memory.

Figure 7-10: FSR Operation (Indirect Addressing)



Example 7-6 shows a simple use of indirect addressing to clear RAM (locations 20h-2Fh) in a minimum number of instructions. A similar concept could be used to move a defined number of bytes (block) of data to the USART transmit register (TXREG). The starting address of the block of data to be transmitted could easily be modified by the program.

Example 7-6: Indirect Addressing

```
        CLRF     FSR1H          ; Clear High byte of FSR
        MOVLW    0x20           ; Load Low byte of 20h
        MOVWF    FSR1L          ;
NEXT    CLRF     POSTINC1       ; Clear register and the
                                ;   increment
        BTFSS    FSR1L, 4       ;
        GOTO     NEXT           ;

CONTINUE                        ;
         :
```

7.8 Initialization

Example 7-7 shows how the bank switching occurs for direct addressing, while Example 7-8 shows some code to do initialization (clearing) of General Purpose RAM.

Example 7-7: Bank Switching

```
     CLRF   BSR            ; Clear BSR register (Bank0)
     :                     ;
     BSF    BSR, 0         ; Bank1
     :                     ;
     BCF    BSR, 0         ; Bank0
     :                     ;
     MOVLW  0x06           ;
     MOVWF  BSR            ; Bank6
     :                     ;
     BCF    BSR, 2         ; Bank2
     :                     ;
     BCF    BSR, 1         ; Bank0
```

Example 7-8: RAM Initialization

```
       CLRF   FSR1H
       CLRF   FSR1L
CLR_LP CLRF   POSTINC1     ; Clear location, increment
       MOVLW  0x0F         ; Bank is FSR1H:FSR1L
       SUBWF  FSR1H,W      ; Are we now in Bank 15?
       BNZ    CLR_LP       ; NO, continue to clear GPRs
CONTINUE                   ;
       :
       :
```

7.9 Design Tips

Question 1: ***I need to initialize RAM to '0's. What is an easy way to do that?***

Answer 1:

Example 7-8 shows this. If the device you are using does not use all 15 data memory banks, the value to compare FSR1H against will need to be modified.

Question 2: ***I want to convert from a PIC16C77 which has 368 bytes of RAM (across 4 banks). What is the best way to remap this memory?***

Answer 2:

In devices where the Access GPR region is greater or equal to 128 bytes and Bank 1 contains 256 bytes of GPR, the RAM should be partitioned with the RAM in Bank0 at locations 0x00 to 0x7F and all of Bank1. This allows a total of 384 bytes, which is larger than the 368 bytes in the PIC16C77. Now the BSR can be loaded with 0x01 (pointing to Bank1). All memory access are now either in Bank1 or the Access RAM, so no bank switching is required.

7.10 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is they may be written for the Base-Line, the Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the Memory Organization are:

Title	Application Note #
Implementing a Table Read	AN556

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

7.11 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Memory Organization description.

Section 8. Table Read/Table Write

HIGHLIGHTS

This section of the manual contains the following major topics:

8.1 Introduction

Enhanced devices have two memory spaces. The program memory space and the data memory space. The program memory space is 16 bits wide, while the data memory space is 8 bits wide. Table Reads and Table Writes have been provided to move data between these two memory spaces through an 8-bit register (TABLAT).

The operations that allow the processor to move data between the data and program memory spaces are:

- Table Read (TBLRD)
- Table Write (TBLWT)

Table Read operations retrieve data from program memory and place it into the data memory space. Figure 8-1 shows the operation of a Table Read with program and data memory.

Table Write operations store data from the data memory space into program memory. Figure 8-2 shows the operation of a Table Write with program and data memory.

Table operations work with byte entities. A table block containing data is not required to be word aligned, so a table block can start and end at any byte address. If Enhanced MCU instructions are being written to program memory, these instructions must be word aligned.

Figure 8-1: Table Read Operation

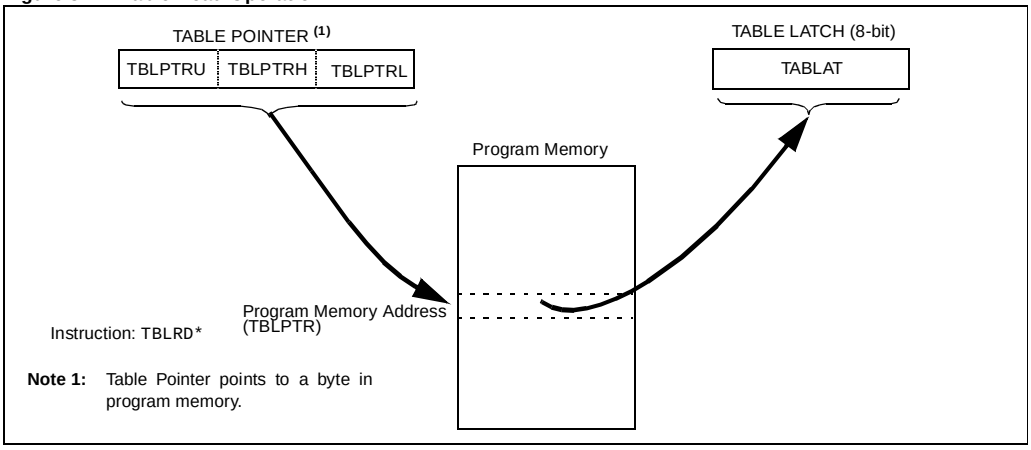
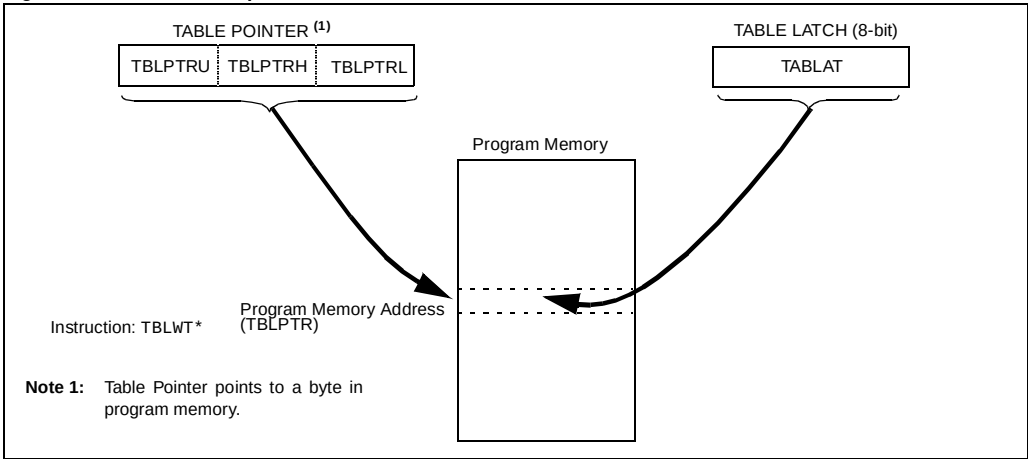


Figure 8-2: Table Write Operation



8.2 Control Registers

Several control registers are used in conjunction with the TBLRD and TBLWT instructions. These include the:

- RCON register
- MEMCON register
- TBLPTR registers
- TABLAT register

8.2.1 RCON Register

The LWRT bit specifies the operation of Table Writes to internal memory when the VPP voltage is applied to the $\overline{\text{MCLR}}$ pin. When the LWRT bit is set, the controller continues to execute user code, but long Table Writes are allowed (for programming internal program memory) from user mode. The LWRT bit can be cleared only by performing either a Power-On Reset (POR) or $\overline{\text{MCLR}}$ reset. The other bits of the RCON register do not relate to Table Read nor Table Write operation.

Register 8-1: RCON Register

R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
IPEN	LWRT	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
bit 7							bit 0

bit 7 **IPEN:** Interrupt Priority Enable
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (PIC16CXXX compatibility mode)

bit 6 **LWRT:** Long Write Enable
1 = Enable TABLE WRITE to internal program memory
0 = Disable TABLE WRITE to internal program memory

Note 1: Only cleared on a POR or $\overline{\text{MCLR}}$ reset.

Note 2: This bit has no effect on TBLWT instructions to external program memory.

bit 5 **Unimplemented**: Read as '0'

bit 4 **$\overline{\text{RI}}$:** Reset Instruction Flag bit
1 = No Reset instruction occurred
0 = A Reset instruction occurred

bit 3 **$\overline{\text{TO}}$:** Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 2 **$\overline{\text{PD}}$:** Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 1 **$\overline{\text{POR}}$:** Power-On Reset Status bit
1 = No Power-On Reset occurred
0 = A Power-On Reset occurred (must be set in software after a Power-On Reset occurs)

bit 0 **$\overline{\text{BOR}}$:** Brown-out Reset Status bit
1 = No Brown-out Reset or POR reset occurred
0 = A Brown-out Reset or POR reset occurred
(must be set in software after a Brown-out Reset occurs)

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

8.2.2 MEMCON Register

This register is only available on devices with a system bus to interface to external program memory. The MEMCON register is used to specify the operation of the 16-bit external system bus for Table Write operations. For additional information see the **"System Bus"** section. This register is not implemented in devices without a System Bus.

Register 8-2: MEMCON Register

R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0
EBDIS	PGRM	WAIT1	WAIT0	—	—	WM1	WM0
bit 7							bit 0

bit 7 **EBDIS:** External bus disable bit
This bit is used to disable the System Bus pins when in Extended Microcontroller mode. When disabled, the system bus pins become general purpose I/O.

1 = External system bus disabled, all I/O pin functions are enabled

0 = External system bus enabled, and I/O pin functions are disabled

bit 6 **PGRM:** Program RAM enable bit
This bit is used to configure internal GPR locations into the program memory map. This is useful for boot loaders in devices operating in microprocessor mode. The amount of GPR mapped will be device dependant

1 = GPR memory is mapped to internal program memory space. External program memory at these locations is unused. The internal GPR memory locations are disabled and returns 00h.

0 = GPR memory remains in data memory space. External program memory space is available.

bit 5-4 **WAIT1:WAIT0:** Wait Cycle count bits
Table reads and writes bus cycle wait count

11 = Table reads and writes will wait 0 TCY
10 = Table reads and writes will wait 1 TCY
01 = Table reads and writes will wait 2 TCY
00 = Table reads and writes will wait 3 TCY

bit 3-2 **Unimplemented:** Read as '0'

bit 1-0 **WM1:WM0:** Write Mode bit
Table Write write mode operation with 16-bit bus

11 = Word Write Mode: TABLAT0 and TABLAT1 word output, $\overline{WRH}$ active when TABLAT1 written

10 = Reserved

01 = Byte Select Mode: TABLAT data copied on both MS and LS Byte, $\overline{WRH}$ and ($\overline{UB}$ or $\overline{LB}$) will activate

00 = Byte Write Mode: TABLAT data copied on both MS and LS Byte, $\overline{WRH}$ or $\overline{WRL}$ will activate

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Note: Any register that has a protected bit requires that the entire register is protected. To write to a protected register requires the proper write sequence on the CMLK1:CMLK0 bits before this register can be updated.

8.2.3 TABLAT - Table Latch Register

The Table Latch (TABLAT) is an 8-bit register mapped into the SFR space. The Table Latch is used to hold 8-bit data during data transfers between program memory and data memory.

8.2.4 TBLPTR - Table Pointer Register

The Table Pointer (TBLPTR) addresses a byte within the program memory. The TBLPTR is comprised of three SFR registers (Table Pointer Upper byte, High byte, and Low byte). These three registers (TBLPTRU:TBLPTRH:TBLPTRL) join to form a 22-bit wide pointer. The low order 21-bits allows the device to address up to 2M bytes (or 1M words) of program memory space. The 22nd bit allows access to the Device ID, the User ID, and the Configuration bits.

The Table Pointer, TBLPTR, is used by the TBLRD and TBLWT instructions. These instructions can update the TBLPTR in one of four ways, based on the table operation. These operations are shown in Table 8-1. These operations on the TBLPTR only affect the low order 21-bits.

Table 8-1: Table Pointer Operations with TBLRD and TBLWT Instructions

Instruction	Operation on Table Pointer
TBLRD* TBLWT*	TBLPTR is not modified
TBLRD*+ TBLWT*+	TBLPTR is incremented after the read/write
TBLRD*- TBLWT*-	TBLPTR is decremented after the read/write
TBLRD+* TBLWT+*	TBLPTR is incremented before the read/write

8.3 Program Memory

The program memory can be either internal or external. External program memory requires that the device has the system bus interface. The operation of Table Reads and Table Writes is different depending on the location of the program memory (internal or external).

For the device to access external memory, the device needs to be operating in either extended microcontroller mode (some program memory is also internal), or microprocessor mode (no program memory is internal).

In this section, the discussion of Table Read and Table Write operation will be limited to the operation with internal program memory. For operation with external program memory, please refer to the **"System Bus"** section.

8.3.1 Internal Program Memory

The device selected will determine the program memory technology used. The Internal Program Memory can currently be one of three different memory technologies:

1. EPROM
2. FLASH
3. ROM

Depending on the memory technology the following statements can be made.

For EPROM devices:

- All unprogrammed memory locations will read back 0xFF (all bits set)
- Any bit that is set can be programmed clear
- Locations with data can be reprogrammed only if 1's are changed to 0's
- No cleared bit can be set unless the entire device is erased, which is only possible with windowed parts.

For FLASH devices:

- Any bit can be modified on a byte/block basis (individual bits cannot be modified)
- All writes occur with the write of an entire write block
- The size of the write block is device dependent

For ROM devices:

- All unprogrammed memory locations will read back 0xFF (all bits set)
- No program memory location can be modified

8.3.2 Internal Program Memory Operation

Table Read operation is independent of internal Program Memory Technology used. Table Write operation will be dependent on the memory technology of the internal Program Memory.

The notation “TBLRD instruction” means any of the four Table Read forms (TBLRD*, TBLRD*+, TBLRD*-, TBLRD+*) and the notation “TBLWT instruction” means any of the four Table Write forms (TBLWT*, TBLWT*+, TBLWT*-, TBLWT+*). For additional details on the operation of these instructions refer to the **“Instruction Set”** section.

8.3.2.1 Table Read Overview (TBLRD)

The TBLRD instructions are used to read data from program memory to data memory.

The Table Reads from program memory are performed one byte at a time.

The TBLPTR points to a byte address in program space. Executing a TBLRD instruction moves the contents of the addressed byte into the TABLAT register. In addition, the TBLPTR can be modified automatically for the next Table Read operation.

The TBLPTR can be automatically modified, depending on the form of the TBLRD instruction (see Table 8-1). All of the TBLRD instructions require two instruction cycles (TCY) to execute.

8.3.2.1.1 Effects of a Reset

The TABLAT register will retain the value read from program memory (if the instruction completed). The Table Pointer registers will not change, and the RCON register will be forced to the appropriate reset state.

8.3.2.2 Table Write Overview (TBLWT)

The TBLWT instructions are used to write data from data memory to program memory.

For devices with EPROM Program Memory, Table Writes are performed in pairs, one byte at a time. Table Writes to an even program memory address (TBLPTR<0> is clear) will load an internal memory latch from TABLAT, and is known as a short write. Table Writes to an odd program memory address (TBLPTR<0> is set) will start long writes. (TABLAT is programmed to the program word high byte, and the internal memory latch is programmed to the same word low byte).

For devices using another program memory technology, the operation may be different. Please refer to the device data sheet.

8.3.2.2.1 Memory Write Block Size

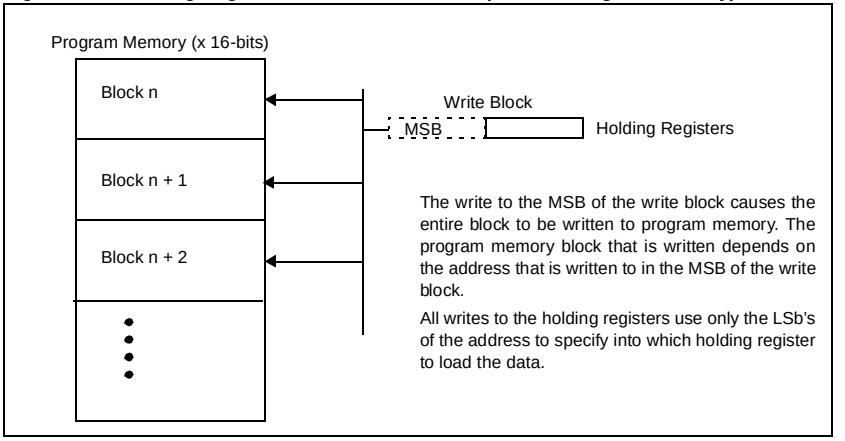
Depending on the device used, the write block size will be different. The larger the block size, the faster the entire internal program memory space can be programmed. This is because the EPROM/FLASH write time is much longer than the time to load the holding registers. For internal program memory, the write block size can vary from 2 bytes to many bytes (in FLASH devices).

A write to the Most Significant byte (MSB) of the block causes the entire block to be written to program memory. Writes to all other locations in the write block only modify the contents of the specified holding register.

After a write cycle has completed, the contents of the Write Block Holding Registers are forced set ('1's). This eases the writing of only a single byte within a program memory block. Figure 8-3 shows the write block for EPROM program memory. The write block size is 2 bytes.

If a single byte is to be programmed, the low (even) byte of the destination program word should be read using TBLRD*, then modified if required, and written back to the same address using TBLWT*+. Then the high (odd) byte should be read using TBLRD*, modified if required, and written back to the same address using a TBLWT instruction. The write to an odd address will cause a long write to begin (in EPROM program memory devices). This process ensures that existing data in either byte will not be changed unless desired.

Figure 8-3: Holding Registers and the Write Block (EPROM Program Memory)



8.3.2.2.2 EPROM Program Memory Operation

The long write is what actually programs the data into the internal memory. When a TBLWT instruction to the MSB of the write block occurs, instruction execution is halted. During this time, programming voltage and the data stored in internal latches is applied to program memory.

The sequence of steps to program the internal program memory is:

1. $\overline{MCLR}/V_{PP}$ pin must be at the programming voltage
2. LWRT bit must be set
3. TBLWT to the address of the MSB of the write block

If the LWRT bit is clear, a short write will occur and program memory will not be changed. If the TBLWT is not to the MSB of the write block, then the programming phase is not initiated.

Setting the LWRT bit enables long writes when the $\overline{MCLR}$ pin is taken to V_{PP} voltage. Once the LWRT bit is set, it can be cleared only by performing a Power-On Reset (POR) or $\overline{MCLR}$ reset.

To ensure that the memory location has been well programmed, a minimum programming time is required. The long write can be terminated after the programming time has expired by any event that can wake the controller from SLEEP. This may be a reset or an interrupt that operates during sleep. Having only one interrupt source enabled to terminate the long write ensures that no unintended interrupts will prematurely terminate the long write. Usable interrupt sources include:

- WDT
- A/D
- External Interrupts (INT0, INT1, or INT2)
- PORTB interrupt on change
- USART on address detect
- Timer1 in async counter mode or async external clock mode.
- Timer3 in async counter mode or async external clock mode.
- Capture
- SPI

8.3.2.2.3 Sequence of events

The sequence of events for programming an internal program memory location should be:

1. Enable the interrupt that terminates the long write. Disable all other interrupts.
2. Clear the source interrupt flag.
3. If Interrupt Service Routine (ISR) execution is desired when the device wakes, enable global interrupts (GIE, GIEH, or GIEL).
4. Set LWRT bit in RCON register.
5. Raise $\overline{MCLR}/V_{PP}$ pin to the programming voltage, V_{PP}.
6. Clear the WDT (if enabled).
7. Set the interrupt source to interrupt at the required time.
8. Load the desired Table Pointer Address.
9. Execute the Table Write for the lower (even) byte. This will be a short write.
10. Execute the Table Write for the upper (odd) byte. This will be a long write. The controller will halt instruction execution while programming. The interrupt wakes the controller.
11. If GIE was set, service the interrupt request.
12. If more locations to program, go to step 2.
13. Lower $\overline{MCLR}/V_{PP}$ pin to V_{DD}.
14. Verify the memory location (Table Read).

8.3.2.2.4 Interrupts

The long write must be terminated by a RESET or any interrupt that can wake the controller from SLEEP mode. Usable interrupt sources include:

- WDT
- A/D
- INT0
- INT1
- INT2
- RB<7:4> interrupt on change
- USART on address detect
- Timer1 in asynchronous counter mode or asynchronous external clock mode
- Timer3 in asynchronous counter mode or asynchronous external clock mode

The interrupt source must have its interrupt enable bit set. When the source sets its interrupt flag, programming will terminate. This will occur regardless of the settings of interrupt priority bits, the GIE/GIEH bit, or the PIE/GIEL bit.

Depending on the states of interrupt priority bits, the GIE/GIEH bit, or the PIE/GIEL bit, program execution can either be vectored to the high or low priority Interrupt Service Routine (ISR), or resume program execution.

In either case, the interrupt flag will not be automatically cleared when programming is terminated, and will need to be cleared by the software.

Table 8-2: SLEEP Mode, Interrupt Enable Bits and Interrupt Results

Interrupt Source	GIE/ GIEH	PIE/ GIEL	Priority	Interrupt Enable	Interrupt Flag	Action
Any interrupt source that operates during SLEEP	X	X	X	0 (default)	X	SLEEP mode continues even if interrupt flag becomes set during SLEEP.
	X	X	X	1	0	SLEEP mode continues, will wake when Interrupt flag is set.
	0 (default)	0 (default)	X	1	1	Wakes controller, terminates long write, executes next instruction. Interrupt flag not cleared.
	0 (default)	1	1 high priority (default)	1	1	Wakes controller, terminates long write, executes next instruction. Interrupt flag not cleared.
	1	0 (default)	0 low	1	1	Wakes controller, terminates long write, executes next instruction. Interrupt flag not cleared.
	0 (default)	1	0 low	1	1	Wakes controller, terminates long write, branches to low priority interrupt vector. Interrupt flag can be cleared by ISR.
	1	0 (default)	1 high priority (default)	1	1	Wakes controller, terminates long write, branches to high priority interrupt vector. Interrupt flag can be cleared by ISR.

8.3.2.2.5 Unexpected Termination of Write Operations

If a table write operation is terminated by an unplanned event, such as loss of power, an unexpected reset, or an interrupt that was not disabled, the memory location just programmed should be verified and reprogrammed if needed.

For applications where a loss of power could occur, a Brown-out reset circuit is recommended to ensure that the write operation is terminated cleanly. This reduces the possibility that a programmed location could not be reprogrammed to the desired value.

8.3.2.2.6 Effects of a RESET

A device reset during a long write may cause the location being programmed to be incompletely programmed. The location should be verified and reprogrammed if needed. A device reset during a write (short) will not effect the value in the TABLAT register (if the write cycle completed). The RCON register will be forced into the appropriate reset state.

8.4 Enabling Internal Programming

There is a combination of actions that need to occur to enable programming of the internal program memory. These modes are entered by applying the VIHH voltage on the MCLR/VPP pin and either setting the LWRT bit (self-programming) or having RB6 and RB7 at a low level when VIHH was detected (ICSP mode).

8.4.1 Programming Modes

Table 8-3 shows the device operating modes depending on the state of the MCLR pin, the LWRT bit, and the RB7:RB6.

Table 8-3: Device Programming Mode (Depending on MCLR voltage and LWRT bit and RB7 and RB6 pins)

MCLR/VPP Voltage	LWRT	RB6	RB7	OPERATING MODE
VPP	0	0	0	ICSP
VPP	0	1	X	Reserved
VPP	0	X	1	Reserved
VPP	1	X	X	Normal execution, long Table Writes enabled
VDD	1	X	X	Normal execution, short Table Writes only
VSS	0 (1)	X	X	In Device Reset

Legend: X = Don't care.

Note 1: The LWRT bit is cleared by any device reset.

8.5 External Program Memory Operation

Regardless of the system bus mode selected, Table Reads and Table Writes to external memory execute in 2 TCY. For information regarding the modes and waveforms of the System Bus, see the **"System Bus"** section. All further details of external program memory and table operations will be described in that section.

8.6 Initialization

Example 8-1 shows the initialization of the Table pointer and the FSR0 register to read the value from Program memory to a RAM buffer (starting at address RAMBUFADDR).

Example 8-2 shows the sequence to initialize these same registers and then write a word to the internal EPROM Program memory.

Example 8-1: Initialization to do a Table Read

```
    LFSR   FSR0, RAMBUFADDR     ;
    MOVLW  UPPER (Read Table)   ;
    MOVWF  TBLPTRU              ;
    MOVLW  HIGH (Read Table)    ;
    MOVWF  TBLPTRH              ;
    MOVLW  LOW (Read Table)     ;
    MOVWF  TBLPTRL              ;
    TBLRD*+                     ; Read location and then increment
                                ;   the table pointer
    MOVFF  TABLAT, POSTINC0     ; Copy contents of table latch to the
                                ;   indirect address and then
                                ;   increment the indirect address
                                ;   pointer.
```

Example 8-2: Initialization to do a Table Write (to internal EPROM memory)

```
    :                           ; Set up interrupts to terminate
    :                           ;   long write
    LFSR   FSR0, RAMBUFADDR     ;
    MOVLW  UPPER (Read Table)   ;
    MOVWF  TBLPTRU              ;
    MOVLW  HIGH (Read Table)    ;
    MOVWF  TBLPTRH              ;
    MOVLW  LOW (Read Table)     ;
    MOVWF  TBLPTRL              ;
    MOVFF  POSTINC0, TABLAT     ; Load table latch with value
                                ;   to write
    TBLWT*+                     ; Write to holding register
    MOVFF POSTINC0, TABLAT      ; Load second byte to table latch
    TBLWT*+                     ; Write to MSB, (odd address)
                                ;   start long write
```

8.7 Design Tips

Question 1: ***The location I programmed does not contain the value I wrote. What is wrong?***

Answer 1:

There are several possibilities. These include, but are not limited to:

- The maximum time requirement of the long write time was not met (for internal program memory).
- The address of the location to program was changed during the write cycle.
- A device reset occurred during the write cycle.
- If the program memory is EPROM or EOTP, then required overprogramming was not done.
- An Interrupt flag may have been set, so the long write cycle was immediately completed (violated long write time specification).

Question 2: ***Occasionally the device hangs. What is this?***

Answer 2:

Your program may have executed a long write, and the device may not have the interrupts/modules set up to terminate this long write.

Question 3: ***When programming the program memory are there any required algorithm algorithms to follow?***

Answer 3:

The programming algorithm will be dependent on the memory technology of the device. For complete information on device programming please refer to the device's programming specifications.

8.8 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is, they may be written for the Base-Line, the Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the oscillator are:

Title	Application Note #

No related application notes at this time.

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

8.9 Revision History

This is the initial released revision of the Enhanced MCU Table Read/Table Write description.

Section 9. System Bus

Please check the Microchip web site for Revision B of the System Bus Section.

9.1 Revision History

Revision A

This is the initial released revision of the Enhanced MCU System Bus module description.

Section 10. Interrupts

HIGHLIGHTS

This section of the manual contains the following major topics:

10.1 Introduction

Interrupts can come from many sources. These sources currently include:

- External interrupt from the INT, INT1, and INT2 pins
- Change on RB7:RB4 pins
- TMR0 Overflow
- TMR1 Overflow
- TMR2 Overflow
- TMR3 Overflow
- USART Interrupts
  - Receive buffer full
  - Transmit buffer empty
- SSP Interrupt
- SSP I^2C bus collision interrupt
- A/D conversion complete
- CCP interrupt
- LVD Interrupt
- Parallel Slave Port
- CAN interrupts
  - Receive buffer 1 full
  - Receive buffer 2 full
  - Receive invalid
  - Transmit buffer 0 empty
  - Transmit buffer 1 empty
  - Transmit buffer 2 empty
  - Bus wakeup
  - Bus invalid error

As other peripheral modules are developed, they will have interrupt sources. These sources will map into the 10 registers used in the control and status of interrupts. These registers are:

- INTCON
- INTCON1
- INTCON2
- INTCON3
- PIR1
- PIR2
- PIE1
- PIE2
- IPR1
- IPR2

The INTCON register contains the GIE/GIEH bit. This is the Global Interrupt Enable bit. When this bit is set, all interrupts are enabled. If needed for any single device, additional INTCON, PIR, PIE, and IPR registers will be defined.

10.1.1 Interrupt Priority

There are two interrupt vectors. One interrupt vector is for high priority interrupts and is located at address 000008h. The other interrupt vector is for low priority interrupts and is located at address 000018h.

When a valid interrupt occurs, program execution vectors to one of these interrupt vector addresses and the corresponding Global Interrupt Enable bit (GIE, GIEH, or GIEL) is automatically cleared. In the interrupt service routine, the source(s) of the interrupt can be determined by testing the interrupt flag bits. The interrupt flag bit(s) must be cleared before re-enabling interrupts to avoid infinite interrupt requests. Most flag bits are required to be cleared by the application software. There are some flag bits that are automatically cleared by the hardware.

When an interrupt condition is met, that individual interrupt flag bit will be set regardless of the status of its corresponding mask bit .

For external interrupt events, such as the RB0/INT0 pin or PORTB change interrupt, the interrupt latency will be three or four instruction cycles. The exact latency depends when the interrupt event occurs. The interrupt latency is the same for one or two cycle instructions.

The “return from interrupt” instruction, RETFIE, can be used to mark the end of the interrupt service routine. When this instruction is executed, the stack is “POPed” and the GIE bit is set (to re-enable interrupts).

Figure 10-1: Interrupt Logic High Level Block Diagram

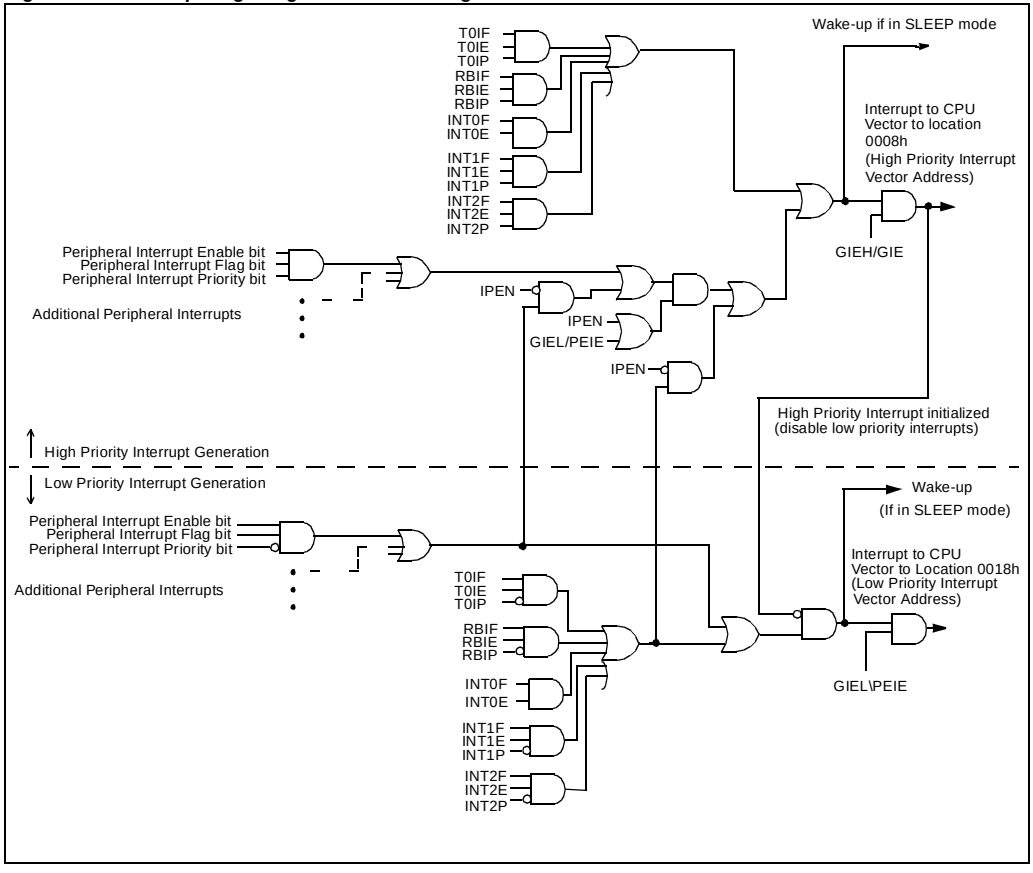


Figure 10-2: High Priority Interrupt Logic Block Diagram

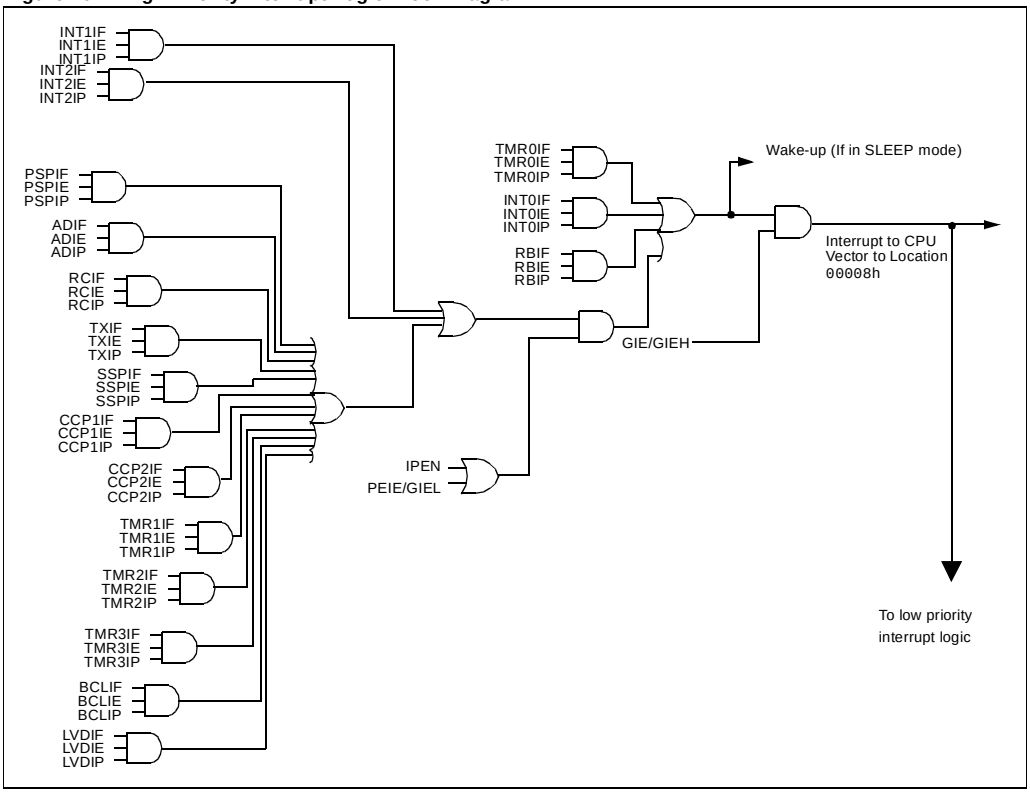
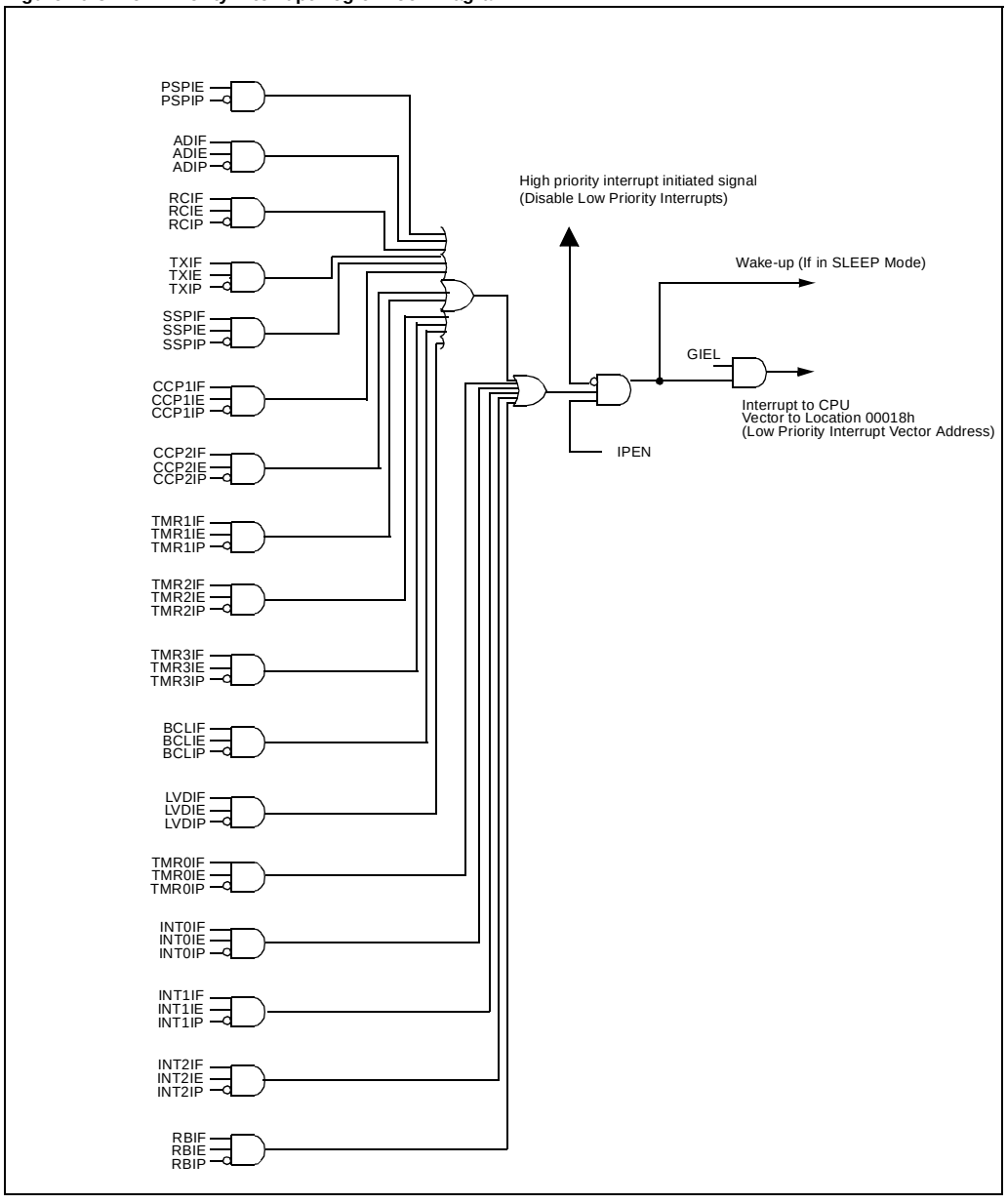


Figure 10-3: Low Priority Interrupt Logic Block Diagram



10.2 Control Registers

Generally devices have a minimum of four registers associated with interrupts. The INTCON register contains the Global Interrupt Enable bit, GIE, as well as the Peripheral Interrupt Enable bit, PEIE, the PIE / PIR register pair that enables the peripheral interrupts and displays the interrupt flag status, and the Interrupt Priority Register (IPR) that controls whether the interrupt source is a high priority or low priority interrupt.

10.2.1 INTCON Register

The INTCON Registers are readable and writable registers that contain various enable, priority, and flag bits.

Register 10-1: INTCON Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
When IPEN = 0:

1 = Enables all un-masked interrupts
0 = Disables all interrupts

When IPEN = 1:

1 = Enables all interrupts
0 = Disables all interrupts

bit 6 **PEIE/GEIL:** Peripheral Interrupt Enable bit
When IPEN = 0:

1 = Enables all un-masked peripheral interrupts
0 = Disables all peripheral interrupts

When IPEN = 1:

1 = Enables all low peripheral interrupts
0 = Disables all priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

Register 10-2: INTCON2 Register

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**: PORTB Pull-up Enable bit
1 = All PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG0**:External Interrupt0 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 5 **INTEDG1**: External Interrupt1 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 4 **INTEDG2**: External Interrupt2 Edge Select bit
1 = Interrupt on rising edge
0 = Interrupt on falling edge

bit 3 **Unimplemented:** Read as '1'

bit 2 **TMR0IP**: TMR0 Overflow Interrupt Priority bit
1 = TMR0 Overflow Interrupt is a high priority event
0 = TMR0 Overflow Interrupt is a low priority event

bit 1 **Unimplemented:** Read as '1'

bit 0 **RBIP**: RB Port Change Interrupt Priority bit
1 = RB Port Change Interrupt is a high priority event
0 = RB Port Change Interrupt is a low priority event

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

Register 10-3: INTCON3 Register

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

bit 7 **INT2IP:** INT2 External Interrupt Priority bit
1 = INT2 External Interrupt is a high priority event
0 = INT2 External Interrupt is a low priority event

bit 6 **INT1IP:** INT1 External Interrupt Priority bit
1 = INT1 External Interrupt is a high priority event
0 = INT1 External Interrupt is a low priority event

bit 5 **Unimplemented:** Read as '0'

bit 4 **INT2IE:** INT2 External Interrupt Enable bit
1 = Enables the INT2 external interrupt
0 = Disables the INT2 external interrupt

bit 3 **INT1IE:** INT1 External Interrupt Enable bit
1 = Enables the INT1 external interrupt
0 = Disables the INT1 external interrupt

bit 2 **Unimplemented:** Read as '0'

bit 1 **INT2IF:** INT2 External Interrupt Flag bit
1 = The INT2 external interrupt occurred (must be cleared in software)
0 = The INT2 external interrupt did not occur

bit 0 **INT1IF:** INT1 External Interrupt Flag bit
1 = The INT1 external interrupt occurred (must be cleared in software)
0 = The INT1 external interrupt did not occur

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Note: Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

10.2.2 PIE Register(s)

Depending on the number of peripheral interrupt sources, there may be multiple Peripheral Interrupt Enable registers (such as PIE1 and PIE2). These registers contain the individual enable bits for the peripheral interrupts. These registers will be generically referred to as PIE.

Note:	If the device has a PIE register and IPEN = 0, the PEIE bit must be set to enable any of the peripheral interrupts.

Although the PIE register bits have a general bit location with each register, future devices may not have consistent placement. Bit location inconsistencies will not be a problem if you use the supplied Microchip Include files for the symbolic use of these bits. This will allow the Assembler/Compiler to automatically take care of the placement of these bits by specifying the correct Register number and bit name.

Register 10-4: PIE Peripheral Interrupt Enable Registers

R/W-0
(Note 1)
bit 7 ... bit 0

bit **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

bit **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit **TMR3IE**: TMR3 Overflow Interrupt Enable bit
1 = Enables the TMR3 overflow interrupt
0 = Disables the TMR3 overflow interrupt

bit **CCPxIE**: CCPx Interrupt Enable bit
1 = Enables the CCPx interrupt
0 = Disables the CCPx interrupt

bit **ECCPxIE**: Enhanced CCPx Interrupt Enable bit
1 = Enables the CCPx interrupt
0 = Disables the CCPx interrupt

bit **SSPIE**: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit **MSSPIE**: Master Synchronous Serial Port Interrupt Enable bit
1 = Enables the MSSP interrupt
0 = Disables the MSSP interrupt

bit **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit **IRXIE:** CAN Invalid Received message Interrupt Enable bit
1 = Enable invalid message received interrupt
0 = Disable invalid message received interrupt

bit **WAKIE:** CAN Bus Activity Wake-up Interrupt Enable bit
1 = Enable Bus Activity Wake-up Interrupt
0 = Disable Bus Activity Wake-up Interrupt

bit **ERRIE:** CAN bus Error Interrupt Enable bit
1 = Enable CAN bus Error Interrupt
0 = Disable CAN bus Error Interrupt

bit **TXB2IE:** CAN Transmit Buffer 2 Interrupt Enable bit
1 = Enable Transmit Buffer 2 Interrupt
0 = Disable Transmit Buffer 2 Interrupt

bit **TXB1IE:** CAN Transmit Buffer 1 Interrupt Enable bit
1 = Enable Transmit Buffer 1 Interrupt
0 = Disable Transmit Buffer 1 Interrupt

bit **TXB0IE:** CAN Transmit Buffer 0 Interrupt Enable bit
1 = Enable Transmit Buffer 0 Interrupt
0 = Disable Transmit Buffer 0 Interrupt

bit **RXB1IE:** CAN Receive Buffer 1 Interrupt Enable bit
1 = Enable Receive Buffer 1 Interrupt
0 = Disable Receive Buffer 1 Interrupt

bit **RXB0IE:** CAN Receive Buffer 0 Interrupt Enable bit
1 = Enable Receive Buffer 0 Interrupt
0 = Disable Receive Buffer 0 Interrupt

bit **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit **PSPIE:** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit **EEIE**: EE Write Complete Interrupt Enable bit
1 = Enables the EE write complete interrupt
0 = Disables the EE write complete interrupt

bit **CMIE**: Comparator Interrupt Enable bit
1 = Enables the Comparator interrupt
0 = Disables the Comparator interrupt

bit **BCLIE**: Bus Collision Interrupt Enable bit
1 = Enabled
0 = Disabled

bit **LVDIE**: Low-voltage Detect Interrupt Enable bit
1 = Enabled
0 = Disabled

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Note 1: The bit position of the enable bits is device dependent. Please refer to the device data sheet for bit placement.

10.2.3 PIR Register(s)

Depending on the number of peripheral interrupt sources, there may be multiple Peripheral Interrupt Flag registers (PIR1, PIR2). These registers contain the individual flag bits for the peripheral interrupts. These registers will be generically referred to as PIR.

Note 1:	Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>).
Note 2:	User software should ensure the appropriate interrupt flag bits are cleared prior to enabling an interrupt and after servicing that interrupt.

Although the PIR bits have a general bit location within each register, future devices may not have consistent placement. It is recommended that you use the supplied Microchip Include files for the symbolic use of these bits. This will allow the Assembler/Compiler to automatically take care of the placement of these bits within the specified register.

Register 10-5: PIR Register

R/W-0
(Note 1)
bit 7 ... bit 0

bit **TMR1IF:** TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed
(must be cleared in software)
0 = TMR1 register did not overflow

bit **TMR2IF:** TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred
(must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit **TMR3IF**: TMR3 Overflow Interrupt Flag bit
1 = TMR3 register overflowed
(must be cleared in software)
0 = TMR3 register did not overflow

bit **CCPxIF**: CCPx Interrupt Flag bit
Capture Mode
1 = A TMR1 register capture occurred
(must be cleared in software)
0 = No TMR1 register capture occurred

Compare Mode
1 = A TMR1 register compare match occurred
(must be cleared in software)
0 = No TMR1 register compare match occurred

PWM Mode
Unused in this mode

bit **ECCPxIF**: Enhanced CCPx Interrupt Flag bit
Capture Mode
1 = A TMR1 register capture occurred
(must be cleared in software)
0 = No TMR1 register capture occurred

Compare Mode
1 = A TMR1 register compare match occurred
(must be cleared in software)
0 = No TMR1 register compare match occurred

PWM Mode
Unused in this mode

bit **IRXIF:** CAN Invalid Received message Interrupt Flag bit

1 = An invalid message has occurred on the CAN bus
0 = No invalid message on CAN bus

bit **WAKIF:** CAN Bus Activity Wake-up Interrupt Flag bit

1 = Activity on CAN bus has occurred
0 = No activity on CAN bus

bit **ERRIF:** CAN bus Error Interrupt Flag bit

1 = An error has occurred in the CAN module (multiple sources)
0 = No CAN module errors

bit **TXB2IF:** CAN Transmit Buffer 2 Interrupt Flag bit

1 = Transmit Buffer 2 has completed transmission of a message, and may be re-loaded
0 = Transmit Buffer 2 has not completed transmission of a message

bit **TXB1IF:** CAN Transmit Buffer 1 Interrupt Flag bit

1 = Transmit Buffer 1 has completed transmission of a message, and may be re-loaded
0 = Transmit Buffer 1 has not completed transmission of a message

bit **TXB0IF:** CAN Transmit Buffer 0 Interrupt Flag bit

1 = Transmit Buffer 0 has completed transmission of a message, and may be re-loaded
0 = Transmit Buffer 0 has not completed transmission of a message

bit **RXB1IF:** CAN Receive Buffer 1 Interrupt Flag bit

1 = Receive Buffer 1 has received a new message
0 = Receive Buffer 1 has not received a new message

bit **RXB0IF:** CAN Receive Buffer 0 Interrupt Flag bit

1 = Receive Buffer 0 has received a new message
0 = Receive Buffer 0 has not received a new message

bit **SSPIF**: Synchronous Serial Port Interrupt Flag bit
1 = The transmission/reception is complete
(must be cleared in software)
0 = Waiting to transmit/receive

bit **MSSPIF**: Master Synchronous Serial Port Interrupt Flag bit
1 = The transmission/reception is complete
(must be cleared in software)
0 = Waiting to transmit/receive

bit **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer, RCREG, is full
(cleared when RCREG is read)
0 = The USART receive buffer is empty

bit **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer, TXREG, is empty
(cleared when TXREG is written)
0 = The USART transmit buffer is full

bit **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed
(must be cleared in software)
0 = The A/D conversion is not complete

bit **PSPIF:** Parallel Slave Port Read/Write Interrupt Flag bit
1 = A read or a write operation has taken place
(must be cleared in software)
0 = No read or write has occurred

bit **EEIF**: EE Write Complete Interrupt Flag bit
1 = The data EEPROM write operation is complete
(must be cleared in software)
0 = The data EEPROM write operation is not complete

bit **CMIF**: Comparator Interrupt Flag bit
1 = Comparator input has changed
(must be cleared in software)
0 = Comparator input has not changed

bit **BCLIF**: Bus Collision Interrupt Flag bit
1 = A Bus Collision occurred
(must be cleared in software)
0 = No Bus Collision occurred

bit **LVDIF**: Low-voltage Detect Interrupt Flag bit
1 = A Low Voltage condition occurred
(must be cleared in software)
0 = The device voltage is above the Low Voltage Detect trip point

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Note 1: The bit position of the enable bits is device dependent. Please refer to the device data sheet for bit placement.

10.2.4 IPR Register

Depending on the number of peripheral interrupt sources, there may be multiple Peripheral Interrupt Priority registers (such as IPR1 and IPR2). These registers contain the individual priority bits for the peripheral interrupts. These registers will be generically referred to as IPR. If the device has an IPR register and IPEN = 0, the PEIE bit must be set to enable any of these peripheral interrupts.

Note:	The IP bit specifies the priority of the peripheral interrupt.

Although the IPR register bits have a general bit location with each register, future devices may not have consistent placement. Bit location inconsistencies will not be a problem if you use the supplied Microchip Include files for the symbolic use of these bits. This will allow the Assembler/Compiler to automatically take care of the placement of these bits by specifying the correct register and bit name.

Register 10-6:IPR Peripheral Interrupt Priority Register

R/W-0
(Note 1)
bit 7 ... bit 0

bit **TMR1IP**: TMR1 Overflow Interrupt Priority bit
1 = TMR1 Overflow Interrupt is a high priority event
0 = TMR1 Overflow Interrupt is a low priority event

bit **TMR2IP**: TMR2 to PR2 Match Interrupt Priority bit
1 = TMR2 to PR2 Match Interrupt is a high priority event
0 = TMR2 to PR2 Match Interrupt is a low priority event

bit **TMR3IP**: TMR3 Overflow Interrupt Priority bit
1 = TMR3 Overflow Interrupt is a high priority event
0 = TMR3 Overflow Interrupt is a low priority event

bit **CCPxIP**: CCPx Interrupt Priority bit
1 = CCPx Interrupt is a high priority event
0 = CCPx Interrupt is a low priority event

bit **ECCPxIP**: Enhanced CCPx Interrupt Priority bit
1 = Enhanced CCPx Interrupt is a high priority event
0 = Enhanced CCPx Interrupt is a low priority event

bit **MSSPIP**: Master Synchronous Serial Port Interrupt Priority bit
1 = Master Synchronous Serial Port Interrupt is a high priority event
0 = Master Synchronous Serial Port Interrupt is a low priority event

bit **SSPIP**: Synchronous Serial Port Interrupt Priority bit
1 = Synchronous Serial Port Interrupt is a high priority event
0 = Synchronous Serial Port Interrupt is a low priority event

bit **RCIP**: USART Receive Interrupt Priority bit
1 = USART Receive Interrupt is a high priority event
0 = USART Receive Interrupt is a low priority event

bit **TXIP**: USART Transmit Interrupt Priority bit
1 = USART Transmit Interrupt is a high priority event
0 = USART Transmit Interrupt is a low priority event

bit **ADIP**: A/D Converter Interrupt Priority bit
1 = A/D Converter Interrupt is a high priority event
0 = A/D Converter Interrupt is a low priority event

bit **PSPIP:** Parallel Slave Port Read/Write Interrupt Priority bit
1 = Parallel Slave Port Read/Write Interrupt is a high priority event
0 = Parallel Slave Port Read/Write Interrupt is a low priority event

bit **IRXIP:** CAN Invalid Received message Interrupt Priority bit
1 = CAN Invalid Received message Interrupt is a high priority event
0 = CAN Invalid Received message Interrupt is a low priority event

bit **WAKIP:** CAN Bus Activity Wake-up Interrupt Priority bit
1 = CAN Bus Activity Wake-up Interrupt is a high priority event
0 = CAN Bus Activity Wake-up Interrupt is a low priority event

bit **ERRIP:** CAN bus Error Interrupt Priority bit
1 = CAN bus Error Interrupt is a high priority event
0 = CAN bus Error Interrupt is a low priority event

bit **TXB2IP:** CAN Transmit Buffer 2 Interrupt Priority bit
1 = CAN Transmit Buffer 2 Interrupt is a high priority event
0 = CAN Transmit Buffer 2 Interrupt is a low priority event

bit **TXB1IP:** CAN Transmit Buffer 1 Interrupt Priority bit
1 = CAN Transmit Buffer 1 Interrupt is a high priority event
0 = CAN Transmit Buffer 1 Interrupt is a low priority event

bit **TXB0IP:** CAN Transmit Buffer 0 Interrupt Priority bit
1 = CAN Transmit Buffer 0 Interrupt is a high priority event
0 = CAN Transmit Buffer 0 Interrupt is a low priority event

bit **RXB1IP:** CAN Receive Buffer 1 Interrupt Priority bit
1 = CAN Receive Buffer 1 Interrupt is a high priority event
0 = CAN Receive Buffer 1 Interrupt is a low priority event

bit **RXB0IP:** CAN Receive Buffer 0 Interrupt Priority bit
1 = CAN Receive Buffer 0 Interrupt is a high priority event
0 = CAN Receive Buffer 0 Interrupt is a low priority event

bit **EEIP**: EE Write Complete Interrupt Priority bit
1 = EE Write Complete Interrupt is a high priority event
0 = EE Write Complete Interrupt is a low priority event

bit **CMIP**: Comparator Interrupt Priority bit
1 = Comparator Interrupt is a high priority event
0 = Comparator Interrupt is a low priority event

bit **BCLIP**: Bus Collision Interrupt Priority bit
1 = Bus Collision Interrupt is a high priority event
0 = Bus Collision Interrupt is a low priority event

bit **LVDIP**: Low-voltage Detect Interrupt Priority bit
1 = Low-voltage Detect Interrupt is a high priority event
0 = Low-voltage Detect Interrupt is a low priority event

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Note 1: The bit position of the priority bits is device dependent. Please refer to the device data sheet for bit placement.

10.2.5 RCON Register

The RCON register contains the bit that is used to enable prioritized interrupts (IPEN) as well as status bits to indicate the cause of a device reset, if the device was in sleep mode and if long writes to internal memory are enabled.

Register 10-7: RCON Register

R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
IPEN	LWRT	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

bit **IPEN:** Interrupt Priority Enable bit
1 = Enable priority levels (high and low) on interrupts
0 = Disable priority levels (all peripherals are high) on interrupts (PIC16CXXX compatibility) (This causes the Interrupt Priority (IP) bits to be ignored)

bit 6 **LWRT:** Long Write Enable
For details of bit operation see description of RCON register bit in Register 3-2

bit 5 **Unimplemented:** Read as '0'

bit 4 **$\overline{RI}$:** Reset Instruction Flag bit
For details of bit operation see description of RCON register bit in Register 3-2

bit 3 **$\overline{TO}$:** Watchdog Time-out Flag bit
For details of bit operation see description of RCON register bit in Register 3-2

bit 2 **$\overline{PD}$:** Power-down Detection Flag bit
For details of bit operation see description of RCON register bit in Register 3-2

bit 1 **$\overline{POR}$:** Power-on Reset Status bit
For details of bit operation see description of RCON register bit in Register 3-2

bit 0 **$\overline{BOR}$:** Brown-out Reset Status bit
For details of bit operation see description of RCON register bit in Register 3-2

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

10.3 Interrupt Handling Operation

The interrupts are controlled and monitored using several Special Function Registers. These may include the following register types:

- INTCON registers
- PIR registers
- PIE registers
- IPR registers

The PIR registers contain the interrupt flag bits, the PIE registers contain the enable bits and the IPR registers contain the priority bits. The number of PIR, PIE, and IPR registers depends on the number of interrupt sources on the device.

10.3.1 Interrupt Priority

Each interrupt can be assigned a priority level by clearing or setting the corresponding interrupt priority bit. The priority bits are located in the interrupt priority registers (IPR1, IPR2, IPR3, INTCON2 and INTCON3). A '1' in the priority register assigns high priority to the corresponding interrupt. A '0' in the register assigns low priority to the interrupt. All interrupt priority bits are reset to '1', meaning that all interrupts are assigned high priority at reset. The IPEN bit in the RCON register enables priority levels for interrupts. If clear, all priorities are set to high.

10.3.1.1 High Priority Interrupts

A global interrupt enable bit, GIE/GIEH (INTCON<7>) enables (if set) all un-masked interrupts or disables (if cleared) all interrupts. When bit GIE/GIEH is enabled and an interrupt's flag bit and enable bit are set while the priority is high, the interrupt will vector immediately. Individual interrupts can be disabled through their corresponding enable bits in various registers. Individual interrupt flag bits are set, regardless of the status of the GIE/GIEH bit. The GIE/GIEH bit is cleared on reset.

When a high priority interrupt is responded to, the GIE/GIEH bit is automatically cleared to disable any further interrupts, the return address is pushed onto the stack, and the PC is loaded with 000008h. Once in the interrupt service routine, the source of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared before re-enabling interrupts to avoid recursive interrupts. Most flag bits are required to be cleared by the application software. There are some flag bits that are automatically cleared by the hardware.

The "return from interrupt" instruction, RETFIE, exits the interrupt routine and sets the GIE/GIEH bit, which re-enables high priority interrupts.

10.3.1.2 Low Priority Interrupts

Low priority interrupts are defined by having a "0" in an interrupt priority register IPRx. To enable low priority interrupts, the IPEN bit must be set.

When the IPEN is set, the PEIE/GIEL bit (INTCON<6>) is no longer used to enable peripheral interrupts. Its new function is to globally enable and disable low priority interrupts only. When the service routine for a low priority interrupt is vectored to, the PEIE/GIEL bit is automatically cleared in hardware to disable any further low priority interrupts.

The return address is pushed onto the stack and the PC is loaded with 000018h instead of 000008h (all low priority interrupts will vector to 000018h). Once in the interrupt service routine, the source(s) of the low priority interrupt can be determined by polling the low priority interrupt flag bits. The interrupt flag bit(s) must be cleared before re-enabling interrupts to avoid recursive interrupts. Most flag bits are required to be cleared by the application software. There are some flag bits that are automatically cleared by the hardware. The RETFIE instruction will reset the PEIE/GIEL bit on return from low priority interrupts.

The GIE/GIEH bit's function has not changed in that it still enables/disables all interrupts, however, it is only cleared by hardware when servicing a high priority interrupt.

10.3.1.3 High Priority Interrupts Interrupting a Low Priority ISR

If a high priority interrupt flag and enable bits are set while servicing a low priority interrupt, the high priority interrupt will cause the low priority ISR to be interrupted (regardless of the state of the PEIE/GIEL bit), because it is used to disable/enable low priority interrupts only. The GIE/GIEH bit is cleared by hardware to disable any further high and low priority interrupts, the return address is pushed onto the stack, and the PC is loaded with 000008h (the high priority interrupt vector). Once in the interrupt service routine, the source of the high priority interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

Figure 10-4 shows a high priority interrupt interrupting a low priority ISR. Figure 10-5 shows a high priority FSR with a low priority interrupt pending.

Note: The GIEH bit, when cleared, will disable all interrupts regardless of priority.

Figure 10-4: Low Priority ISR Interrupted By High Priority Interrupt

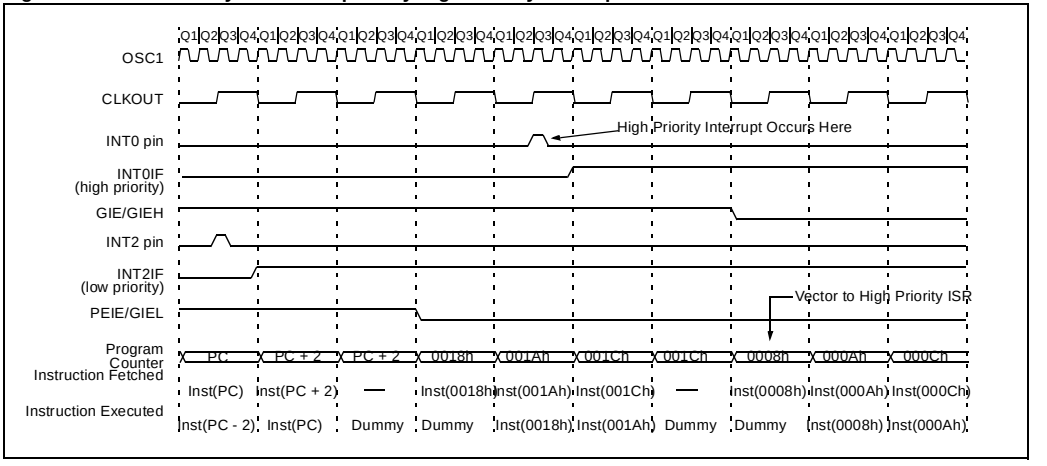


Figure 10-5: High Priority Interrupt With Pending Low Priority Interrupt

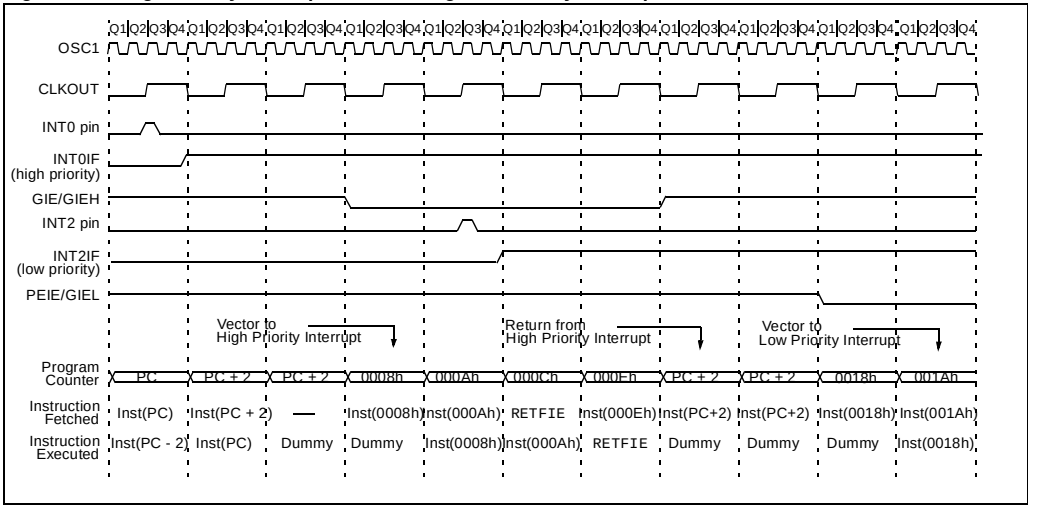


Figure 10-6 and Figure 10-7 show the two cases where a low priority interrupt has occurred and then a high priority interrupt occurs before the low priority ISR can begin execution. Figure 10-8 shows the first instruction of the low priority interrupt (at address 18h) beginning execution, when the high priority interrupt causes the program counter to be forced to the high priority interrupt vector address (08h).

Figure 10-6: Low Interrupt With High Interrupt Within 1 Cycle

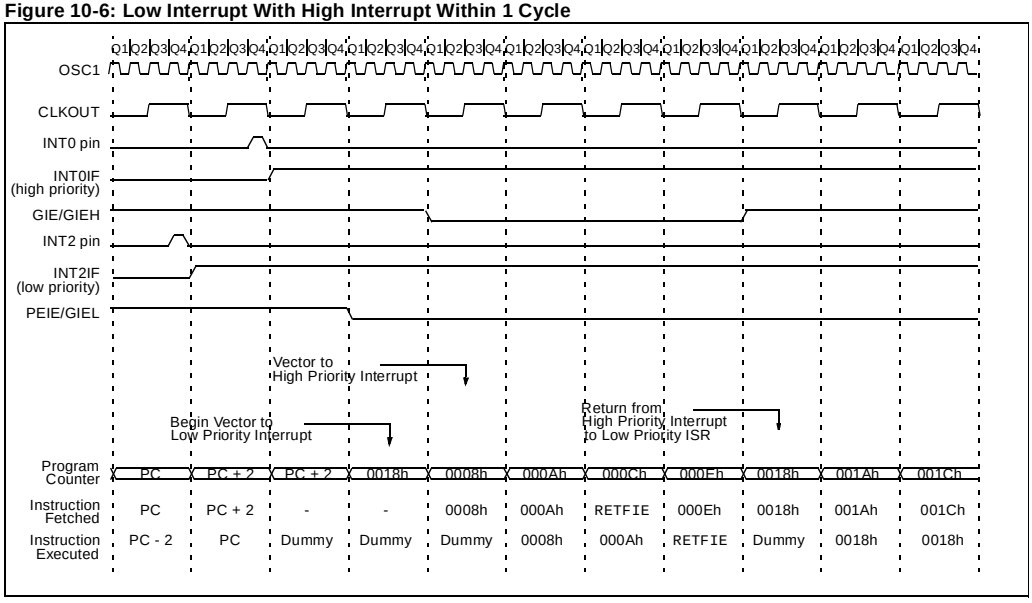


Figure 10-7: Low Interrupt With High Interrupt Within 2 Cycles

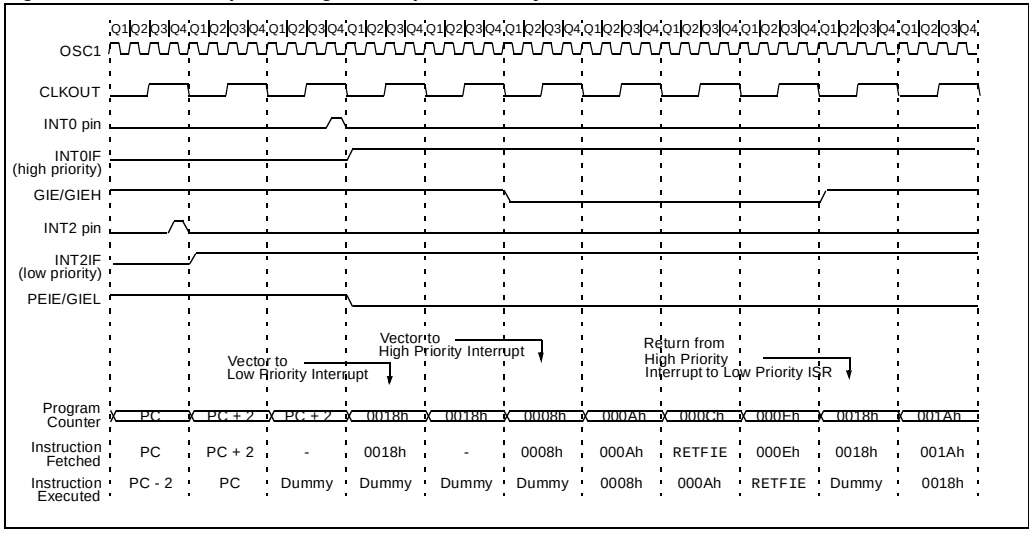
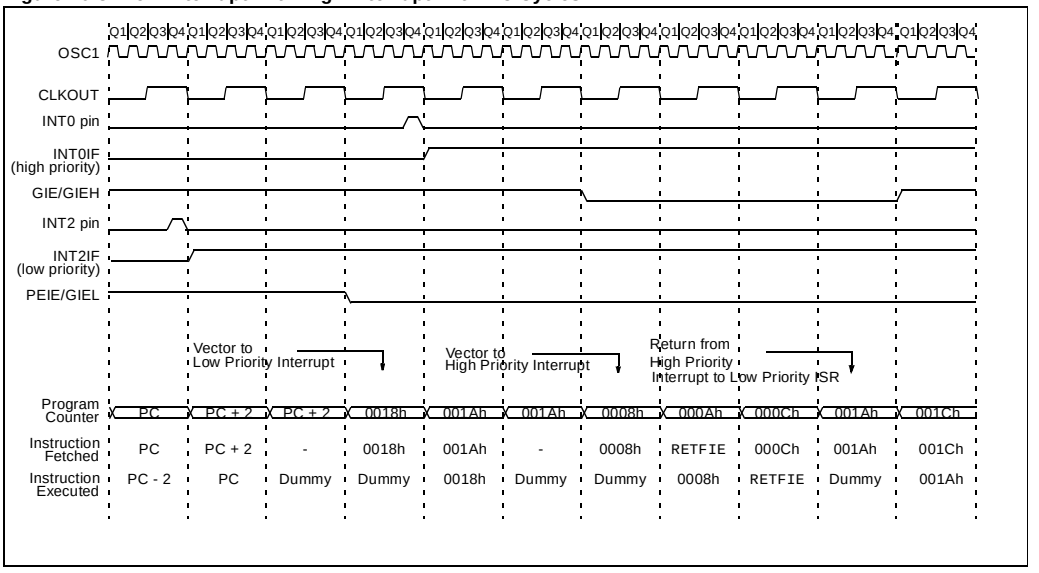


Figure 10-8: Low Interrupt With High Interrupt Within 3 Cycles



10.3.1.4 Low Priority Interrupts Interrupting a High Priority ISR

A low priority interrupt cannot interrupt a high priority ISR. The low priority interrupt will be served after all high priority interrupts have been served.

10.3.1.5 Simultaneous High and Low Priority Interrupts

If a high priority interrupt and a low priority interrupt are sampled at the same time, the high priority interrupt service routine is always serviced first. The GIE/GIEH bit is cleared by the hardware and the device vectors to location 000008h to the high priority ISR. After the interrupt is serviced, the corresponding interrupt flag should be cleared to avoid a recursive interrupt. The RETFIE instruction resets the GIE/GIEH bit, and if no other high priority interrupts are pending, the low priority interrupt is serviced.

10.3.1.6 Fast Context Saving During High Priority Interrupts

A "fast interrupt service" option is available for high priority interrupts. This is done by creating shadow registers for a few key registers (WREG, BSR and STATUS). Shadow registers are provided for the STATUS, WREG, and BSR registers and are only 1 deep. The shadow registers are not readable and are loaded with the current value of their corresponding register when the processor vectors for a high priority interrupt. The values in the shadow registers are then loaded back into the actual register if the fast return instruction (RETFIE 0x01) is used to return from the interrupt. An example for fast context saving is shown in Example 10-1.

Example 10-1: Fast Context Saving

```
        ORG 0x08
;
; Interrupt Service Routine (ISR) code. WREG, BSR and STATUS need
; to be saved upon entering the high priority interrupt service
routine
;
        RETFIE 0x01   ; WREG, BSR and STATUS will be restored
```

Note:	Fast interrupt saving cannot be used reliably if high and low priority interrupts are enabled. See Section **10.3.1.7**.

10.3.1.7 Context Saving During Low Priority Interrupts

Low priority interrupts may use the shadow registers. Any interrupt pushes values into the shadow registers. If both low and high priority interrupts are enabled, the shadow registers cannot be used reliably for low priority interrupts, as a high priority interrupt event will overwrite the shadow registers.

Users must save the key registers in software during a low priority interrupt.

For example:

a) Store the STATUS, WREG and BSR registers on a software stack.
b) Execute the ISR code.
c) Restore the STATUS, WREG and BSR registers from the software stack.

Example 10-2 shows example service routine code for when high and low priority interrupts are enabled.

Example 10-2: Interrupt Service Routine Template

```
            ORG 0x08                        ; high priority ISR
PUSH_REG_H MOVWF  WREG_TEMP_HIGH
            MOVFF  BSR, BSR_TEMP_HIGH
            MOVFF  STATUS, STATUS_TEMP_HIGH
;
; High Priority Interrupt Service Routine (ISR) Code goes here
;
POP_REG_H  MOVFF  BSR_TEMP_HIGH, BSR
            MOVF   WREG_TEMP_HIGH, W
            MOVFF  STATUS_TEMP_HIGH, STATUS
            RETFIE 0x00
;
PUSH_REG_L ORG 0x18                        ; Low Priority ISR
            MOVWF  WREG_TEMP_LOW
            MOVFF  BSR, BSR_TEMP_LOW
            MOVFF  STATUS, STATUS_TEMP_LOW
;
; Low Priority Interrupt Service Routine (ISR) code goes here
;
Pop_REG_L  MOVFF  BSR_TEMP_LOW, BSR
            MOVF   WREG_TEMP_LOW
            MOVFF  STATUS_TEMP_LOW, STATUS
            RETFIE 0x00
```

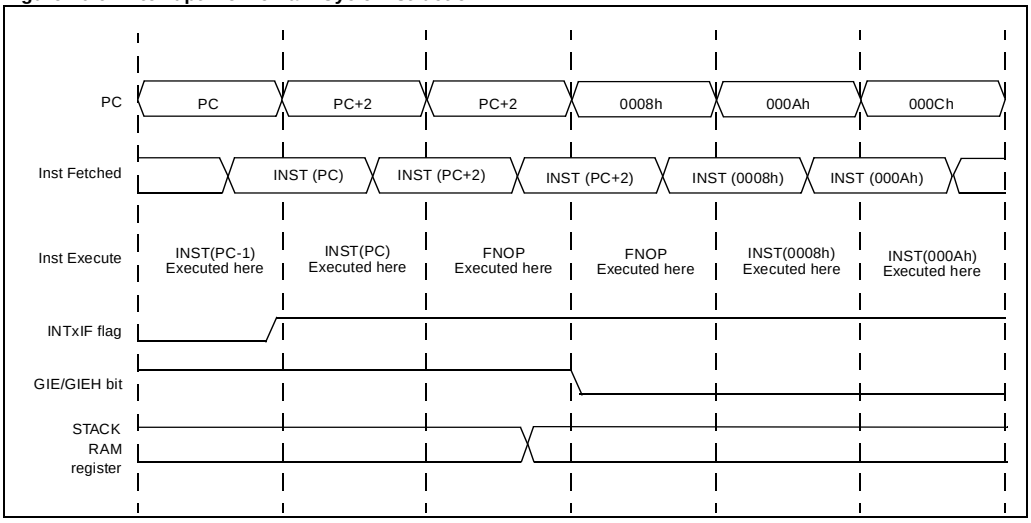
10.3.1.8 Interrupt Latency

For external interrupt events, such as the RB0/INT0 pin or PORTB change interrupt, the interrupt latency will be three or four instruction cycles. The exact latency depends when the interrupt event occurs. The interrupt latency is the same for one or two cycle instructions.

10.3.1.8.1 Interrupt Latency For One Cycle Instructions

Figure 10-9 shows the timing when an external interrupt is asserted during a one cycle instruction. The interrupt is sampled on Q4. The interrupt is then acknowledged on the Q2 cycle of the following instruction cycle when instruction PC is executed. This is followed by a forced NOP (dummy cycle) and the contents of the PC are stored on the stack during the Q3 cycle of this machine cycle. By the Q3/Q4 boundary of instruction cycle two, the interrupt vector is placed into the PC, and is presented on the program memory bus on the following cycle. This cycle is also a dummy cycle executing a forced NOP (FNOP) so that the CPU can fetch the first instruction from the interrupt service routine.

Figure 10-9: Interrupt Flow on a 1 Cycle Instruction

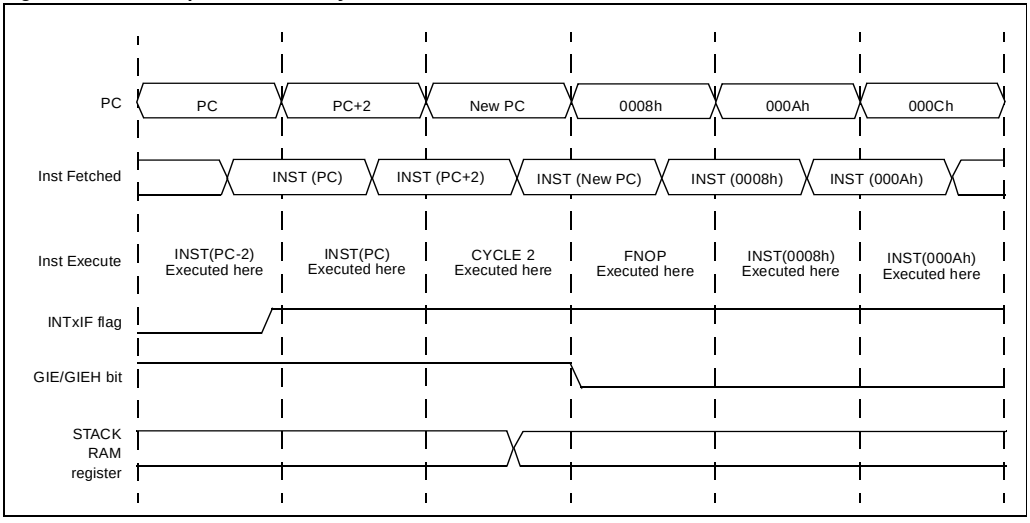


10.3.1.8.2 Interrupt Latency For Two Cycle Instructions

Figure 10-10 shows the timing when an external interrupt is asserted during a two cycle instruction. The interrupt is sampled on Q4. The interrupt is then acknowledged on the Q1 of the following instruction cycle when instruction PC is executed. This is followed by the second cycle of the instruction and the contents of the PC are stored on the stack during Q3 of this machine cycle. For all two cycle instructions, the PC may be updated with a new PC value due to execution control instructions like GOTO and CALL. The reason for the forced NOP (dummy cycle) is to maintain consistent interrupt latency between one and two cycle instructions. Two cycle instructions require this cycle for the update of the PC to a new PC value, because all two cycle instructions with the exception of MOVFF and MOVLF are execution control type instructions that update the PC with a new value (i.e. GOTO and CALL). The MOVFF and MOVLF instructions will increment the PC by 2 in this cycle because an operand fetch takes place in the second cycle. By Q3/Q4 the interrupt vector 000008h is placed into the PC and is presented on the program memory bus on the following cycle. This cycle is a dummy cycle executing a forced NOP (FNOP) so that the CPU can fetch the first instruction from the interrupt service routine.

Note:	When using the MOVFF instruction with any one of the PCL, TOSU, TOSH, and TOSL registers as destination, all interrupts have to be disabled.

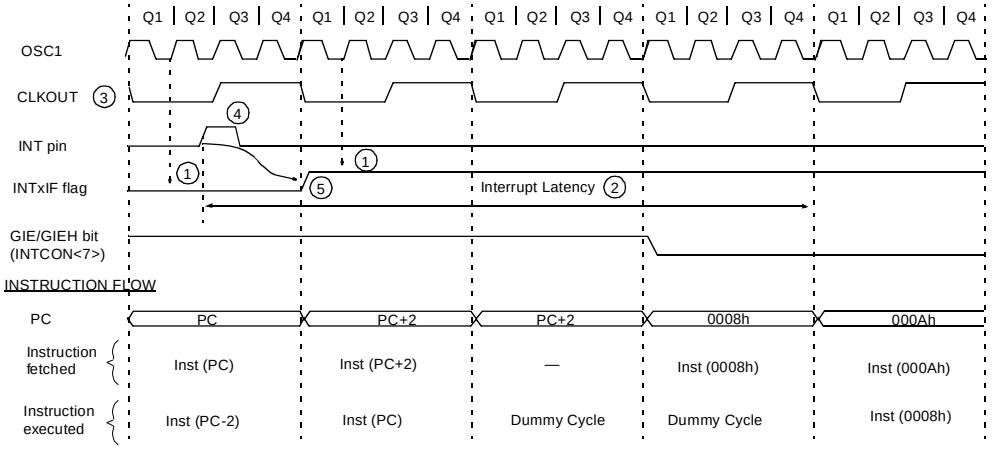
Figure 10-10:Interrupt Flow on a 2 Cycle or 2 Word Instruction



10.3.1.9 InTerrupts During Table Write Operations (Long Writes)

The long write is necessary for programming the internal EPROM. Instruction execution is halted while in a long write cycle. The long write will be terminated by any enabled interrupt. To ensure that the EPROM location has been well programmed, a minimum programming time is required. Typically, a Timer interrupt is used to time and terminate the long write. Having only one interrupt enabled to terminate the long write ensures that no unintended interrupts will prematurely terminate the long write.

Figure 10-11:INT0, INT1, and INT2 Pin Interrupt Timing (High Priority Shown)



Note 1: INTxIF flag is sampled here (every Q1).

Note 2: Interrupt latency = 3-4TCY where TCY = instruction cycle time.
Latency is the same whether Inst (PC) is a single cycle or a 2-cycle instruction.

Note 3: CLKOUT is available only in RC oscillator mode.

Note 4: For minimum width of INT pulse, refer to AC specs.

Note 5: INTxIF is enabled to be set anytime during the Q1 cycle.

10.4 Initialization

Example 10-3 enables high and low priority interrupts. The priority level for the peripherals is loaded into the IRP1 register (IRP1_VALUE) and the peripherals that are enabled depend on the value of PIE1_VALUE, which is loaded into the PIE1 register.

Example 10-3: Generic Initialization Example

```
    MOVLW   RCON_VALUE      ; RCON_VALUE = 1???????b
    MOVWF   RCON            ;
    MOVLW   IPR1_VALUE      ; Peripherals with high priority
                            ;   have a '1' in their bit
                            ;   position.
                            ;   Those with a low priority have
                            ;   a '0' in their bit position.
    MOVWF   IRP1            ;
    CLRF    PIR1            ; Clear all flag bits
    MOVLW   PIE1_VALUE      ; Enable desired peripheral
                            ;   interrupts by setting their
                            ;   bit position.
                            ; Disable others by clearing their
                            ;   bit position.
    MOVWF   PIE1            ;
    CLRF    INTCON3         ;
    CLRF    INTCON2         ;
    MOVLW   OxC0            ; Enable high and low global
                            ;   interrupts.
    MOVWF   INTCON          ;
```

10.5 Design Tips

Question 1: ***My code does not seem to execute properly.***

Answer 1:

There are many possible reasons. A couple of possibilities related to Interrupts are:

- Interrupts are not enabled, so the code cannot execute your expected ISR.
- The Interrupt may not be set to the priority level where your ISR code is located.

10.6 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is they may be written for the Base-Line, the Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the interrupts are:

Title	Application Note #
No related application notes at this time.	

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

10.7 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Interrupt description.

Section 11. I/O Ports

HIGHLIGHTS

This section of the manual contains the following major topics:

11.1 Introduction

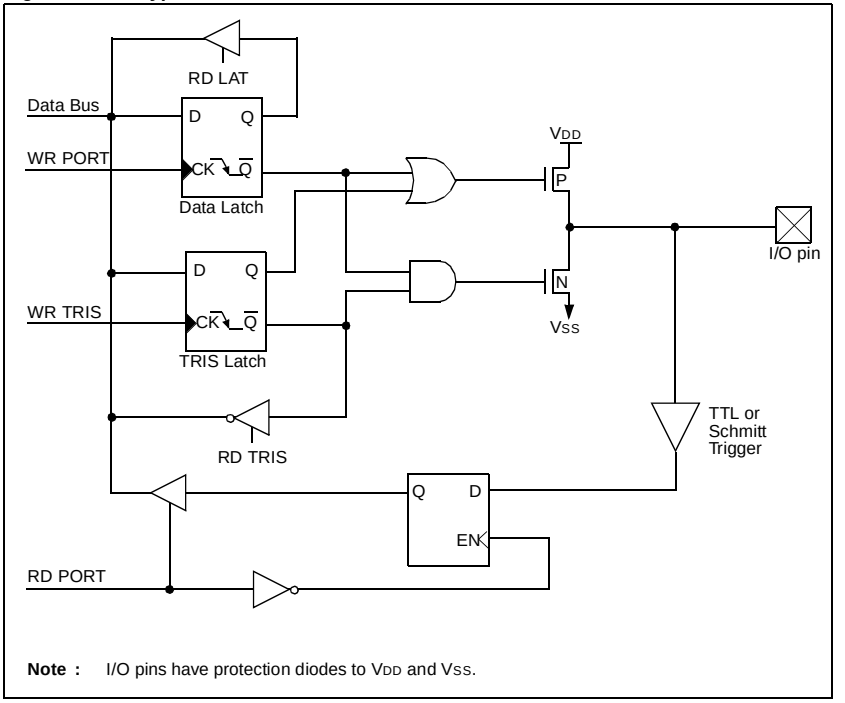
General purpose I/O pins can be considered the simplest of peripherals. They allow the PICmicro to monitor and control other devices. To add flexibility and functionality to a device, some pins are multiplexed with an alternate function(s). These functions depend on which peripheral features are on the device. In general, when a peripheral is functioning, that pin may not be used as a general purpose I/O pin.

For most ports, the I/O pin's direction (input or output) is controlled by the data direction register, called the TRIS register. TRIS<x> controls the direction of PORT<x>. A '1' in the TRIS bit corresponds to that pin being an input, while a '0' corresponds to that pin being an output. An easy way to remember is that a '1' looks like an I (input) and a '0' looks like an O (output).

The PORT register is the latch for the data to be output. When the PORT is read, the device reads the levels present on the I/O pins (not the latch). This means that care should be taken with read-modify-write commands on the ports and changing the direction of a pin from an input to an output.

Figure 11-1 shows a typical I/O port. This does not take into account peripheral functions that may be multiplexed onto the I/O pin. Reading the PORT register reads the status of the pins whereas writing to it will write to the port latch. All write operations (such as BSF and BCF instructions) are read-modify-write operations. Therefore, a write to a port implies that the port pins are read, this value is modified, and then written to the port data latch.

Figure 11-1: Typical I/O Port



Note : I/O pins have protection diodes to V_{DD} and V_{SS}.

When peripheral functions are multiplexed onto general I/O pins, the functionality of the I/O pins may change to accommodate the requirements of the peripheral module. An example of this is the Analog to Digital converter module which forces the I/O pin to the peripheral function when the device is reset. This prevents the device from consuming excess current if any analog levels were on the A/D pins after a reset occurred.

With some peripherals, the TRIS bit is overridden while the peripheral is enabled. Therefore, read-modify-write instructions (BSF, BCF, XORWF) with TRIS as destination should be avoided. The user should refer to the corresponding peripheral section for the correct TRIS bit settings.

PORT pins may be multiplexed with analog inputs and analog VREF inputs. The operation of each of these pins is selected, to be an analog input or digital I/O, by clearing/setting the control bits in other Special Function registers (SFRs). An example of this is the ADCON1 register for the 10-bit A/D module. Currently, when devices have pins selected as an analog input, these pins will read as '0's.

The TRIS registers control the direction of the port pins, even when they are being used as analog inputs. The user must ensure the TRIS bits are maintained set when using the pins as analog inputs.

Note 1: If pins are multiplexed with analog inputs, then on a Power-on Reset these pins are configured as analog inputs, as controlled by the ADCON1 register. Reading port pins configured as analog inputs read a '0'.

Note 2: If pins are multiplexed with comparator inputs, then on a Power-on Reset these pins are configured as analog inputs, as controlled by the CMCON register. Reading port pins configured as analog inputs read a '0'.

Note 3: Pins may be multiplexed with the Parallel Slave Port (PSP). For the PSP to function, the I/O pins must be configured as digital inputs and the PSPMODE bit must be set.

Note 4: At present, the Parallel Slave Port (PSP) is only multiplexed onto PORTD and PORTE. The PSP port becomes enabled when the PSPMODE bit is set. In this mode, the user must make sure that the TRISE bits are set (pins are configured as digital inputs) and that PORTE is configured for digital I/O. PORTD will override the values in the TRISD register. In this mode, the PORTD and PORTE input buffers are TTL. The control bits for the PSP operation are located in TRISE.

11.1.1 Multiplexed Peripherals

Pins may be configured as either digital inputs or digital outputs. Digital inputs are either TTL buffers or Schmitt Triggers. Outputs are CMOS drivers except for pin RA4, which is an open-drain output.

All pins also support one or more peripheral modules. When configured to operate with a peripheral, a pin may not be used for general input or output. In many cases, a pin must still be configured for input or output, although some peripherals override the TRIS configuration.

Peripherals supported include:

- Analog to Digital Converter Modules (A/D)
- Timer Modules
  - Timer0
  - Timer1
  - Timer2
  - Timer3
- Capture/Compare/Pulse Width Modulation (CCP) modules
- External Interrupts
- Interrupt On Change pins
- Parallel Slave Port (PSP) module
- In Circuit Serial Programming
- System Oscillator
- Weak Pull-Up sources
- Synchronous Serial Port (SSP) module
  - Serial Peripheral Interface (SPI)
  - I^2C
- Master Synchronous Serial Port (MSSP) module
  - Serial Peripheral Interface (SPI)
  - I^2C with full hardware Master mode support
- Addressable USART module
- Controller Area Network (CAN) module
- Comparator modules
- Voltage Reference modules
- Low Voltage Detect (LVD) module

11.1.2 Output Data Latches (LATx) and Data Direction Register (TRISx)

All port pins have an output data latch. Writing to a port writes to that latch (LATx). The data latch may also be read from and written to directly. If the pin is not being used by a peripheral, and is configured as an output by its TRIS bit, data in the latch will be output to the pin.

All port pins have corresponding Data Direction Register bits (TRISx register) which configure each pin as an input or output. Clearing a bit in a TRIS register (bit=0) configures the corresponding pin for output, and drives the contents of the output data latch (LATx) to the selected pin. Setting a TRIS register bit (bit=1) configures the corresponding pin as an input, and puts the corresponding output driver in a high impedance state. After a reset, all pins are configured as inputs.

Example 11-1 shows that writing the value in the WREG register to PORTB actually writes the value to the LATB register.

Example 11-1: Writing to PORTB actually writes to LATB

```
                        ;    LATB = 1100 0011
                        ; RB<7:0> = 1001 0011
                        ;   TRISB = 1111 0000 1=input 0=output
                        ;   W_REG = 0010 1110
       movwf   PORTB    ; writes W_REG to PORTB output data
                        ; latch (LATB)
                        ;    LATB = 0010 1110
                        ; RB<7:0> = 1001 1110 high nibble
                        ;                     no change (TRISB)
```

There are two input paths from a port. One path simply reads back what is in the data latch (LATx) without regard to whether or not the bits are being output, and may return values not present at the pin. The other path reads back the state of the pin (PORTx) unless a peripheral forces it to read back a fixed state.

Example 11-2 demonstrates the difference between reading a PORT and reading the output latch of the PORT.

Example 11-2: Reading PORTB compared to reading LATB

```
                        ; RB<7:0> = 1001 0101
                        ;    LATB = 0111 0101
                        ;   TRISB = 1111 0000    1=input 0=output
       movf    PORTB,W  ; reads states of PORTB pins
                        ;   W_REG = 1001 0101
       movf    LATB,W   ; reads contents of LATB data latch
                        ;   W_REG = 0111 0101
```

Reading the PORTx register reads the status of the pins whereas writing to it will write to the port data latch (LATx). A write to LATx can also be performed.

Example 11-3 shows the result of simply reading the PORT register. In this example, RB0 is being overdrive low and RB1 is being overdriven high. This is NOT recommended, and may actually violate device specifications, but is shown to give insight to the operation of an instruction which reads the I/O port with respect to the I/O ports data latch.

Example 11-3: Reading PORTB reads the state of RB7:RB0

```
                        ; RB<7:0> = 1001 0110
                        ;    LATB = 1100 0011
                        ;   TRISB = 1111 0000     1=input 0=output
       movf   PORTB,W   ; reads state of pins
                        ;   W_REG = 1001 0110
```

Example 11-4 shows what effects can occur when writing to the PORT registers.

Example 11-4: Writing to PORTB

```
                    ;   TRISB = 1111 0000    1=input 0=output
                    ;   W_REG = 1011 0110
                    ;    LATB = 1100 0011
                    ; RB<7:0> = 1001 0011
movwf   PORTB       ; writes W_REG to LATB
                    ;    LATB = 1011 0110
                    ; RB<7:0> = 1001 0110 low nibble only
                    ;                        is output
```

Example 11-5 shows what effects can occur when writing to the LAT registers.

Example 11-5: Writing to LATB

```
                    ;   TRISB = 1111 0000    1=input 0=output
                    ;   W_REG = 1011 0110
                    ;    LATB = 1100 0011
                    ; RB<7:0> = 1001 0011
movwf   LATB        ; writes W_REG to LATB
                    ;    LATB = 1011 0110
                    ; RB<7:0> = 1001 0110 same result as
                    ;                      'movwf PORTB'
```

Any instruction that performs a write operates internally as a read-modify-write operation. Caution must be used when these instructions are applied to a port where pins are switching between input and output states.

For example, a BSF PORTB, 5 instruction will cause all eight bits of PORTB to be read into the CPU. Then the instruction sets bit 5 and the resulting data is written to LATB.

If the RB7 pin is used for bi-directional I/O and is defined as an input when BSF PORTB, 5 executes, the input signal present on the pin itself would be read into the CPU and be written to LATB<7>, overwriting the previous contents. As long as the RB7 pin stays in the input mode, no problem occurs. However, if the RB7 pin is switched to an output, the contents of the data latch may be in an unintended state, causing the RB7 pin to be in the wrong state.

Example 11-6 shows how read-modify-write operations can affect the PORT register or the TRIS register.

Example 11-6: Read-modify-write of PORTB, and TRISB change toggles RB7

```
                    ;  RB<7:0> = 0001 0110
                    ;     LATB = 1001 0110
                    ;    TRISB = 1100 0000
bsf     PORTB,5     ; read-modify-write operation.
                    ;     LATB = 0011 0110 bit 7 cleared
                    ;  RB<7:0> = 1011 0110 RB7 changes to high speed
bcf     TRISB,7     ; changes RB7 from input to output
                    ;    TRISB = 0100 0000
                    ;  RB<7:0> = 0011 0110 RB7 in now driven low
```

A better solution would be to use the data latch instead. A BSF LATB, 5 instruction will read the bits in the output latch, set bit 5, and write the results back to the output latch. LATB<7> will never be at risk of being changed.

Example 11-7 shows that doing read-modify-writes on the LATx register and TRISx register may not cause the voltage level on the pin to change.

Example 11-7: Read-modify-write of LATB, and TRISB change has no effect on RB7

```
                    ;  RB<7:0> = 1001 0110
                    ;     LATB = 1001 0110 bit 7 is high
                    ;    TRISB = 1100 0000
bsf   LATB,5        ; read-modify-write operation
                    ;     LATB = 1011 0110 bit 7 has not changed
                    ;  RB<7:0> = 1011 0110
bcf   TRISB,7       ; changes RB7 from input to output
                    ;    TRISB = 0100 0000
                    ;  RB<7:0> = 1011 0110 RB7 remains high
```

11.2 PORTA, TRISA, and the LATA Register

PORTA is a 6-bit, or 7-bit latch depending upon the oscillator configuration selected by the FOSC configuration bits. The corresponding data direction register is TRISA, the data output latch is LATA, and the pins are PORTA. Except for RA4, all PORTA pins have TTL input buffers and full CMOS output drivers. All pins are configured as inputs on a reset.

The RA4 pin is a Schmitt Trigger input and an open drain output. All other RA port pins have TTL input levels and full CMOS output drivers. All pins have data direction bits (TRIS registers) which can configure these pins as output or input.

Setting a TRISA register bit puts the corresponding output driver in a hi-impedance mode. Clearing a bit in the TRISA register puts the contents of the output latch on the selected pin(s).

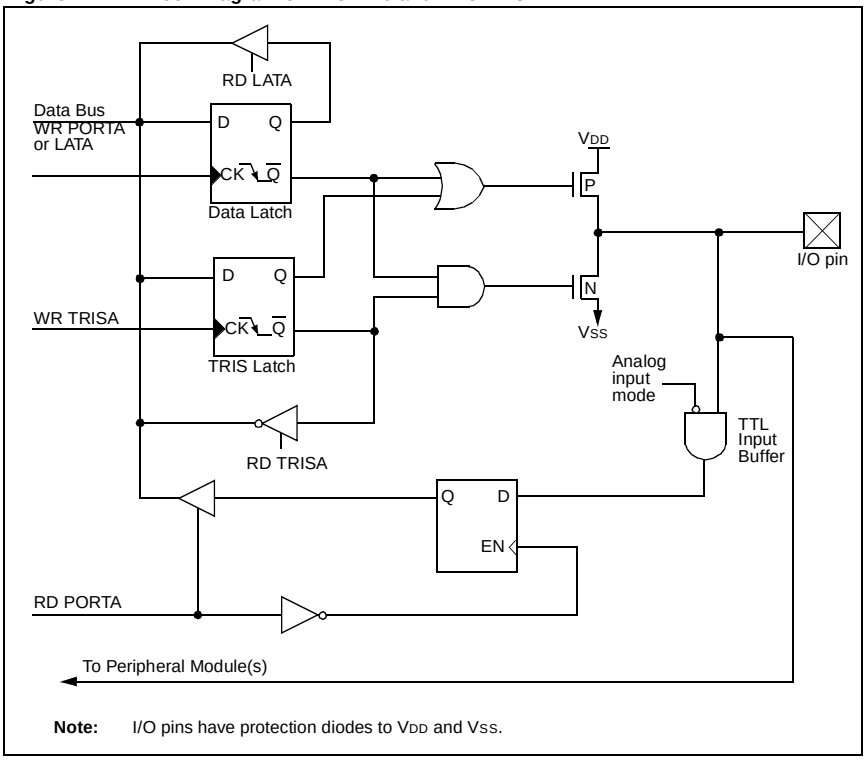
Example 11-8: Initializing PORTA

```
    CLRF   PORTA        ; Initialize PORTA by clearing output
                        ;    data latches
;   CLRF   LATA         ; Alternate method to initialize PORTA
    MOVLW  0xCF         ; Value used to initialize data direction
    MOVWF  TRISA        ; PORTA<3:0> = inputs PORTA<5:4> = outputs
                        ;    TRISA<7:6> always read as '0'
```

11.2.1 PORTA multi-plexed with Analog inputs

PORTA may be multiplexed with the AD module. When used as analog inputs, the TRISA must configure the corresponding pins as digital inputs (‘1’ on TRIS bit). On all resets, the PORTA pins are configured as analog inputs and a read of the digital inputs will result in read values of ‘0’.

Figure 11-2: Block Diagram of RA3:RA0 and RA5 Pins

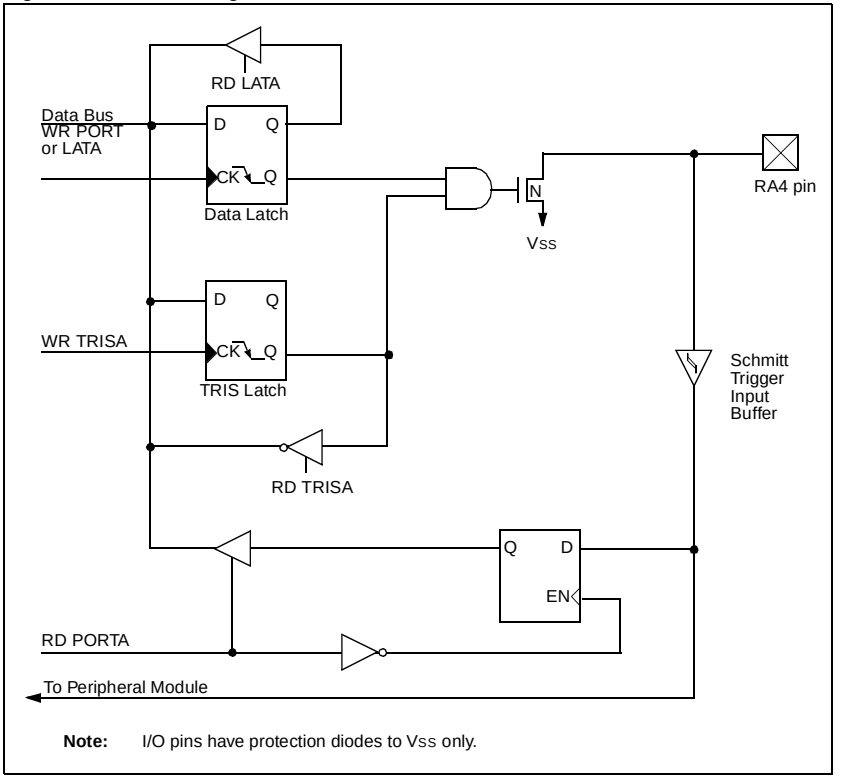


Note: I/O pins have protection diodes to VDD and VSS.

11.2.2 RA4 / Timer0 Clock Input

The RA4/T0CKI pin is a Schmitt Trigger input and an open drain output. Pin RA4 may be multiplexed with the peripheral module. All other PORTA pins have TTL input levels and CMOS output drivers.

Figure 11-3: Block Diagram of RA4 Pin



Note: I/O pins have protection diodes to Vss only.

Table 11-1: PORTA Functions

Name	Bit#	Buffer	Function
RA0	bit0	TTL	Input/output port pin
RA1	bit1	TTL	Input/output port pin
RA2	bit2	TTL	Input/output port pin
RA3	bit3	TTL	Input/output port pin
RA4/T0CKI	bit4	ST	Input/output port pin or external clock input for Timer0 Output is open drain type
RA5	bit5	TTL	Input/output port pin
RA6	bit6	TTL	Input/output port pin

Legend: TTL = TTL input, ST = Schmitt Trigger input.

Table 11-2: Summary of Registers Associated with PORTA

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISA	—	PORTA Data Direction Control Register							-111 1111	-111 1111
PORTA	—	Read PORTA pin/Write PORTA Data Latch							-00x 0000	-00u 0000
LATA	—	Read PORTA Data Latch/Write PORTA Data Latch							-xxx xxxx	-uuu uuuu
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'.
Shaded cells are not used by PORTA. ST = Schmitt Trigger input, TTL = TTL input.

11.3 PORTB, TRISB, and the LATB Register

PORTB is an 8-bit wide bidirectional port. The corresponding data direction register is TRISB, the data output latch is LATB, and the pins are PORTB. All pins have TTL inputs.

Setting a bit in the TRISB register puts the corresponding output driver in a hi-impedance input mode. Clearing a bit in the TRISB register puts the contents of the output latch on the selected pin. All pins are configured as inputs on a reset.

Four of the PORTB pins have a weak internal pull-up. Clearing the $\overline{\text{RBPU}}$ bit (INTCON2<7>) turns on pull-ups on all pins. The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a reset.

Example 11-9: Initializing PORTB

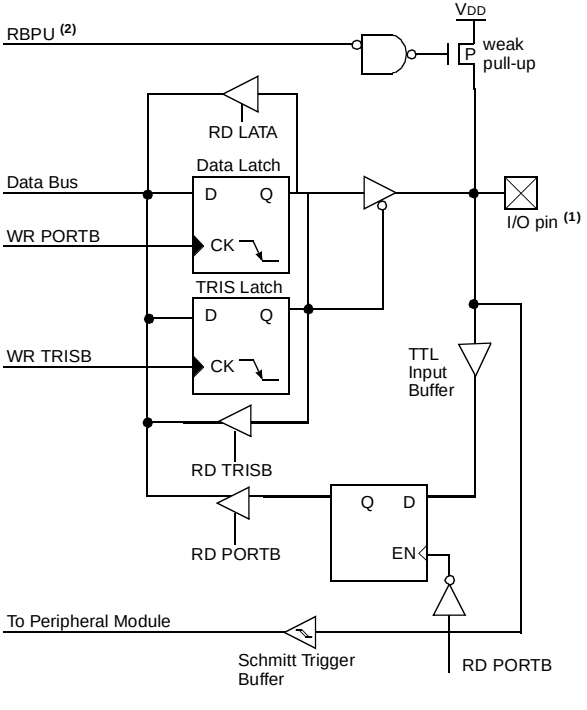
```
    CLRF  PORTS        ; Initialize PORTS by clearing output
                       ;     data latches
;   CLRF  LATB         ; Alternate method to initialize data latches
    MOVLW 0xCF         ; Value used to initialize data direction
    MOVWF TRISB        ; PORTB<3:0> = inputs, PORTB<5:4> = outputs
                       ;    PORTB<7:6> = inputs
```

11.3.1 RB2:RB0 / External Interrupts INT2:INT0

RB2:RB0 pins can also function as external interrupt sources INT2:INT0 while working as digital inputs. These interrupts are edge triggered on the edges selected by the bits. If enabled prior to entering sleep mode, these interrupts can wake the controller.

INT2:INT0 inputs have Schmitt trigger inputs, while the RB2:RB0 inputs have TTL buffer inputs.

Figure 11-4: Block Diagram of RB3:RB0 Pins



Note 1: I/O pins have diode protection to VDD and VSS.
Note : To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the $\overline{\text{RBPU}}$ bit.

Four of PORTB's pins, RB7:RB4, have an interrupt on change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt on change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB. The present inputs of RB7:RB4 and their previous values are XOR'ed together to detect a "mismatch" condition and set the RB Port change interrupt flag bit RBIF. When enabled, this flag will generate an interrupt that can wake the device from SLEEP.

This interrupt can wake the device from SLEEP. The user, in the interrupt service routine, can clear the interrupt in the following manner:

a) Any read or write of PORTB will end the mismatch condition, except a write using the MOVFF instruction.

b) Clear flag bit RBIF.

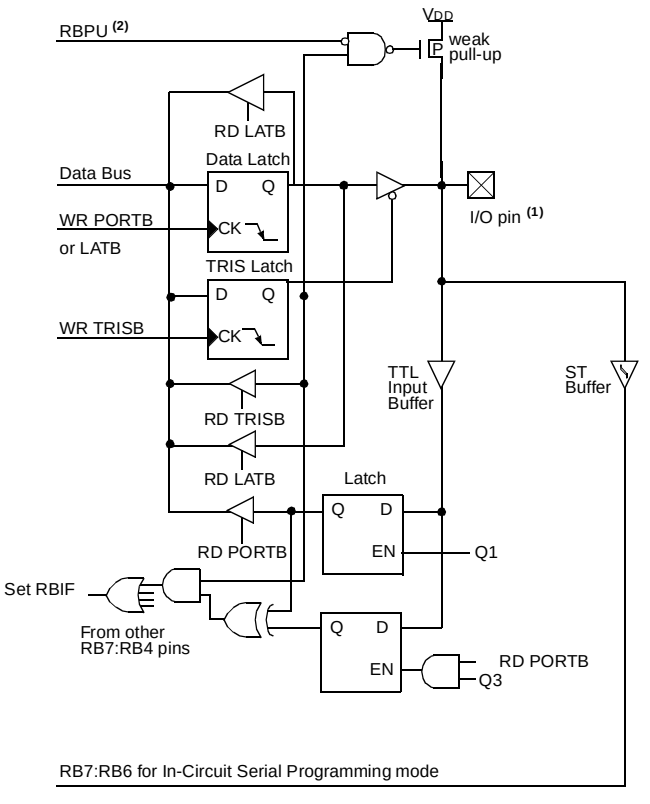
The MOVFF instruction will not end the mismatch condition if PORTB is used only as the destination register. The contents of the destination register are not automatically read by this instruction in the second cycle. All other reads, writes, and bit operations will read the port during execution.

A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition, and allow flag bit RBIF to be cleared.

This interrupt on change (i.e., mismatch) feature, together with software configurable pull-ups on these four pins allow easy interface to a keypad and make it possible for wake-up on key-depression.

The interrupt on change feature is recommended for wake-up on key depression and operations where PORTB is only used for the interrupt on change feature. Polling of PORTB is not recommended while using the interrupt on change feature.

Figure 11-5: Block Diagram of RB7:RB4 Pins



Note 1: I/O pins have diode protection to VDD and VSS.
Note 2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the $\overline{\text{RBPU}}$ bit.

11.3.2 RB7:RB6 - In Circuit Serial Programming

If ICSP is implemented in the target application, some means of isolating RB7:RB6 from the rest of the circuit should be provided. The ISCP inputs have Schmitt Triggers while the RB7:RB6 inputs have TTL inputs.

Table 11-3: PORTB Functions

Name	Bit#	Buffer	Function
RB0/INT0	bit0	TTL/ST (1)	Input/output port pin or external interrupt0 input. Internal software programmable weak pull-up.
RB1/INT1	bit1	TTL/ST (1)	Input/output port pin or external interrupt1 input. Internal software programmable weak pull-up.
RB2/INT2	bit2	TTL/ST (1)	Input/output port pin or external interrupt2 input. Internal software programmable weak pull-up.
RB3/CCP2 (3)	bit3	TTL/ST (4)	Input/output port pin or Capture2 input/Compare2 output/PWM2 output if CCP2MX is enabled in the configuration register. Internal software programmable weak pull-up.
RB4	bit4	TTL	Input/output port pin (with interrupt on change). Internal software programmable weak pull-up.
RB5	bit5	TTL	Input/output port pin (with interrupt on change). Internal software programmable weak pull-up.
RB6	bit6	TTL/ST (2)	Input/output port pin (with interrupt on change). Internal software programmable weak pull-up. Serial programming (CLOCK).
RB7	bit7	TTL/ST (2)	Input/output port pin (with interrupt on change). Internal software programmable weak pull-up. Serial programming (DATA).

Legend: TTL = TTL input, ST = Schmitt Trigger input.

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
Note 2: This buffer is a Schmitt Trigger input when used in serial programming mode.
Note 3: The CCP2 input is only multiplexed on the RB3 pin if the CCP2MX configuration bit is '0'.
Note 4: The CCP2 input is a Schmitt Trigger if the CCP2MX configuration bit is '0'.

Table 11-4: Summary of Registers Associated with PORTB

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISB	PORTB Data Direction Register								1111 1111	1111 1111
PORTB	Read PORTB pins/Write PORTB Data Latch								xxxx xxxx	uuuu uuuu
LATB	Read PORTB Data Latch/Write PORTB Data Latch								xxxx xxxx	uuuu uuuu
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
INTCON2	$\overline{\text{RBPU}}$	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	1111 -1-1	1111 -1-1
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF	11-0 0-00	11-0 0-00

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'.
Shaded cells are not used by PORTB.

11.4 PORTC, TRISC, and the LATC Register

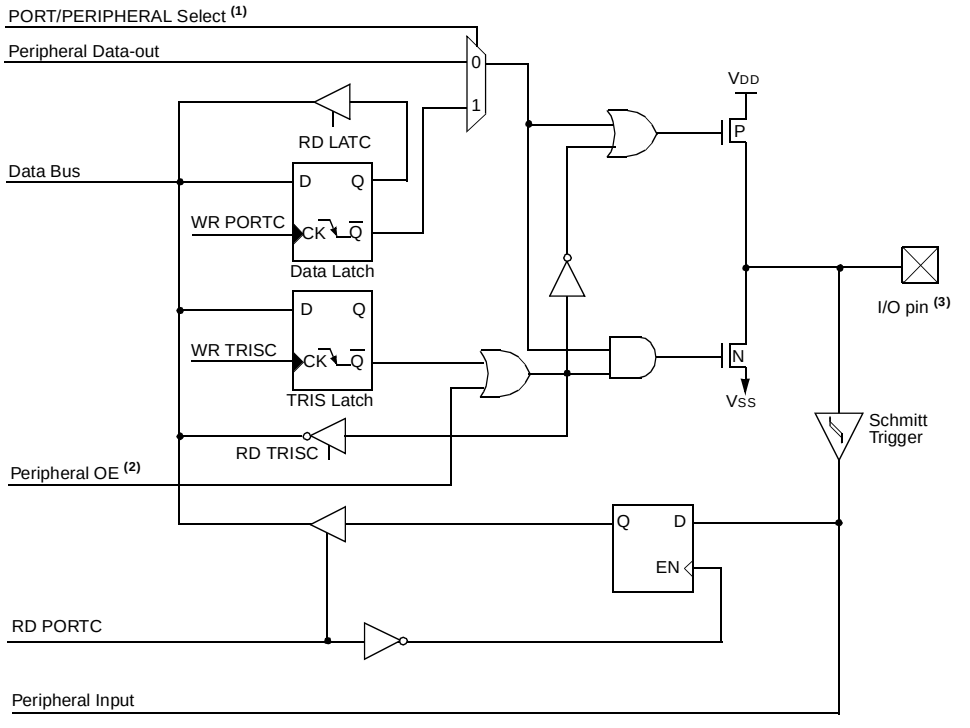
PORTC is an 8-bit bi-directional port. Each pin is individually configurable as an input or output through the TRISC register. The data output latch is LATC. PORTC pins have Schmitt Trigger input buffers.

When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRIS bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input, and other peripherals may not override the TRIS bits (requires that TRIS bits are configured for proper peripheral operation). The user should refer to the corresponding peripheral section for the correct TRIS bit settings.

Example 11-10: Initializing PORTC

```
    CLRF   PORTC    ; Initialize PORTC
                    ;   by clearing output data latches
;   CLRF   LATC     ; Alternate method
                    ;   to clear output latch
    MOVLW  0xCF     ; Value used to initialize data direction
    MOVWF  TRISC    ; PORTC<3:0> = inputs,
                    ;   PORTC<5:4> = outputs,
                    ;   PORTC<7:6> = inputs
```

Figure 11-6: PORTC Block Diagram (Peripheral Output Override)



Note 1: Port/Peripheral select signal selects between port data and peripheral output. Peripheral OE (output enable) is only activated if peripheral select is active.
I/O pins have diode protection to VDD and VSS.

All PORTC pins have Schmitt Trigger input buffers. When a peripheral uses a pin for output, the peripheral will override the TRIS and force the pin to be an output. Conversely, when a peripheral uses a pin for input, the peripheral will override the TRIS and force the pin to be an input. The TRIS is ignored while the peripheral controls the pin.

A read from the TRISC register bits will always yield the value contained in the TRISC latch whether or not a peripheral TRIS override is being asserted. This will allow a user to read the status of the TRISC bits at all times.

Since the TRIS bit override is in effect when the peripheral is enabled, read-modify-write instructions (BSF, BCF and others) with TRIS as destination should be used with care.

These instructions will have no effect on the current state of the pin. However, prior to disabling the peripheral and returning the pin to general use, the user should ensure that the TRIS bit is correctly set for that pin.

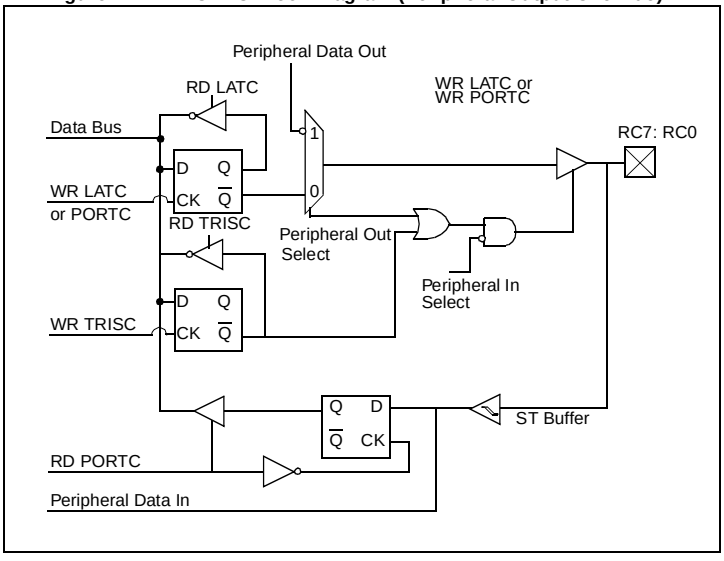
When a peripheral uses a pin for output, the peripheral will override the TRIS and force the pin to be an output. Conversely, when a peripheral uses a pin for input, the peripheral will override the TRIS and force the pin to be an input. The TRIS is ignored while the peripheral controls the pin.

A read from the TRISC register bits will always yield the value contained in the TRISC latch whether or not a peripheral TRIS override is being asserted. This will allow a user to read the status of the TRISC bits at all times.

Since the TRIS bit override is in effect when the peripheral is enabled, read-modify-write instructions (BSF, BCF, and others) with TRIS as destination should be used with care.

These instructions will have no effect on the current state of the pin. However, prior to disabling the peripheral and returning the pin to general use, the user should ensure that the TRIS bit is correctly set for that pin.

Figure 11-7: PORTC Block Diagram (Peripheral Output Override)



11.4.1 RC1 / CCP2 Input / Output

The RC1 pin can be multiplexed with the CCP2 module input/output. To achieve this, the CCP2MX Configuration bit must be programmed to a '1'.

Table 11-5: PORTC Functions

Name	Bit#	Buffer Type	Function
RC0	bit0	ST	Input/output port pin or Timer1 oscillator output or Timer1/Timer3 clock input
RC1	bit1	ST	Input/output port pin or Timer1 oscillator input
RC2	bit2	ST	Input/output port pin or Capture1 input/Compare1 output/PWM1 output
RC3	bit3	ST	Input/output port pin
RC4	bit4	ST	Input/output port pin
RC5	bit5	ST	Input/output port pin
RC6	bit6	ST	Input/output port pin
RC7	bit7	ST	Input/output port pin

Legend: ST = Schmitt Trigger input.

Table 11-6: Summary of Registers Associated with PORTC

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISC	PORTC Data Direction Control Register								1111 1111	1111 1111
PORTC	Read PORTC pin/Write PORTC Data Latch (LATC)								xxxx xxxx	uuuu uuuu
LATC	LATC Data Output Register								xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged.

11.5 PORTD, LATD, and the TRISD Register

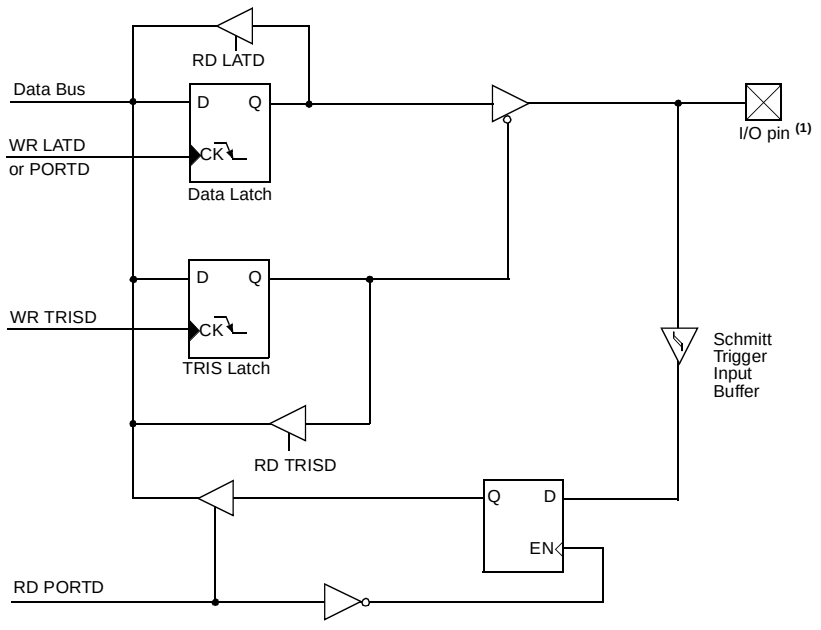
PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configured as an input or output.

All PORTD pins have latch bits (LATD register). The LATD register, when read, will yield the contents of the PORTD latch, and when written, will modify the contents of the PORTD latch. This modifies the value driven out on a pin if the corresponding TRISD bit is configured for output. This can be used in read-modify-write instructions that allow the user to modify the contents of the latch register regardless of the status of the corresponding pins.

Example 11-11: Initializing PORTD

```
    CLRF   PORTD    ; Initialize PORTD
                    ;   by clearing output data latches
;   CLRF   LATD     ; Alternate method to initialize
                    ;   data output latch
    MOVLW  0xCF     ; Value used to initialize data direction
    MOVWF  TRISD    ; PORTD<3:0> = inputs,
                    ;   PORTD<5:4> = outputs,
                    ;   PORTD<7:6> = inputs
```

Figure 11-8: Typical PORTD Block Diagram (in I/O Port Mode)



Note 1: I/O pins have protection diodes to VDD and VSS.

Table 11-7: PORTD Functions

Name	Bit#	Buffer Type	Function
RD0	bit0	ST	Input/output port pin
RD1	bit1	ST	Input/output port pin
RD2	bit2	ST	Input/output port pin
RD3	bit3	ST	Input/output port pin
RD4	bit4	ST	Input/output port pin
RD5	bit5	ST	Input/output port pin
RD6	bit6	ST	Input/output port pin
RD7	bit7	ST	Input/output port pin

Legend: ST = Schmitt Trigger input, TTL = TTL input.

Table 11-8: Summary of Registers Associated with PORTD

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISD	PORTD Data Direction Control Register								1111 1111	1111 1111
PORTD	Read PORTD pin / Write PORTD Data Latch								xxxx xxxx	uuuu uuuu
LATD	Read PORTD Data Latch/Write PORTD Data Latch								xxxx xxxx	uuuu uuuu
PSPCON (1)	IBF	OBF	IBOV	PSPMODE	—	—	—	—	0000 xxxx	0000 uuuu

Legend: x = unknown, u = unchanged.

Note 1: In some devices, the four bits in the PSPCON register may be located in the upper four bits of the TRISE register.

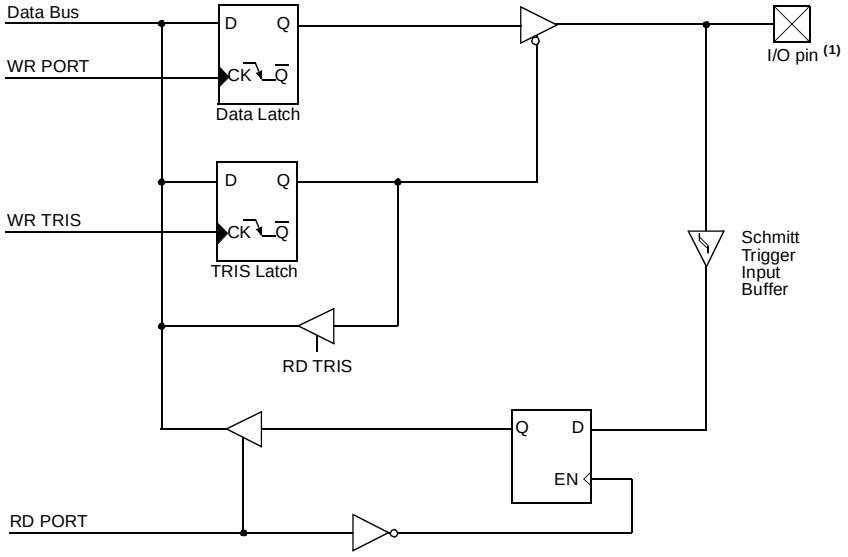
11.6 PORTE, TRISE, and the LATE Register

PORTE can be up to an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.

Example 11-12: Initializing PORTE

```
   CLRF   PORTE    ; Initialize PORTE by clearing output
                   ;   data latches
;  CLRF   LATE     ; Alternate method to initialize
                   ;   data output latch
   MOVLW  0x03     ; Value used to initialize data direction
   MOVWF  TRISE    ; PORTE<1:0> = inputs,
                   ;   PORTE<7:2> = outputs
```

Figure 11-9: Typical PORTE Block Diagram (in I/O Port Mode)



Note 1: I/O pins have protection diodes to VDD and VSS.

Note: On some devices with PORTE, the upper bits of the TRISE register are used for the Parallel Slave Port control and status bits.

Table 11-9: PORTE Functions

Name	Bit#	Buffer Type	Function
RE0	bit0	ST	Input/output port pin
RE1	bit1	ST	Input/output port pin
RE2	bit2	ST	Input/output port pin

Legend: ST = Schmitt Trigger input, TTL = TTL input.

Table 11-10: Summary of Registers Associated with PORTE

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISE	IBF	OBF	IBOV	PSP-MODE	—	PORTE Data Direction Bits			0000 -111	0000 -111
PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -000	---- -000
LATE	—	—	—	—	—	LATE Data Output Register			---- -xxx	---- -uuu
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	--0- 0000	--0- 0000
PSPCON (1)	IBF	OBF	IBOV	PSP-MODE	—	—	—	—	0000 xxxx	0000 uuuu

Legend: x = unknown, u = unchanged.

Note 1: In some devices, the four bits in the PSPCON register may be located in the upper four bits of the TRISE register.

11.7 PORTF, LATF, and the TRISF Register

PORTF is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configured as an input or output.

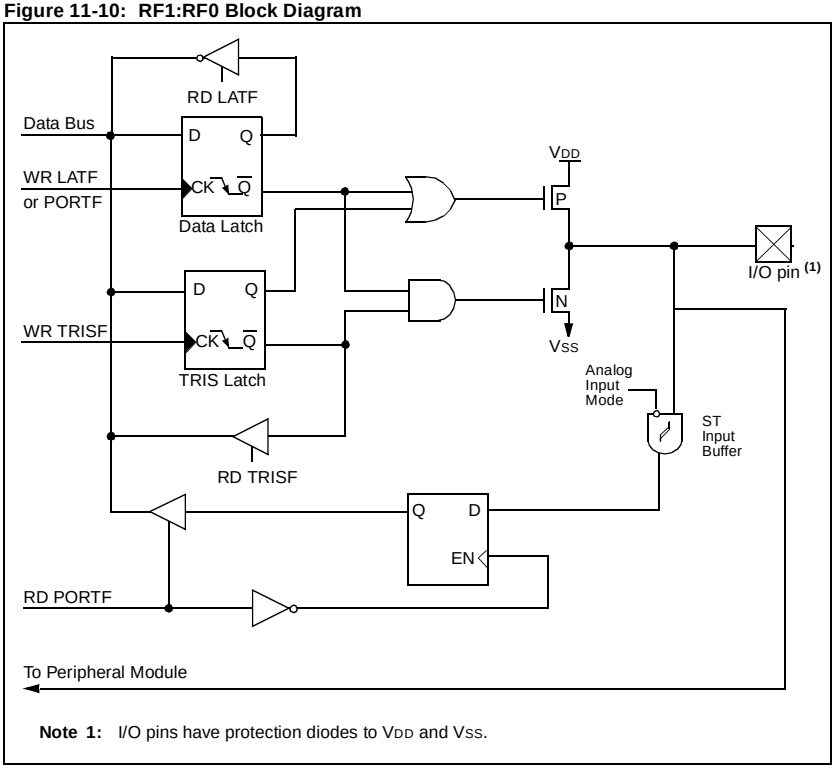
All PORTF pins have latch bits (LATF register). The LATF register, when read, will yield the contents of the PORTF latch, and when written, will modify the contents of the PORTF latch. This modifies the value driven out on a pin if the corresponding TRISF bit is configured for output. This can be used in read-modify-write instructions that allow the user to modify the contents of the latch register regardless of the status of the corresponding pins.

PORTF pins are multiplexed with analog inputs, system bus address bits, chip enables, and the $\overline{UB}$ and $\overline{LB}$ external bus control signals. The operation of each analog pin is selected by clearing/setting the control bits in the ADCON0 and ADCON1 register.

The TRISF register controls the direction of the RF pins, even when they are being used as analog inputs. The user must ensure the bits in the TRISF register are maintained set when using them as analog inputs.

Note:	On all forms of Reset, the RF2:RF0 are configured as analog inputs and read as '0'.

Figure 11-10: RF1:RF0 Block Diagram



Note 1: I/O pins have protection diodes to VDD and VSS.

Table 11-11: PORTF Functions

Name	Bit#	Buffer Type	Function
RF0	bit0	ST	Input/output port pin
RF1	bit1	ST	Input/output port pin
RF2	bit2	ST	Input/output port pin
RF3	bit3	ST	Input/output port pin
RF4	bit4	ST	Input/output port pin
RF5	bit5	ST	Input/output port pin
RF6	bit6	ST	Input/output port pin
RF7	bit7	ST	Input/output port pin

Legend: ST = Schmitt Trigger input.

Table 11-12: Summary of Registers Associated with PORTF

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISF	PORTF Data Direction Control Register								1111 1111	1111 1111
PORTF	Read PORTF pin / Write PORTF Data Latch								xxxx xx00	uuuu u000
LATF	Read PORTF Data Latch/Write PORTF Data Latch								0000 0000	uuuu u000

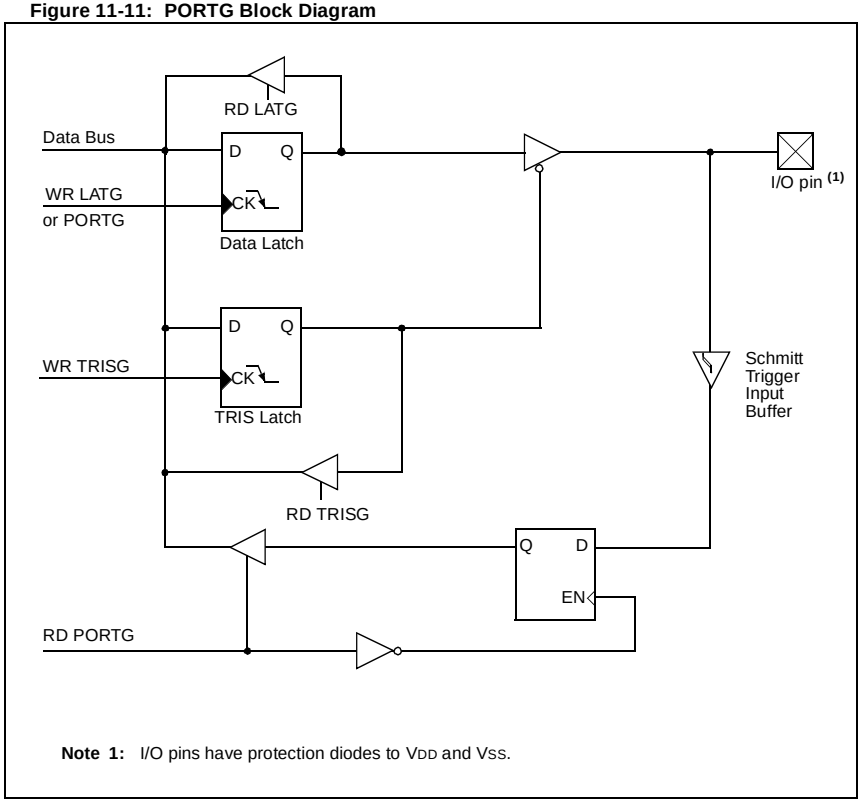
Legend: x = unknown, u = unchanged. Shaded cells are not used by Port F.

11.8 PORTG, LATG, and the TRISG Register

PORTG is a 5-bit port with Schmitt Trigger input buffers. Each pin is individually configured as an input or output.

All PORTG pins have latch bits (LATG register). The LATG register, when read, will yield the contents of the PORTG latch, and when written, will modify the contents of the PORTG latch. This modifies the value driven out on a pin if the corresponding TRISG bit is configured for output. This can be used in read-modify-write instructions that allow the user to modify the contents of the latch register regardless of the status of the corresponding pins.

Figure 11-11: PORTG Block Diagram



Note 1: I/O pins have protection diodes to VDD and VSS.

Table 11-13: PORTG Functions

Name	Bit#	Buffer Type	Function
RG0	bit0	ST	Input/output port pin
RG1	bit1	ST	Input/output port pin
RG2	bit2	ST	Input/output port pin
RG3	bit3	ST	Input/output port pin
RG4	bit4	ST	Input/output port pin
RG5	bit5	ST	Input/output port pin
RG6	bit6	ST	Input/output port pin
RG7	bit7	ST	Input/output port pin

Legend: ST = Schmitt Trigger input.

Table 11-14: Summary of Registers Associated with PORTG

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISG	PORTG Data Direction Control Register								---1 1111	---1 1111
PORTG	Read PORTG pin / Write PORTG Data Latch								---x xxxx	---u uuuu
LATG	Read PORTG Data Latch/Write PORTG Data Latch								---x xxxx	---u uuuu

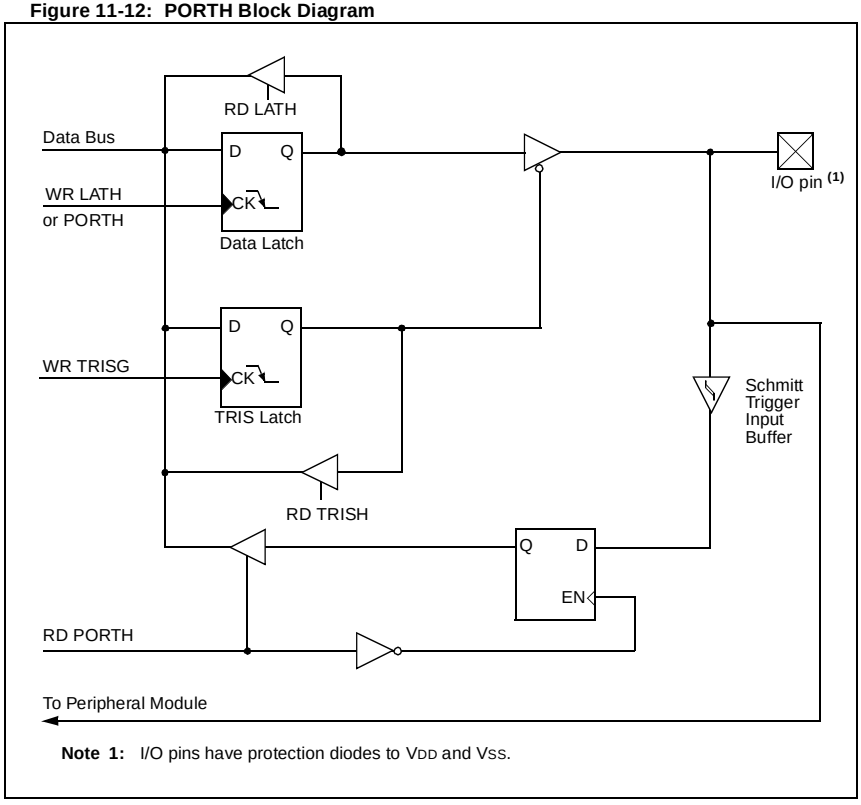
Legend: x = unknown, u = unchanged.

11.9 PORTH, LATH, and the TRISH Register

PORTH is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configured as an input or output.

All PORTH pins have latch bits (LATH register). The LATH register, when read, will yield the contents of the PORTH latch, and when written, will modify the contents of the PORTH latch. This modifies the value driven out on a pin if the corresponding TRISH bit is configured for output. This can be used in read-modify-write instructions that allow the user to modify the contents of the latch register regardless of the status of the corresponding pins.

Figure 11-12: PORTH Block Diagram



Note 1: I/O pins have protection diodes to VDD and VSS.

Table 11-15: PORTH Functions

Name	Bit#	Buffer Type	Function
RH0	bit0	ST	Input/output port pin
RH1	bit1	ST	Input/output port pin
RH2	bit2	ST	Input/output port pin
RH3	bit3	ST	Input/output port pin
RH4	bit4	ST	Input/output port pin
RH5	bit5	ST	Input/output port pin
RH6	bit6	ST	Input/output port pin
RH7	bit7	ST	Input/output port pin

Legend: TTL = TTL input.

Table 11-16: Summary of Registers Associated with PORTH

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISH	PORTH Data Direction Control Register								1111 1111	1111 1111
PORTH	Read PORTH pin / Write PORTH Data Latch								0000 xxxx	0000 uuuu
LATH	Read PORTH Data Latch/Write PORTH Data Latch								xxxx xxxx	uuuu uuuu

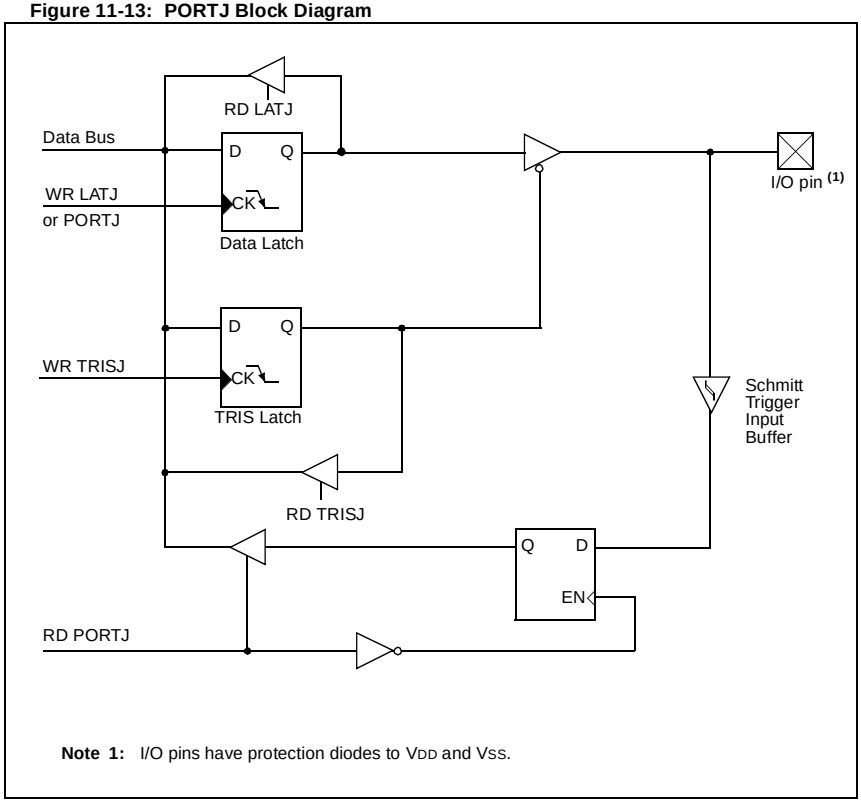
Legend: x = unknown, u = unchanged, - = unimplemented. Shaded cells are not used by Port H.

11.10 PORTJ, LATJ, and the TRISJ Register

PORTJ is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configured as an input or output.

All PORTJ pins have latch bits (LATJ register). The LATJ register, when read, will yield the contents of the PORTJ latch, and when written, will modify the contents of the PORTJ latch. This modifies the value driven out on a pin if the corresponding TRISJ bit is configured for output. This can be used in read-modify-write instructions that allow the user to modify the contents of the latch register regardless of the status of the corresponding pins.

Figure 11-13: PORTJ Block Diagram



Note 1: I/O pins have protection diodes to VDD and VSS.

Table 11-17: PORTJ Functions

Name	Bit#	Buffer Type	Function
RJ0	bit0	ST	Input/output port pin
RJ1	bit1	ST	Input/output port pin
RJ2	bit2	ST	Input/output port pin
RJ3	bit3	ST	Input/output port pin
RJ4	bit4	ST	Input/output port pin
RJ5	bit5	ST	Input/output port pin
RJ6	bit6	ST	Input/output port pin
RJ7	bit7	ST	Input/output port pin

Legend: ST = Schmitt Trigger input.

Table 11-18: Summary of Registers Associated with PORTJ

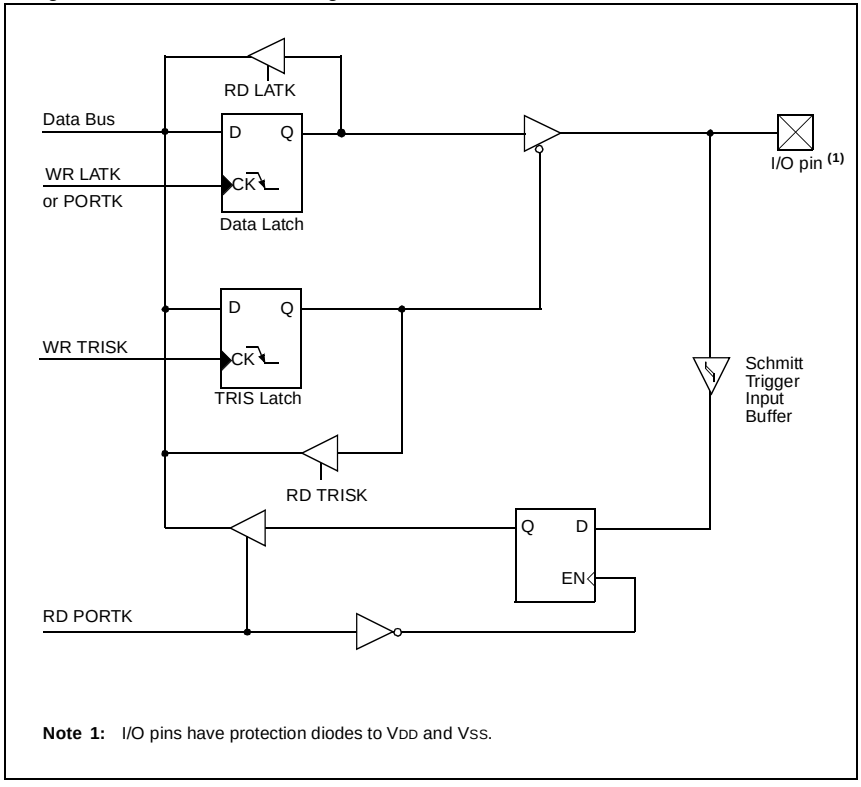
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISJ	PORTJ Data Direction Control Register								1111 1111	1111 1111
PORTJ	Read PORTJ pin / Write PORTJ Data Latch								xxxx xxxx	uuuu uuuu
LATJ	Read PORTJ Data Latch/Write PORTJ Data Latch								xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged.

11.11 PORTK, LATK, and the TRISK Register

PORTK is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configured as an input or output.

Figure 11-14: PORTK Block Diagram



Note 1: I/O pins have protection diodes to VDD and VSS.

Table 11-19: PORTK Functions

Name	Bit#	Buffer Type	Function
RK0	bit0	ST	Input/output port pin
RK1	bit1	ST	Input/output port pin
RK2	bit2	ST	Input/output port pin
RK3	bit3	ST	Input/output port pin
RK4	bit4	ST	Input/output port pin
RK5	bit5	ST	Input/output port pin
RK6	bit6	ST	Input/output port pin
RK7	bit7	ST	Input/output port pin

Legend: ST = Schmitt Trigger input.

Table 11-20: Summary of Registers Associated with PORTK

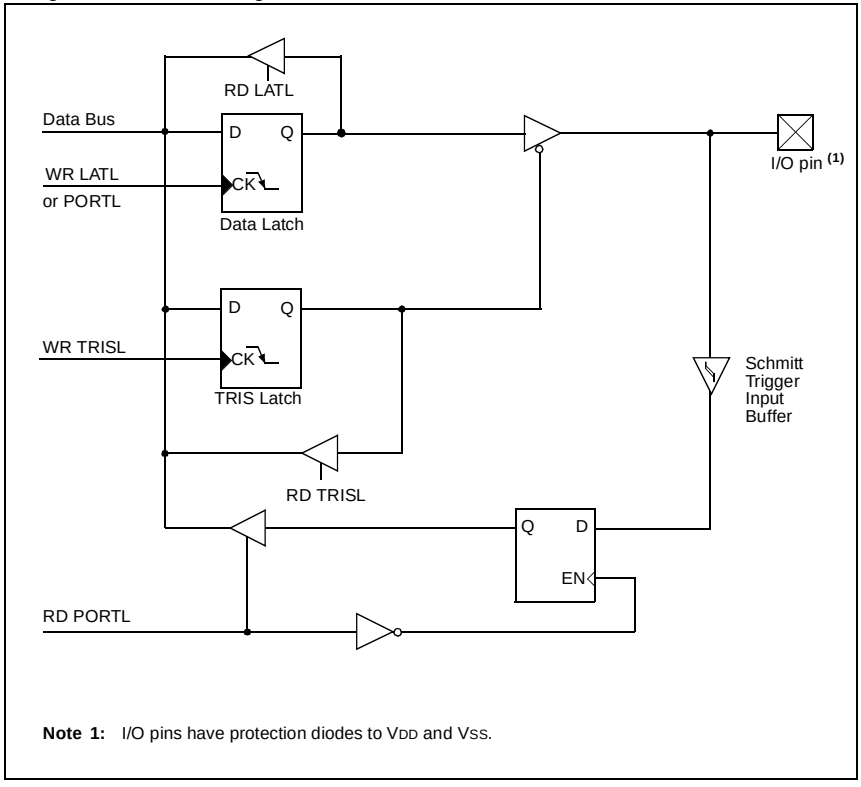
Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISK	PORTK Data Direction Control Register								1111 1111	1111 1111
PORTK	Read PORTK pin / Write PORTK Data Latch								xxxx xxxx	uuuu uuuu
LATK	Read PORTK Data Latch/Write PORTK Data Latch								xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged.

11.12 PORTL, LATL, and the TRISL Register

PORTL is a 8-bit port with Schmitt Trigger input buffers. Each pin is individually configured as an input or output.

Figure 11-15: Block Diagram of PORTL Pins



Note 1: I/O pins have protection diodes to VDD and VSS.

Table 11-21: PORTL Functions

Name	Bit#	Buffer Type	Function
RL0	bit0	ST	Input/output port pin
RL1	bit1	ST	Input/output port pin
RL2	bit2	ST	Input/output port pin
RL3	bit3	ST	Input/output port pin
RL4	bit4	ST	Input/output port pin
RL5	bit5	ST	Input/output port pin
RL6	bit6	ST	Input/output port pin
RL7	bit7	ST	Input/output port pin

Legend: ST = Schmitt Trigger input.

Table 11-22: Summary of Registers Associated with PORTL

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TRISL	PORTL Data Direction Control Register								1111 1111	1111 1111
PORTL	Read PORTL pin / Write PORTL Data Latch								xxxx xxxx	uuuu uuuu
LATL	Read PORTL Data Latch/Write PORTL Data Latch								xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged.

11.13 Functions Multiplexed on I/O Pins

This section discusses a couple of functions that are multiplexed on to I/O pins that are new concepts when compared to the Mid-Range family.

11.13.1 Oscillator Configuration

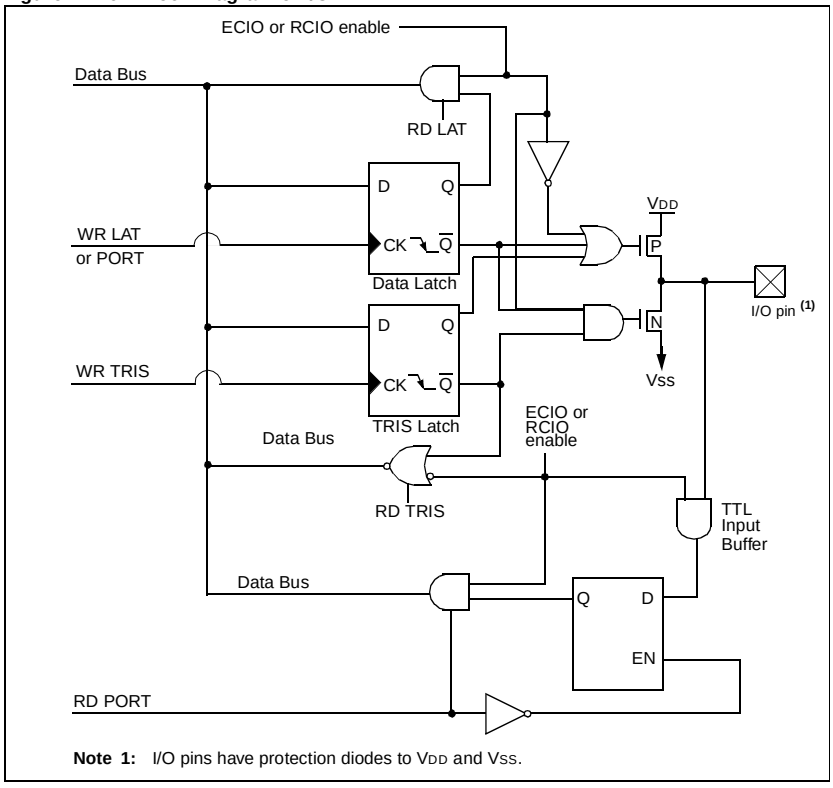
If the system oscillator uses RCIO or ECIO mode, then the OSC2 pin may be used as a general purpose I/O pin. If any other oscillator mode is used, the I/O pin multiplexed with OSC2 is disabled and will read '0', as will the TRIS bit and LAT bit associated with the I/O pin. Writes to I/O pin will have no effect. See Table 11-23.

If the system oscillator uses RC or EC mode, then the I/O pin is configured as OSC2 and outputs Fosc/4.

Table 11-23: RA6 Configuration for Oscillator Configuration

Oscillator Configuration	TRIS	PORT	LAT	OSC2 / I/O Function
RCIO / ERIO	Read / Write	Read / Write	Read / Write	General I/O
RC / EC	Disabled (reads 0)	Disabled (reads 0)	Disabled (reads 0)	$F_{OSC}/4$
Other system oscillator modes	Disabled (reads 0)	Disabled (reads 0)	Disabled (reads 0)	OSC2

Figure 11-16: Block Diagram of I/O

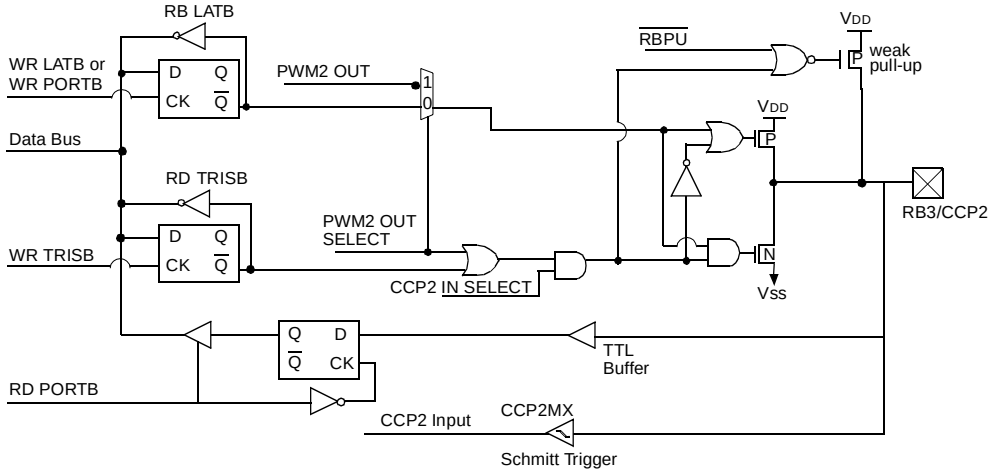


Note 1: I/O pins have protection diodes to VDD and VSS.

11.13.2 CCP2 Pin Multiplexing

In the PIC18CXX2 devices, the RB3 pin can be multiplexed with the CCP2 module input/output. To achieve this, the CCP2MX configuration bit must be programmed to a '0'.

Figure 11-17: Block Diagram of RB3



Note: I/O pin has diode protection to VDD and VSS.
Note: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the $\overline{\text{RBPU}}$ bit (INTCON2).
Note: The CCP2 input/output is multiplexed with RB3 if the CCP2MX bit is enabled (='0') in the configuration register.

11.14 I/O Programming Considerations

When using the ports as I/O, design considerations need to be taken into account to ensure that the operation is as intended.

11.14.1 Bi-directional I/O Ports

Any instruction that performs a write operation, actually does a read followed by a write operation. The BCF and BSF instructions, for example, read the register into the CPU, execute the bit operation and write the result back to the register. Caution must be used when these instructions are applied to a port with both inputs and outputs defined. For example, a BSF operation on bit5 of PORTB will cause all eight bits of PORTB to be read into the CPU. Then the BSF operation takes place on bit5 and PORTB is written to the output latches. If another bit of PORTB is used as a bi-directional I/O pin (e.g., bit0) and it is defined as an input at this time, the input signal present on the pin itself would be read into the CPU and rewritten to the data latch of this particular pin, overwriting the previous content. As long as the pin stays in the input mode, no problem occurs. However, if bit0 is switched to an output, the content of the data latch may now be unknown.

Reading the port register, reads the values of the port pins. Writing to the port register writes the value to the port latch. When using read-modify-write instructions (e.g., BCF, BSF, etc.) on a port, the value of the port pins is read, the desired operation is performed on this value, and the value is then written to the port latch.

Example 11-13 shows the effect of two sequential read-modify-write instructions on an I/O port.

Example 11-13: Read-Modify-Write Instructions on an I/O Port

```
; Initial PORT settings: PORTB<7:4> Inputs
;                        PORTB<3:0> Outputs
; PORTB<7:6> have external pull-ups and are not connected to other circuitry
;
;                       PORT latch  PORT pins
;                       ----------  ---------
    BCF  PORTB, 7     ; 01pp pppp    11pp pppp
    BCF  PORTB, 6     ; 10pp pppp    11pp pppp
    BCF  TRISB, 7     ; 10pp pppp    11pp pppp
    BCF  TRISB, 6     ; 10pp pppp    10pp pppp
;
; Note that the user may have expected the pin values to be 00pp ppp.
; The 2nd BCF caused RB7 to be latched as the pin value (high).
```

A pin configured as an output, actively driving a Low or High, should not be driven from external devices at the same time in order to change the level on this pin (“wired-or”, “wired-and”). The resulting high output currents may damage the chip.

11.14.2 Successive Operations on an I/O Port

The actual write to an I/O port happens at the end of an instruction cycle, whereas for reading, the data must be valid at the beginning of the instruction cycle (Figure 11-18). Therefore, care must be exercised if a write followed by a read operation is carried out on the same I/O port. The sequence of instructions should be such to allow the pin voltage to stabilize (load dependent) before the next instruction that causes that file to be read into the CPU is executed. Otherwise, the previous state of that pin may be read into the CPU rather than the new state. When in doubt, it is better to separate these instructions with a NOP or another instruction not accessing this I/O port.

This example shows a write to PORTB followed by a read from PORTB.

Note:	Data setup time = (0.25TCY - TPD), where TCY = instruction cycle, TPD = propagation delay.

Therefore, at higher clock frequencies, a write followed by a read may be problematic due to external capacitance.

Figure 11-18: Successive I/O Operation

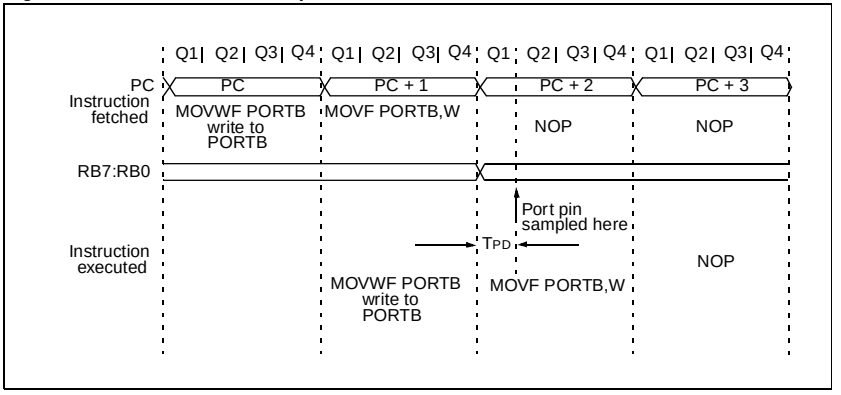
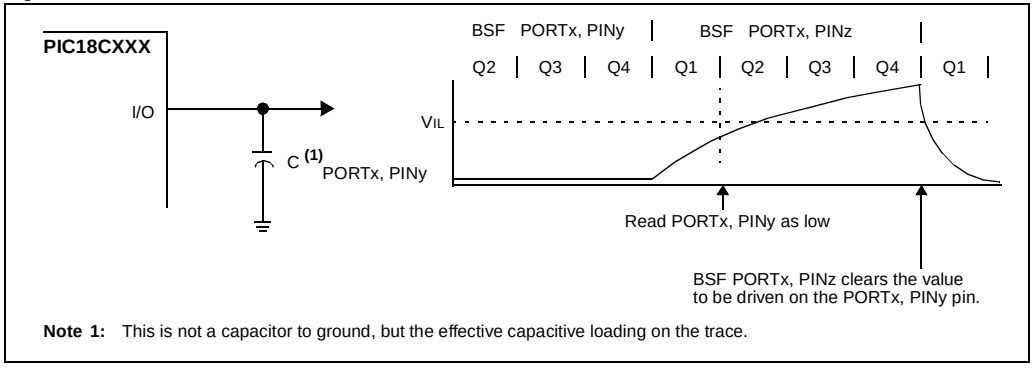


Figure 11-19 shows the I/O model that causes this situation. As the effective capacitance (C) becomes larger, the rise/fall time of the I/O pin increases. As the device frequency increases or the effective capacitance increases, the possibility of this subsequent PORTx read-modify-write instruction issue increases. This effective capacitance includes the effects of the board traces.

A way to address this is to add an series resistor at the I/O pin. This resistor allows the I/O pin to get to the desired level before the next instruction.

The use of NOP instructions between the subsequent PORTx read-modify-write instructions, is a lower cost solution, but has the issue that the number of NOP instructions is dependent on the effective capacitance C and the frequency of the device.

Figure 11-19: I/O Connection Issues



Note 1: This is not a capacitor to ground, but the effective capacitive loading on the trace.

11.15 Initialization

See the section describing each port for examples of initialization of the ports.

Note:	It is recommended that when initializing the port, the PORT data latch (LAT or PORT register) should be initialized first, and then the data direction (TRIS register). This will eliminate a possible pin glitch, since the LAT register (PORT data latch values) power up in a random state.

11.16 Design Tips

Question 1: ***Code will not toggle any I/O ports, but the oscillator is running. What can I be doing wrong?***

Answer 1:

1. Have the TRIS registers been initialized properly? These registers can be written to directly in the access bank (Bank15).
2. Is there a peripheral multiplexed onto those pins that are enabled?
3. Is the Watchdog Timer enabled (done at programming)? If it is enabled, is it being cleared properly with a CLRWDT instruction at least every 9 ms (or more if prescaled)?
4. Are you using the correct instructions to write to the port? More than one person has used the MOVF command when they should have used MOVWF.
5. For parts with interrupts, are the interrupts disabled? If not, try disabling them to verify they are not interfering.

Question 2: ***When my program reads a port, I get a different value than what I put in the port register. What can cause this?***

Answer 2:

1. When a port is read, it is always the pin that is read, regardless of its being set to input or output. So if a pin is set to an input, you will read the value on the pin regardless of the register value.
2. If a pin is set to output, for instance, it has a one in the data latch; if it is shorted to ground, you will still read a zero on the pin. This is very useful for building fault tolerant systems, or handling I^2C bus conflicts. (The I^2C bus is only driven low, and the pin is high impedance for a one. If the pin is low and you are not driving it, some other device is trying to take the bus).
3. Enhanced devices all have at least one open drain (or open collector) pin. These pins can only drive a zero or high impedance. For most Enhanced devices, this is pin RA4. Open drain pins must have a pull-up resistor to have a high state. This pin is useful for driving odd voltage loads. The pull-up can be connected to a voltage (typically less than V_{DD}) which becomes the high state.
4. Some analog modules, when enabled, will force a read value of '0' from the pin, regardless of the voltage level on the pin.

Question 3: ***I have a PIC18CXX2 with pin RB0 configured as an interrupt input, but am not getting interrupted. When I change my routine to poll the pin, it reads the high input and operates fine. What is the problem?***

Answer 3:

PORTB accepts TTL input levels (on most parts), so when you have an input of say 3V (with V_{DD} = 5V), you will read a '1'. However, the buffer to the interrupt structure from pin RB0 is a Schmitt Trigger, which requires a higher voltage (than TTL input) before the high input is registered. So it is possible to read a '1', but not get the interrupt. The interrupt was given a Schmitt Trigger input with hysteresis to minimize noise problems. It is one thing to have short noise spikes on a pin that is a data input that can potentially cause bad data, but quite another to permit noise to cause an interrupt, hence the difference.

Question 4: ***When I perform a BCF instruction, other pins get cleared in the port. Why?***

Answer 4:

1. Another case where a read-modify-write instruction may seem to change other pin values unexpectedly can be illustrated as follows: Suppose you make PORTC all outputs, and drive the pins low. On each of the port pins is an LED connected to ground, such that a high output lights it. Across each LED is a 100 µF capacitor. Let's also suppose that the processor is running very fast, say 20 MHz. Now if you go down the port, setting each pin in order; `BSF PORTC,0` then `BSF PORTC,1` then `BSF PORTC,2` and so on, you may see that only the last pin was set, and only the last LED actually turns on. This is because the capacitors take a while to charge. As each pin was set, the pin before it was not charged yet, and so was read as a zero. This zero is written back out to the port latch (r-m-w, remember), which clears the bit you just tried to set in the previous instruction. This is usually only a concern at high speeds and for successive port operations, but it can happen, so take it into consideration.
2. If this is on a PIC18CXXX device with A/D, you have not configured the I/O pins properly in the ADCON1 register. If a pin is configured for analog input, any read of that pin will read a zero, regardless of the voltage on the pin. This is an exception to the normal rule that the pin state is always read. You can still configure an analog pin as an output in the TRIS register, and drive the pin high or low by writing to it, but you will always read a zero. Therefore, if you execute a Read-Modify-Write instruction (see previous question), all analog pins are read as zero; those not directly modified by the instruction will be written back to the port latch as zero. A pin configured as analog is expected to have values that may be neither high nor low to a digital pin, or floating. Floating inputs on digital pins are a no-no, and can lead to high current draw in the input buffer, so the input buffer is disabled.

11.17 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is they may be written for the Baseline, the Midrange, or High-end families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to I/O ports are:

Title	Application Note #
Improving the Susceptibility of an Application to ESD	AN595
Clock Design using Low Power/Cost Techniques	AN615
Implementing Wake-up on Keystroke	AN528
Interfacing to AC Power Lines	AN521
Multiplexing LED Drive and a 4 x 4 Keypad Sampling	AN529
Using PIC16C5X as an LCD Drivers	AN563
Serial Port Routines Without Using TMR0	AN593
Implementation of an Asynchronous Serial I/O	AN510
Using the PORTB Interrupt on Change Feature as an External Interrupt	AN566
Implementing Wake-up on Keystroke	AN522
Apple Desktop Bus	AN591
Software Implementation of Asynchronous Serial I/O	AN555
Communicating with the I^2C Bus using the PIC16C5X	AN515
Interfacing 93CX6 Serial EEPROMs to the PIC16C5X Microcontrollers	AN530
Logic Powered Serial EEPROMs	AN535
Interfacing 24LCXXB Serial EEPROMs to the PIC16C54	AN567
Using the 24XX65 and 24XX32 with Stand-alone PIC16C54 Code	AN558

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

11.18 Revision History

Revision A

This is the initial released revision of the Enhanced MCU I/O Ports description.

Section 12. Parallel Slave Port

HIGHLIGHTS

This section of the manual contains the following major topics:

12.1 Introduction

Some devices have an 8-bit wide Parallel Slave Port (PSP). This port is multiplexed onto one of the device's I/O ports. The port operates as an 8-bit wide Parallel Slave Port, or microprocessor port, when the PSPMODE control bit is set. In this mode, the input buffers are TTL.

In slave mode, the module is asynchronously readable and writable by the external world through the $\overline{RD}$ control input pin and the $\overline{WR}$ control input pin.

It can directly interface to an 8-bit microprocessor data bus. The external microprocessor can read or write the PORT latch as an 8-bit latch. Setting the PSPMODE bit enables port pins to be the $\overline{RD}$ input, the $\overline{WR}$ input, and the $\overline{CS}$ (chip select) input.

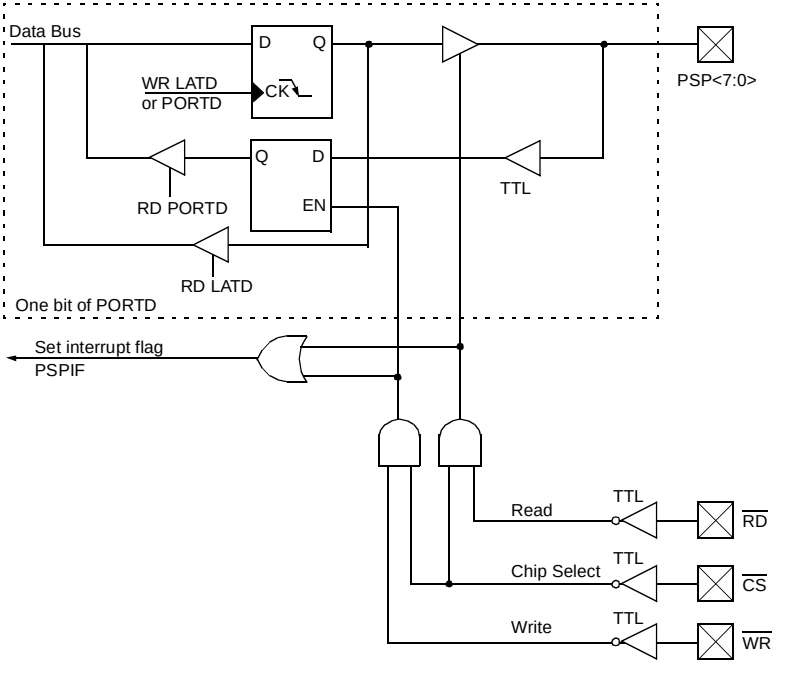
Note 1: At present the Parallel Slave Port (PSP) is only multiplexed onto PORTD and PORTE. The microprocessor port becomes enabled when the PSPMODE bit is set. In this mode, the user must make sure that PORTD and PORTE are configured as digital I/O. That is, peripheral modules multiplexed onto the PSP functions are disabled (such as the A/D).
When PORTE is configured for digital I/O, PORTD will override the values in the TRISD register.

2: In this mode the PORTD and PORTE input buffers are TTL. The control bits for the PSP operation are located in TRISE.

There are actually two 8-bit latches, one for data-out (from the PICmicro) and one for data input. The user writes 8-bit data to the PORT data latch and reads data from the port pin latch (note that they have the same address). In this mode, the TRIS register is ignored, since the microprocessor is controlling the direction of data flow.

Register 12-1 shows the block diagram for the PSP module.

Figure 12-1: PORTD and PORTE Block Diagram (Parallel Slave Port)



Note: I/O pins have protection diodes to VDD and VSS.

12.2 Control Register

Register 12-1 is the PSP control register (PSPCON). The TRISE register (Register 12-2) contains the 4 bits for the PSP module found in some devices (such as PIC18C4X2) for compatibility with 40-pin midrange devices.

Register 12-1: PSPCON Register

R-0	R-0	R/W-0	R/W-0	U-0	U-0	U-0	U-0
IBF	OBF	IBOV	PSPMODE	—	—	—	—
bit 7							bit 0

bit 7 **IBF:** Input Buffer Full Status bit

1 = A word has been received and waiting to be read by the CPU
0 = No word has been received

bit 6 **OBF**: Output Buffer Full Status bit

1 = The output buffer still holds a previously written word
0 = The output buffer has been read

bit 5 **IBOV**: Input Buffer Overflow Detect bit (in microprocessor mode)

1 = A write occurred when a previously input word has not been read (must be cleared in software)

0 = No overflow occurred

bit 4 **PSPMODE**: Parallel Slave Port Mode Select bit

1 = Parallel slave port mode
0 = General purpose I/O mode

bits 3:0 **Unimplemented:** Read as '0'

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 12-2: TRISE Register

R-0	R-0	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1
IBF	OBF	IBOV	PSPMODE	—	TRISE2	TRISE1	TRISE0
bit 7							bit 0

bit 7 **IBF:** Input Buffer Full Status bit

1 = A word has been received and waiting to be read by the CPU
0 = No word has been received

bit 6 **OBF**: Output Buffer Full Status bit

1 = The output buffer still holds a previously written word
0 = The output buffer has been read

bit 5 **IBOV**: Input Buffer Overflow Detect bit (in microprocessor mode)

1 = A write occurred when a previously input word has not been read (must be cleared in software)

0 = No overflow occurred

bit 4 **PSPMODE**: Parallel Slave Port Mode Select bit

1 = Parallel slave port mode
0 = General purpose I/O mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **TRISE2**: RE2 Direction Control bit

1 = Input
0 = Output

bit 1 **TRISE1**: RE1 Direction Control bit

1 = Input
0 = Output

bit 0 **TRISE0**: RE0 Direction Control bit

1 = Input
0 = Output

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

12.3 Operation

A write to the PSP from the external system occurs when both the $\overline{CS}$ and $\overline{WR}$ lines are first detected low. When either the $\overline{CS}$ or $\overline{WR}$ lines become high (edge triggered), the Input Buffer Full status flag bit IBF is set on the Q4 clock cycle following the next Q2 cycle. This signals that the write is complete. The interrupt flag bit, PSPIF, is also set on the same Q4 clock cycle. The IBF flag bit is inhibited from being cleared for additional TCY cycles (see parameter 66 in the **"Electrical Specifications"** section). If the IBF flag bit is cleared by reading the PORTD input latch, then this has to be a read-only instruction (i.e., MOVF) and not a read-modify-write instruction. The Input Buffer Overflow status flag bit IBOV is set if a second write to the Parallel Slave Port is attempted when the previous byte has not been read out of the buffer.

A read of the PSP from the external system occurs when both the $\overline{CS}$ and $\overline{RD}$ lines are first detected low. The Output Buffer Full status flag bit OBF is cleared immediately indicating that the PORTD latch was read by the external bus. When either the $\overline{CS}$ or $\overline{RD}$ pin becomes high (edge triggered), the interrupt flag bit, PSPIF, is set on the Q4 clock cycle following the next Q2 cycle, indicating that the read is complete. OBF remains low until data is written to PORTD by the user firmware.

Input Buffer Full Status Flag bit, IBF, is set if a received word is waiting to be read by the CPU. Once the PORT input latch is read, the IBF bit is cleared. The IBF bit is a read only status bit. Output Buffer Full Status Flag bit, OBF, is set if a word written to the PORT latch is waiting to be read by the external bus. Once the PORTD output latch is read by the microprocessor, OBF is cleared. Input Buffer Overflow Status Flag bit, IBOV, is set if a second write to the microprocessor port is attempted when the previous word has not been read by the CPU (the first word is retained in the buffer).

When not in Parallel Slave Port mode, the IBF and OBF bits are held clear. However, if the IBOV bit was previously set, it must be cleared in the software.

An interrupt is generated and latched into flag bit PSPIF when a read or a write operation is completed. Interrupt flag bit PSPIF must be cleared by user software and the interrupt can be disabled by clearing interrupt enable bit PSPIE.

Table 12-1: PORTE Functions

Name	Function
$\overline{RD}$	Read Control Input in parallel slave port mode: $\overline{RD}$ 1 = Not a read operation 0 = Read operation. Reads PORTD register (if chip selected)
$\overline{WR}$	Write Control Input in parallel slave port mode: $\overline{WR}$ 1 = Not a write operation 0 = Write operation. Writes PORTD register (if chip selected)
$\overline{CS}$	Chip Select Control Input in parallel slave port mode: $\overline{CS}$ 1 = Device is not selected 0 = Device is selected

Note: The PSP may have other functions multiplexed onto the same pins. For the PSP to operate, the pins must be configured as digital I/O.

12.4 Operation in SLEEP Mode

When in SLEEP mode, the microprocessor may still read and write the Parallel Slave Port. These actions will set the PSPIF bit. If the PSP interrupts are enabled, this will wake the processor from SLEEP mode so that the PSP data latch may be either read, or written with the next value for the microprocessor.

12.5 Effect of a RESET

After any RESET, the PSP is disabled and PORTD and PORTE are forced to their default mode.

12.6 PSP Waveforms

Register 12-2 shows the waveform for a write from the microprocessor to the PSP, while Register 12-3 shows the waveform for a read of the PSP by the microprocessor.

Figure 12-2: Parallel Slave Port Write Waveforms

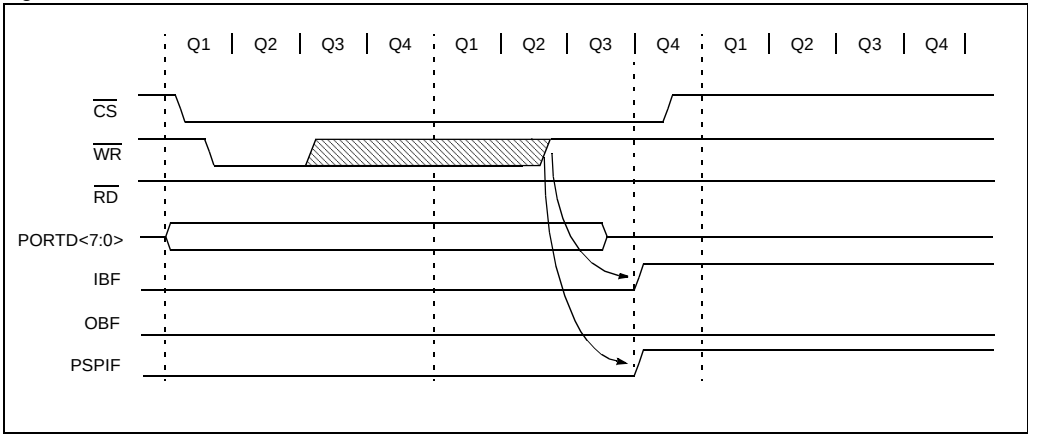


Figure 12-3: Parallel Slave Port Read Waveforms

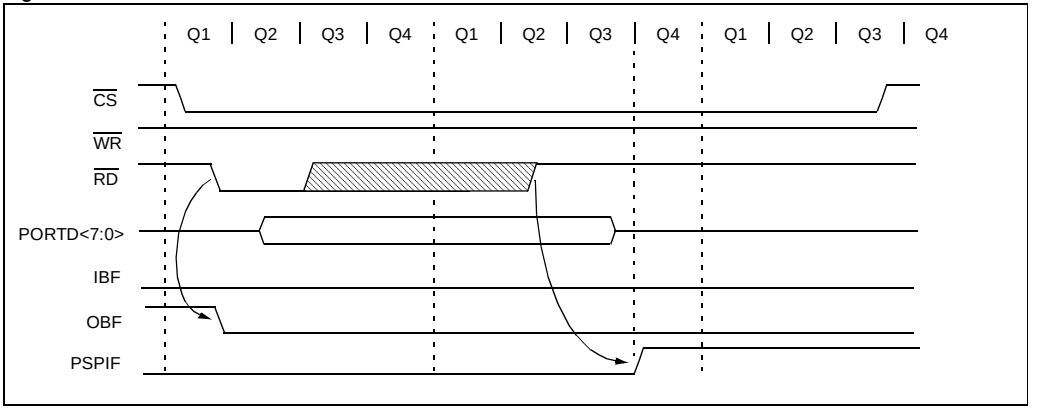


Table 12-2: Registers Associated with Parallel Slave Port

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other resets
PORTD	Port data latch when written; port pins when read								xxxx xxxx	uuuu uuuu
LATD	LATD Data Output Bits								xxxx xxxx	uuuu uuuu
TRISD	PORTD Data Direction Bits								1111 1111	1111 1111
PORTE (1)	—	—	—	—	—	RE2	RE1	RE0	---- -000	---- -000
LATE	—	—	—	—	—	LATE Data Output Bits			---- -xxx	---- -uuu
TRISE (1)	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111
PSPCON	IBF	OBF	IBOV	PSPMODE	—	—	—	—	0000 ----	0000 ----
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IF	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	0000 0000	0000 0000
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	--0- -000	--0- -000

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'.
Shaded cells are not used by the Parallel Slave Port.

Note 1: On some devices the entire PORTE will be implemented with I/O functions. In these devices, the TRISE register will contain the eight data direction bits and the PSP bits will be located in the PSPCON register.

12.7 Design Tips

Question 1: ***Migrating from the PIC16C74 to the PIC18CXX2, the operation of the PSP seems to have changed.***

Answer 1:

Yes, a design change was made so the PIC18CXX2 is edge sensitive (while the PIC16C74 was level sensitive).

12.8 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced MCU family (that is, they may be written for the Base-Line, the Mid-Range or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the Parallel Slave Port are:

Title	Application Note #
Using the 8-bit Parallel Slave Port	AN579

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

12.9 Revision History

Revision A

This is the initial released revision of the Parallel Slave Port description.

Section 13. Timer0

HIGHLIGHTS

This section of the manual contains the following major topics:

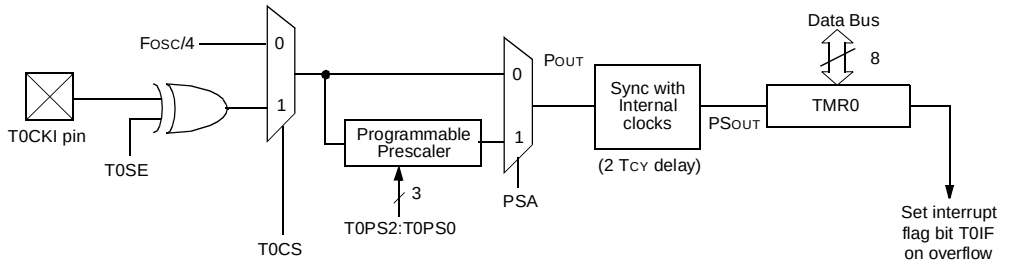
13.1 Introduction

The Timer0 module has the following features:

- Software selectable as an 8-bit or 16-bit timer/counter
- Readable and writable
- Dedicated 8-bit software programmable prescaler
- Clock source selectable to be external or internal
- Interrupt on overflow from FFh to 00h (FFFFh to 0000h in 16-bit mode)
- Edge select for external clock

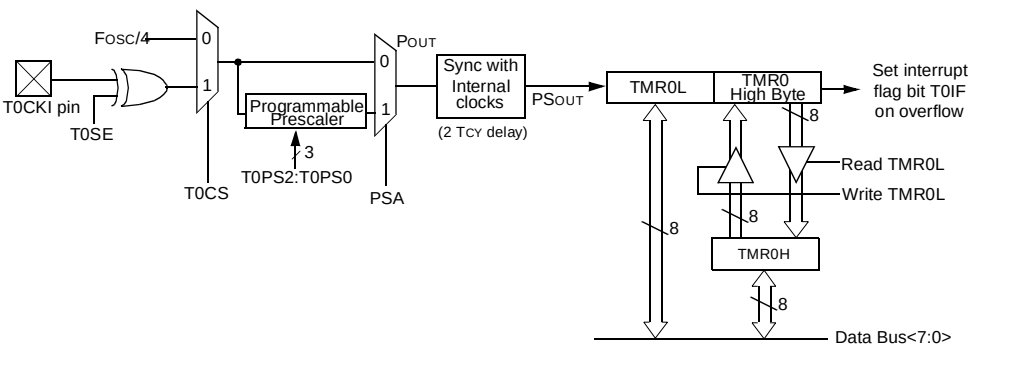
Figure 13-1 shows a simplified block diagram of the Timer0 module in 8-bit mode and Figure 13-2 shows a simplified block diagram of the Timer0 module in 16-bit mode.

Figure 13-1: Timer0 Block Diagram in 8-bit Mode



Note 1: T0CS, T0SE, PSA, T0PS2:T0PS0 (T0CON<5:0>).
2: Upon reset, Timer0 is enabled in 8-bit mode, with clock input from T0CKI, max. prescale.

Figure 13-2: Timer0 Block Diagram in 16-bit Mode



Note 1: T0CS, T0SE, PSA, T0PS2:T0PS0 (T0CON<5:0>).
2: Upon reset, Timer0 is enabled in 8-bit mode, with clock input from T0CKI, max. prescale.

13.2 Control Register

The T0CON register is a readable and writable register that controls all the aspects of Timer0, including the prescale selection.

Register 13-1: T0CON: TImer0 Control Register

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin is clock (counter mode)
0 = Internal instruction cycle is clock (timer mode)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = Timer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
These bits are ignored if PSA = 1

111 = 1:256 prescale value
110 = 1:128 prescale value
101 = 1:64 prescale value
100 = 1:32 prescale value
011 = 1:16 prescale value
010 = 1:8 prescale value
001 = 1:4 prescale value
000 = 1:2 prescale value

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

13.3 Operation

When initializing Timer0, several options need to be specified. This is done by programming the appropriate bits in the T0CON register.

13.3.1 8-Bit/16-Bit Modes

Timer0 can be configured as an 8-bit or a 16-bit counter. The default state for Timer0 is an 8-bit counter. To configure the timer as a 16-bit counter, the T08BIT bit (T0CON register) must be cleared.

If the timer is configured as an 8-bit timer, the MSB of TMR0 (TMR0H) is held clear and will read 00h.

Normally once the mode of the timer is selected, it is not changed. Some applications may require the ability to switch back and forth between 8-bit and 16-bit modes. The two cases are:

1. Changing from 8-bit to 16-bit mode
2. Changing from 16-bit to 8-bit mode

The condition when bit 7 of the Timer0 rolls over must be addressed.

If Timer0 is configured as an 8-bit timer and is changed to a 16-bit timer on the same cycle as a rollover occurs, no interrupt is generated.

If Timer0 is configured as a 16-bit timer and is changed to an 8-bit timer on the same cycle as a rollover occurs, the TMR0IF bit will be set.

13.3.1.1 16-Bit Mode Timer Reads

TMR0H is not the high byte of the timer/counter, but actually a buffered version of the high byte of Timer0. The high byte of the Timer0 counter/timer is not directly readable or writable. TMR0H is updated with the contents of the high byte of Timer0 during a read of TMR0L. This provides a user with the ability to read all 16 bits of Timer0 without having to verify that the read of the high and low byte were valid due to a rollover between successive reads of the high and low byte. The user simply reads the low byte of Timer0, followed by a read of TMR0H, which contains the value in the high byte of Timer0 at the time that the low byte was read.

13.3.1.2 16-Bit Mode Timer Write

A write to the high byte of Timer0 must also take place through the TMR0H buffer register. Timer0 high byte is updated with the contents of TMR0H when a write occurs to TMR0L. This allows a user to update all 16 bits to both the high and low bytes of Timer0 at once (see Figure 13-2).

When performing a write of TMR0, the carry is held off during the write of the TMR0L register. Writes to the TMR0H register only modify the holding latch, not the timer (TMR0<15:8>).

Steps to write to the TMR0:

1. Load the TMR0H register.
2. Write to the TMR0L register.

13.3.1.3 16-Bit Read/Modify Write

Read-modify-write instructions like BSF or BCF, read the contents of a register, make the appropriate changes, and place the result back into the register. The read cycle of a read-modify-write instruction of TMR0L will not update the contents of the TMR0H buffer. The TMR0H buffer will remain unchanged. When the write cycle (to TMR0L) of the instruction takes place, the contents of TMR0H are placed into the high byte of Timer0.

13.3.2 Timer/Counter Modes

Timer mode is selected by clearing the T0CS bit (T0CON register). In timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0 register is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

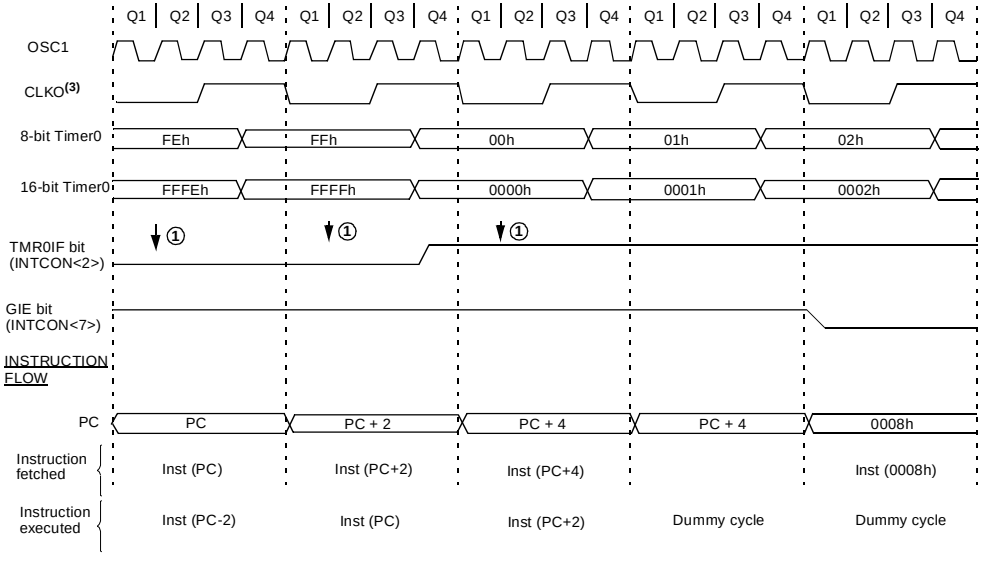
Counter mode is selected by setting the T0CS bit (T0CON register). In counter mode, Timer0 will increment either on every rising or falling edge of the T0CKI pin. The incrementing edge is determined by the Timer0 Source Edge Select bit T0SE (T0CON register). Clearing the T0SE bit selects the rising edge. Restrictions on the external clock input are discussed in detail in Section **13.5.1**.

13.4 Timer0 Interrupt

The TMR0 interrupt flag bit is set when the TMR0 register overflows. When TMR0 is in 8-bit mode, this means the overflow from FFh to 00h. When TMR0 is in 16-bit mode, this means the overflow from FFFFh to 0000h.

This overflow sets the TMR0IF bit (INTCON register). The interrupt can be disabled by clearing the TMR0IE bit (INTCON register). The TMR0IF bit must be cleared in software by the interrupt service routine. The TMR0 interrupt cannot awaken the processor from SLEEP, since the timer is shut off during SLEEP. See Figure 13-3 for Timer0 interrupt timing.

Figure 13-3: TMR0 Interrupt Timing



Note 1: Interrupt flag bit TMR0IF is sampled here (every Q1).
2: Interrupt latency = 4TCY where TCY = instruction cycle time.
3: CLKO is available only in RC oscillator mode.

13.5 Using Timer0 with an External Clock

When an external clock input is used for Timer0, it must meet certain requirements as detailed in **13.5.1 "External Clock Synchronization"**. The requirements ensure the external clock can be synchronized with the internal phase clock (TSCLK). Also, there is a delay in the actual incrementing of Timer0 after synchronization.

13.5.1 External Clock Synchronization

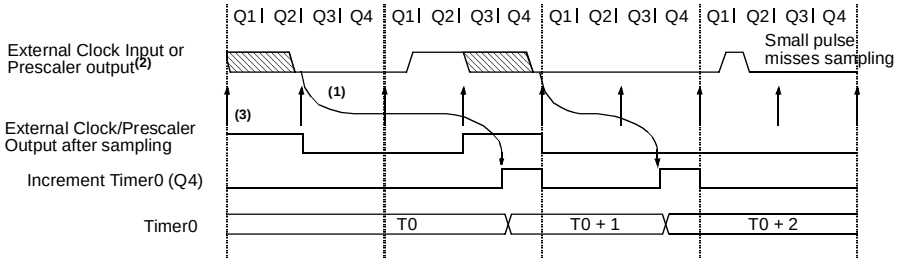
When no prescaler is used, the external clock input is used instead of the prescaler output. The synchronization of T0CKI with the internal phase clocks is accomplished by sampling the prescaler output on the Q2 and Q4 cycles of the internal phase clocks (Figure 13-4). Therefore, it is necessary for T0CKI to be high for at least 2TSCLK (and a small RC delay) and low for at least 2TSCLK (and a small RC delay). Refer to parameters 40, 41 and 42 in the electrical specification of the desired device.

When a prescaler is used, the external clock input is divided by the prescaler so that the prescaler output is symmetrical. For the external clock to meet the sampling requirement, the ripple-counter must be taken into account. Therefore, it is necessary for T0CKI to have a period of at least 4TSCLK (and a small RC delay) divided by the prescaler value. The only requirement on T0CKI high and low time is that they do not violate the minimum pulse width requirement. Refer to parameters 40, 41 and 42 in the electrical specification of the desired device.

13.5.2 TMR0 Increment Delay

Since the prescaler output is synchronized with the internal clocks, there is a small delay from the time the external clock edge occurs to the time the Timer0 module is actually incremented. Figure 13-4 shows the delay from the external clock edge to the timer incrementing.

Figure 13-4: Timer0 Timing with External Clock



Note 1: Delay from clock input change to Timer0 increment is 3Tosc to 7Tosc. (Duration of Q = Tosc). Therefore, the error in measuring the interval between two edges on Timer0 input = ±4Tosc max.

2: External clock if no prescaler selected, prescaler output otherwise.

3: The arrows indicate the points in time where sampling occurs.

13.6 Timer0 Prescaler

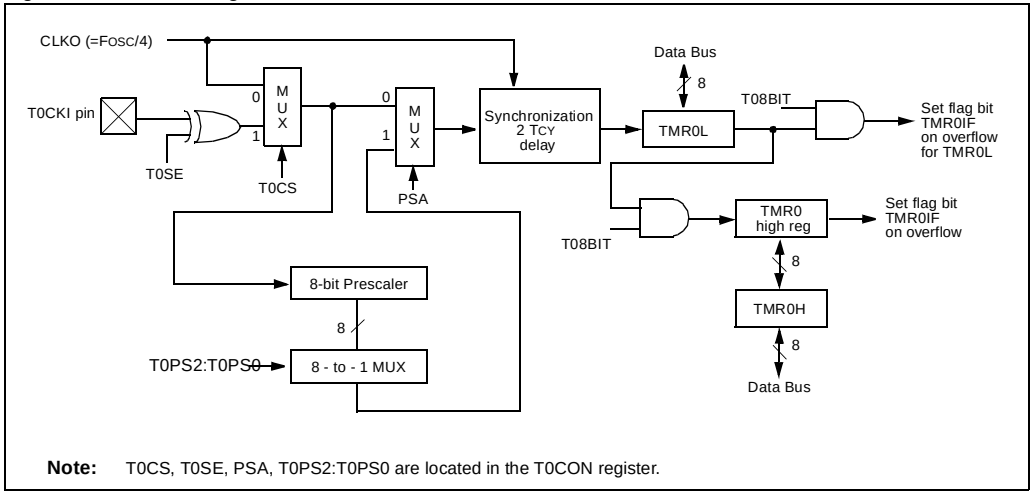
An 8-bit counter is available as a prescaler for the Timer0 module (Figure 13-5).

The PSA and T0PS2:T0PS0 bits (T0CON register) are the prescaler enable and prescale select bits.

All instructions that write to the Timer0 (TMR0) register (such as: `CLRF TMR0; BSF TMR0,x; MOVWF TMR0;` ....etc.) will clear the prescaler if enabled. The prescaler is not readable or writable.

Writes to TMR0H do not clear the Timer0 prescaler in 16-bit mode, because a write to TMR0H only modifies the Timer0 latch and does not change the contents of Timer0. The prescaler is only cleared on writes to TMR0L.

Figure 13-5: Block Diagram of the Timer0 Prescaler



Note: T0CS, T0SE, PSA, T0PS2:T0PS0 are located in the T0CON register.

The prescaler for Timer0 is enabled or disabled in software by the PSA bit (T0CON register). Setting the PSA bit will enable the prescaler. The prescaler can be modified under software control through the T0PS2:T0PS0 bits. This allows the prescaler reload value to be readable and writable. The prescaler count value (the contents of the prescaler) can not be read or written. When the prescaler is enabled, prescale values of 1:2, 1:4, ..., 1:256 are selectable.

Any write to the TMR0 register will cause a 2 instruction cycle (2TCY) inhibit. That is, after the TMR0 register has been written with the new value, TMR0 will not be incremented until the third instruction cycle later (Figure 13-6). When the prescaler is assigned to the Timer0 module, any write to the TMR0 register will immediately update the TMR0 register and clear the prescaler. The incrementing of Timer0 (TMR0 and Prescaler) will also be inhibited 2 instruction cycles (TCY). So if the prescaler is configured as 2, then after a write to the TMR0 register, TMR0 will not increment for 4 Timer0 clocks (Figure 13-7). After that, TMR0 will increment every prescaler number of clocks later.

Figure 13-6: Timer0 Timing: Internal Clock/No Prescale

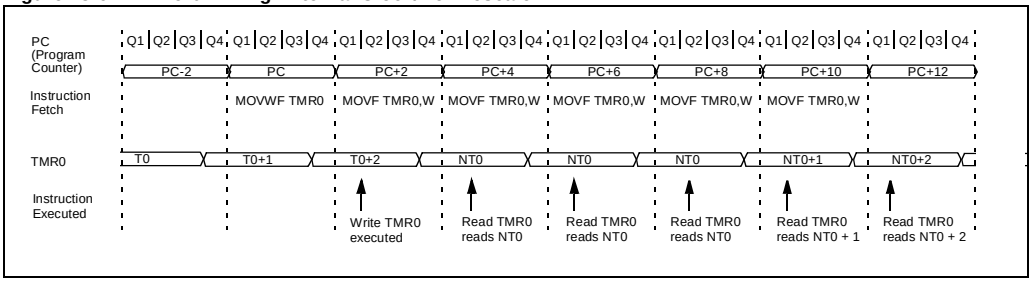


Figure 13-7: Timer0 Timing: Internal Clock/Prescale 1:2

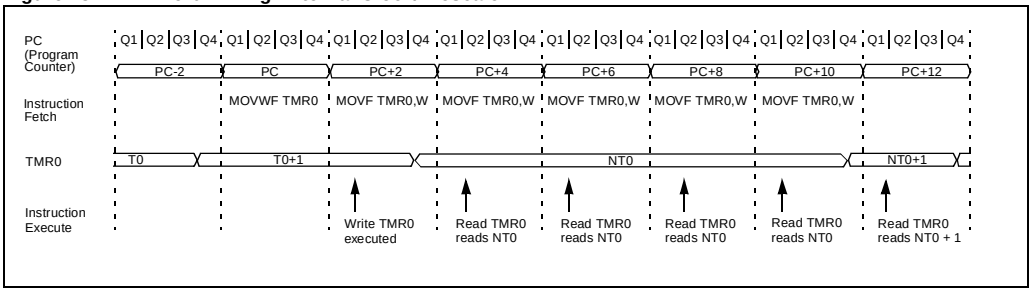


Table 13-1: Registers Associated with Timer0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other resets
TMR0L	Timer0 Module's Low Byte Register								xxxx xxxx	uuuu uuuu
TMR0H	Timer0 Module's High Byte Register								0000 0000	0000 0000
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	1111 1111	1111 1111
TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'.
Shaded cells are not used by Timer0.

13.7 Initialization

Since Timer0 has a software programmable clock source, there are two examples to show the initialization of Timer0 with each source. Example 13-1 shows the initialization for the internal clock source (timer mode), while Example 13-2 shows the initialization for the external clock source (counter mode).

Example 13-1: Timer0 Initialization (Internal Clock Source)

```
   CLRF   TMR0           ; Clear Timer0 register
   CLRF   INTCON         ; Disable interrupts and clear T0IF
   BCF    INTCON2, RBPU  ;
   MOVLW  0x80           ; PortB pull-ups are disabled,
   MOVWF  T0CON          ;   Interrupt on rising edge of RB0,
                         ;     TMR0 = 16-Bit Time
                         ;   Timer0 increment from internal clock
                         ;   with a prescaler of 1:2.
;**   BSF INTCON, T0IE   ; Enable TMR0 interrupt
;**   BSF    INTCON, GIE ; Enable all interrupts
;
; The TMR0 interrupt is disabled, do polling on the overflow bit
;
T0_OVFL_WAIT
        BTFSS  INTCON, T0IF
        GOTO   T0_OVFL_WAIT
; Timer has overflowed
```

Example 13-2: Timer0 Initialization (External Clock Source)

```
    CLRF   TMR0           ; Clear Timer0 register
    CLRF   INTCON         ; Disable interrupts and clear T0IF
    BCF    INTCON2, RBPU ;
    MOVLW  0xBF           ; PortB pull-ups are enabled,
    MOVWF  T0CON          ;   Interrupt on falling edge of RB0
                          ;   Timer0 increment from external clock
                          ;   on the high-to-low transition
                          ;   of T0CKI
                          ;   with a prescaler of 1:256.
;** BSF    INTCON, T0IE  ; Enable TMR0 interrupt
;** BSF    INTCON, GIE   ; Enable all interrupts
;
; The TMR0 interrupt is disabled, do polling on the overflow bit
;
T0_OVFL_WAIT
    BTFSS  INTCON, T0IF
    GOTO   T0_OVFL_WAIT
; Timer has overflowed
```


13.8 Design Tips

Question 1: ***I am implementing a counter/clock, but the clock loses time or is inaccurate.***

Answer 1:

If you are polling TMR0 to see if it has rolled over to zero, you could do this by executing:

```
wait   MOVF    TMR0,W     ; read the timer into W
       BTFSS   STATUS,Z   ; see if it was zero, if so,
                          ;    break from loop
       GOTO     wait      ; if not zero yet, keep waiting
```

Two possible scenarios to lose clock cycles are:

1. If you are incrementing TMR0 from the internal instruction clock (or an external source that is about as fast), the overflow could occur during the two cycle GOTO, so you could miss it. In this case, the TMR0 source should be prescaled.
2. When writing to TMR0, two instruction clock cycles are lost. Often you have a specific time period you want to count, say 100 decimal. In that case, you might put 156 into TMR0 (256 - 100 = 156). However, since two instruction cycles are lost when you write to TMR0 (for internal logic synchronization), you should actually write 158 to the timer.

13.9 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is, they may be written for the Base-Line, the Mid-Range or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to Timer0 are:

Title	Application Note #
Frequency Counter Using PIC16C5X	AN592
A Clock Design using the PIC16C54 for LED Display and Switch Inputs	AN590

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

13.10 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Timer0 Module description.

Section 14. Timer1

HIGHLIGHTS

This section of the manual contains the following major topics:

14.1 Introduction

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) that are readable and writable. The TMR1 register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. If enabled, the Timer1 Interrupt is generated on overflow that is latched in the TMR1IF interrupt flag bit. This interrupt can be enabled/disabled by setting/clearing the TMR1IE interrupt enable bit.

Timer1 can operate in one of three modes:

- As a synchronous timer
- As a synchronous counter
- As an asynchronous counter

The operating mode is determined by clock select bit, TMR1CS (T1CON register), and the synchronization bit, T1SYNC (Figure 14-1).

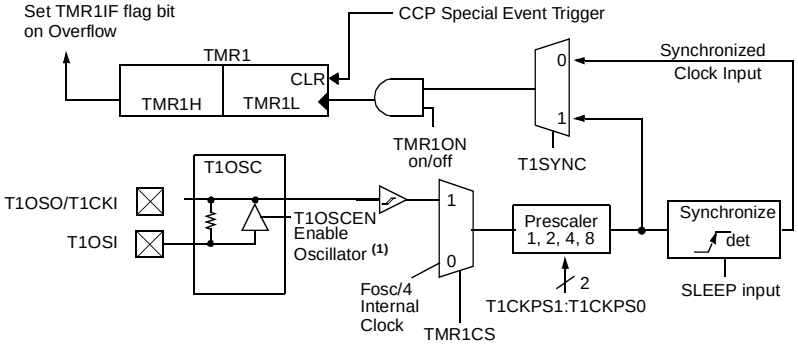
In timer mode, Timer1 increments every instruction cycle. In counter mode, it increments on every rising edge of the external clock input pin T1OSI.

Timer1 can be turned on and off using theTMR1ON control bit (T1CON register).

Timer1 also has an internal “reset input”, which can be generated by a CCP module.

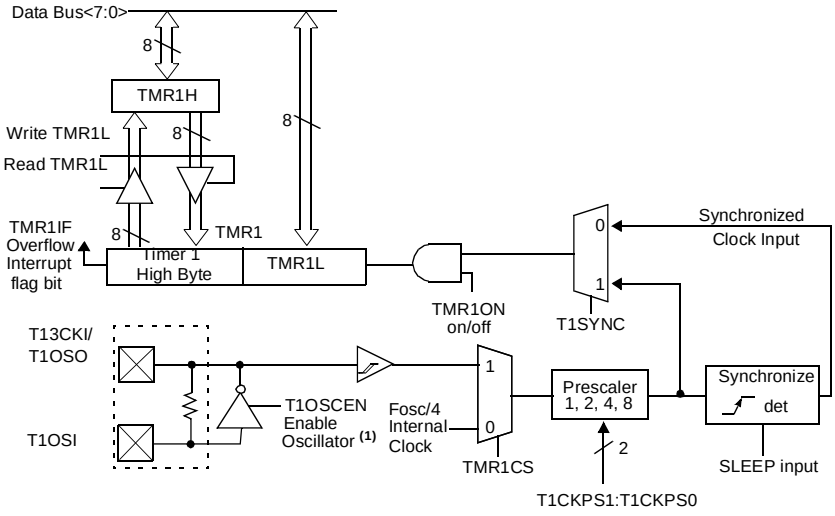
Timer1 has the capability to operate off an external crystal. When the Timer1 oscillator is enabled (T1OSCEN is set), the T1OSI and T1OSO pins become inputs, so their corresponding TRIS values are ignored.

Figure 14-1: Timer1 Block Diagram



Note 1: When enable bit T1OSCEN is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

Figure 14-2: Timer1 Block Diagram 16-Bit Read/Write Mode



Note 1: When enable bit T1OSCEN is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

14.2 Control Register

Register 14-1 shows the Timer1 Control register. This register controls the operating mode of the Timer1 module and contains the Timer1 oscillator enable bit (T1OSCEN).

Register 14-1: T1CON: Timer1 Control Register

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7 **RD16:** 16-bit Read/Write Mode Enable bit

1 = Enables register Read/Write of Timer1 in one 16-bit operation
0 = Enables register Read/Write of Timer1 in two 8-bit operations

bit 6 **Unimplemented:** Read as '0'

bit 5:4 **T1CKPS1:T1CKPS0**: Timer1 Input Clock Prescale Select bits

11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN**: Timer1 Oscillator Enable bit

1 = Timer1 Oscillator is enabled
0 = Timer1 Oscillator is shut off. The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **$\overline{\text{T1SYNC}}$**: Timer1 External Clock Input Synchronization Select bit

When TMR1CS = 1:

1 = Do not synchronize external clock input
0 = Synchronize external clock input

When TMR1CS = 0:

This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit

1 = External clock from pin T1OSO/T13CKI (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON**: Timer1 On bit

1 = Enables Timer1
0 = Stops Timer1

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

14.3 Timer1 Operation in Timer Mode

Timer mode is selected by clearing the TMR1CS (T1CON register) bit. In this mode, the input clock to the timer is FOSC/4. The synchronize control bit, T1SYNC (T1CON register), has no effect since the internal clock is always synchronized.

14.4 Timer1 Operation in Synchronized Counter Mode

Counter mode is selected by setting the TMR1CS bit. In this mode, the timer increments on every rising edge of clock input on the T1OSI pin when the Timer1 oscillator enable bit (T1OSCEN) is set, or the T1OSO/T13CKI pin when the T1OSCEN bit is cleared.

If the T1SYNC bit is cleared, then the external clock input is synchronized with internal phase clocks. The synchronization is done after the prescaler stage. The prescaler operates asynchronously.

The timer increments at the Q4:Q1 edge.

In this configuration, during SLEEP mode, Timer1 will not increment even if the external clock is present, since the synchronization circuit is shut off. The prescaler however will continue to increment.

14.4.1 External Clock Input Timing for Synchronized Counter Mode

When an external clock input is used for Timer1 in synchronized counter mode, it must meet certain requirements. The external clock requirement is due to internal phase clock (TSCLK) synchronization. Also, there is a delay in the actual incrementing of TMR1 after synchronization.

When the prescaler is 1:1, the external clock input is the same as the prescaler output. The synchronization of T1CKI with the internal phase clocks is accomplished by sampling the prescaler output on alternating TSCLK clocks of the internal phase clocks. Therefore, it is necessary for the T1CKI pin to be high for at least 2TSCLK (and a small RC delay) and low for at least 2TSCLK (and a small RC delay). Refer to parameters 45, 46, and 47 in the **"Electrical Specifications"** section.

When a prescaler other than 1:1 is used, the external clock input is divided by the asynchronous prescaler so that the prescaler output is symmetrical. In order for the external clock to meet the sampling requirement, the prescaler counter must be taken into account. Therefore, it is necessary for the T1CKI pin to have a period of at least 4TSCLK (and a small RC delay) divided by the prescaler value. Another requirement on the T1CKI pin high and low time is that they do not violate the minimum pulse width requirements). Refer to parameters 40, 42, 45, 46, and 47 in the **"Electrical Specifications"** section.

14.5 Timer1 Operation in Asynchronous Counter Mode

If $\overline{\text{T1SYNC}}$ (T1CON register) is set, the external clock input is not synchronized. The timer continues to increment asynchronously to the internal phase clocks. The timer will continue to run during SLEEP and can generate an interrupt on overflow that will wake-up the processor. However, special precautions in software are needed to read/write the timer (Subsection **14.6.4 "Reading and Writing Timer1 in Asynchronous Counter Mode with RD16 = 0"**). Since the counter can operate in sleep, Timer1 can be used to implement a true real-time clock.

The timer increments at the Q4:Q1 and Q2:Q3 edges.

In asynchronous counter mode, Timer1 cannot be used as a time-base for capture or compare operations.

14.5.1 External Clock Input Timing with Unsynchronized Clock

If the $\overline{\text{T1SYNC}}$ control bit is set, the timer will increment completely asynchronously. The input clock must meet certain minimum high time and low time requirements. Refer to the Device Data Sheet **"Electrical Specifications"** section, timing parameters 45, 46, and 47.

14.6 Reading and Writing of Timer1

Timer1 has modes that allow the 16-bit timer register to be read/written as two 8-bit registers or one 16-bit register. The mode depends on the state of the RD16 bit. The following subsections discuss this operation.

14.6.1 Timer1 and 16-bit Read/Write Modes

Timer1 can be configured for 16-bit reads and writes. When the RD16 control bit (T1CON register) is set, the address for TMR1H is mapped to a buffer register for the high byte of Timer1. A read from TMR1L will load the contents of the high byte of Timer1 into the Timer1 high byte buffer. This provides the user with the ability to accurately read all 16 bits of Timer1 without having to determine whether a read of the high byte followed by a read of the low byte is valid due to a rollover between reads.

14.6.2 16-bit Mode Timer Write

A write to the high byte of Timer1 must also take place through the TMR1H buffer register. Timer1 high byte is updated with the contents of TMR1H when a write occurs to TMR1L. This allows a user to write all 16 bits to both the high and low bytes of Timer1 at once (See Figure 14-3).

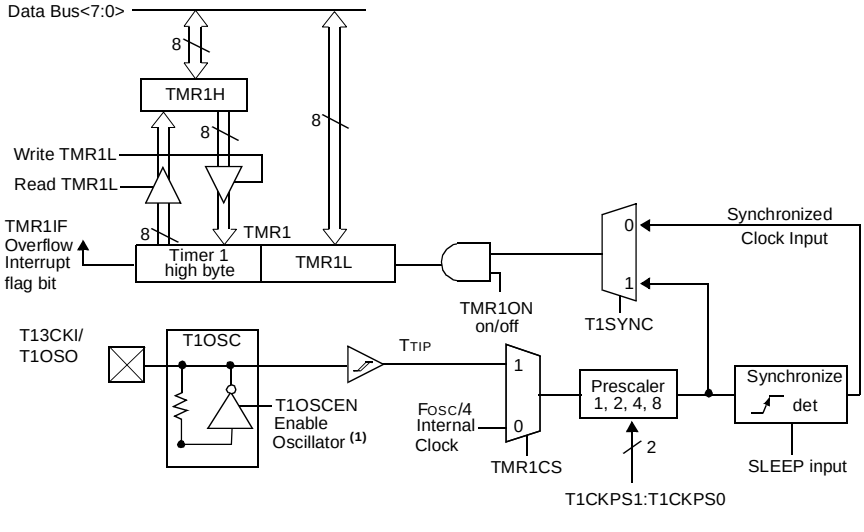
The high byte of Timer1 is not directly readable or writable in this mode. All reads and writes must take place through the Timer1 high byte buffer register.

Writes to TMR1H do not clear the Timer1 prescaler. The prescaler is only cleared on writes to TMR1L.

14.6.3 16-bit Read-Modify-Write

Read-modify-write instructions like BSF or BCF will read the contents of a register, make the appropriate changes, and place the result back into the register. In the case of Timer1 when configured in 16-bit mode, the read portion of a read-modify-write instruction of TMR1L will not update the contents of the TMR1H buffer. The TMR1H buffer will remain unchanged. When the write of TMR1L portion of the instruction takes place, the contents of TMR1H will be placed into the high byte of Timer1.

Figure 14-3: Timer1 Block Diagram When Configured in 16-bit Read/Write Mode



Note 1: When enable bit T1OSCEN is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

14.6.4 Reading and Writing Timer1 in Asynchronous Counter Mode with RD16 = 0

Reading TMR1H or TMR1L while the timer is running from an external asynchronous clock will ensure a valid read (taken care of in hardware). However, the user should keep in mind that reading the 16-bit timer in two 8-bit values itself poses certain problems, since the timer may overflow between the reads.

For writes, it is recommended that the user simply stop the timer and write the desired values. A write contention may occur by writing to the timer registers while the register is incrementing. This may produce an unpredictable value in the timer register.

Reading the 16-bit value requires some care, since two separate reads are required to read the entire 16-bits. Example 14-1 shows why this may not be a straight forward read of the 16-bit register.

Example 14-1: Reading 16-bit Register Issues

TMR1	Sequence 1		Sequence 2	
	Action	TMPH:TMPL	Action	TMPH:TMPL
04FFh	READ TMR1L	xxxxh	READ TMR1H	xxxxh
0500h	Store in TMPL	xxFFh	Store in TMPH	04xxh
0501h	READ TMR1H	xxFFh	READ TMR1L	04xxh
0502h	Store in TMPH	05FFh	Store in TMPL	0401h

Example 14-2 shows a routine to read the 16-bit timer value without experiencing the issues shown in Example 14-1. This is useful if the timer cannot be stopped.

Example 14-2: Reading a 16-bit Free-Running Timer

```
; All interrupts are disabled
       MOVF   TMR1H, W    ; Read high byte
       MOVWF  TMPH        ;
       MOVF   TMR1L, W    ; Read low byte
       MOVWF  TMPL        ;
       MOVF   TMR1H, W    ; Read high byte
       SUBWF  TMPH,  W    ; Sub 1st read with 2nd read
       BTFSC  STATUS,Z    ; Is result = 0
       GOTO   CONTINUE    ; Good 16-bit read
;
; TMR1L may have rolled over between the read of the high and low bytes.
; Reading the high and low bytes now will read a good value.
;
       MOVF   TMR1H, W    ; Read high byte
       MOVWF  TMPH        ;
       MOVF   TMR1L, W    ; Read low byte
       MOVWF  TMPL        ;
; Re-enable the Interrupt (if required)
CONTINUE                  ; Continue with your code
```

Writing a 16-bit value to the 16-bit TMR1 register is straightforward. First the TMR1L register is cleared to ensure that there are many Timer1 clock/oscillator cycles before there is a rollover into the TMR1H register. The TMR1H register is then loaded, and finally the TMR1L register is loaded. Example 14-3 shows a routine that does a 16-bit write to a Free Running Timer.

Example 14-3: Writing a 16-bit Free Running Timer

```
; All interrupts are disabled
       CLRF   TMR1L       ; Clear Low byte, Ensures no
                          ;   rollover into TMR1H
       MOVLW  HI_BYTE     ; Value to load into TMR1H
       MOVWF  TMR1H, F    ; Write High byte
       MOVLW  LO_BYTE     ; Value to load into TMR1L
       MOVWF  TMR1H, F    ; Write Low byte
; Re-enable the Interrupt (if required)
CONTINUE                  ; Continue with your code
```

14.7 Timer1 Oscillator

An alternate crystal oscillator circuit is built into the device. The output of this oscillator can be selected as the input into Timer1. The Timer1 oscillator is primarily intended to operate as a time-base for the timer modules; therefore, the oscillator is primarily intended for a 32 kHz crystal, which is an ideal frequency for real-time keeping. In real-time applications, the timer needs to increment during SLEEP, so SLEEP does not disable the Timer1 oscillator. For many applications, power consumption is also an issue, so the oscillator is designed to minimize power consumption.

The Timer1 oscillator is enabled by setting the T1OSCEN control bit (T1CON register). After the Timer1 oscillator is enabled, the user must provide a software time delay to ensure proper oscillator start-up.

Table 14-1 shows the capacitor selection for the Timer1 oscillator.

Note:	The Timer1 oscillator allows the counter to operate (increment) when the device is in sleep mode. This allows Timer1 to be used as a real-time clock.

Table 14-1: Capacitor Selection for the Timer1 Oscillator

Osc Type	Freq	C1	C2
LP	32 kHz	33 pF	33 pF
	100 kHz	15 pF	15 pF
	200 kHz	15 pF	15 pF
Crystals Tested:			
32.768 kHz	Epson C-001R32.768K-A	± 20 PPM	
100 kHz	Epson C-2 100.00 KC-P	± 20 PPM	
200 kHz	STD XTL 200.000 kHz	± 20 PPM	

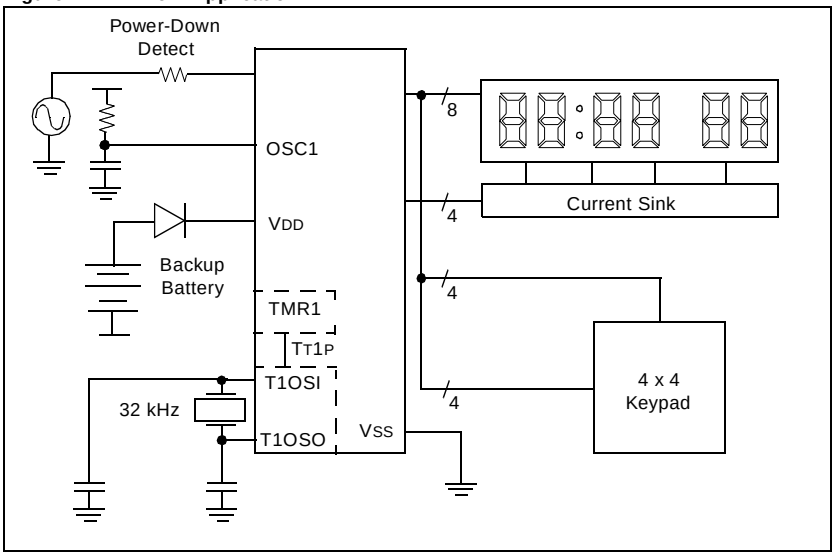
Note 1: Higher capacitance increases the stability of oscillator but also increases the start-up time.

2: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.

14.8 Typical Application

In Figure 14-4 an example application is given, where Timer1 is driven from an external 32 kHz oscillator. The external 32 kHz oscillator is typically used in applications where real-time needs to be kept, but it is also desirable to have the lowest possible power consumption. The Timer1 oscillator allows the device to be placed in sleep while the timer continues to increment. When Timer1 overflows, the interrupt wakes up the device so that the appropriate registers can be updated.

Figure 14-4: Timer1 Application



In this example, a 32 kHz crystal is used as the time base for the Real Time Clock. If the clock needs to be updated at 1 second intervals, then the Timer1 must be loaded with a value to allow the Timer1 to overflow at the desired rate. In the case of a 1 second Timer1 overflow, the TMR1H register should be loaded with a value of 80k after each overflow.

Note:	The TMR1L register should never be modified, since an external clock is asychronous to the system clock. Writes to the TRM1L register may corrupt the real time counter value causing inaccuracies.

14.9 Sleep Operation

When Timer1 is configured for asynchronous operation, the TMR1 registers will continue to increment for each timer clock (or prescale multiple of clocks). When the TMR1 register overflows, the TMR1IF bit will get set. If enabled, this will generate an interrupt that will wake the processor from sleep mode.

The Timer1 oscillator will add a delta current, due to the operation of this circuitry. That is, the power-down current will no longer only be the leakage current of the device, but also the active current of the Timer1 oscillator and other circuitry.

14.10 Resetting Timer1 Using a CCP Trigger Output

If a CCP module is configured in compare mode to generate a “Special Event Trigger” (CCP1M3:CCP1M0 = 1011), this signal resets Timer1.

Note:	The special event trigger from the CCP module does not set interrupt flag bit TMR1IF.

Timer1 must be configured for either timer or synchronized counter mode to take advantage of the special event trigger feature. If Timer1 is running in asynchronous counter mode, this reset operation may not work and should not be used.

In the event that a write to Timer1 coincides with a special event trigger from the CCP module, the write will take precedence.

In this mode of operation, the CCPRxH:CCPRxL register pair effectively becomes the period register for Timer1.

14.10.1 CCP Trigger and A/D Module

Some devices that have the CCP Trigger capability also have an A/D module. These devices may be able to be configured, so the “Special Event Trigger” not only resets the Timer1 registers, 0but will start an A/D conversion. This allows a constant sampling rate for the A/D, as specified by the value of the compare registers.

14.11 Resetting Timer1 Register Pair (TMR1H:TMR1L)

TMR1H and TMR1L registers are not cleared on any reset, only by the CCP special event triggers.

T1CON register is reset to 00h on a Power-on Reset or a Brown-out Reset. In any other reset, the register is unaffected.

Timer1 is the default time base for the CCP1 and CCP2 modules. The timer can be disabled as the time base for either CCP1, CCP2, or both, and Timer3 can be substituted. This is achieved by setting control bits in the Timer3 control register.

This is explained in Section 16.8 - Timer3 and CCPx Enable.

When Timer1 is disabled as a the time base for a CCP, the reset on Compare will have no effect on Timer1.

14.12 Timer1 Prescaler

The prescaler counter is cleared on writes to the TMR1H or TMR1L registers.

14.12.1 Timer1 Prescaler 16-bit Read/WriteMode

Writes to TMR1H do not clear the Timer1 prescaler in 16-bit read/write mode, because a write to TMR1H only modifies the Timer1 latch and does not change the contents of Timer1. The prescaler is only cleared on writes to TMR1L.

Table 14-2: Registers Associated with Timer1 as a Timer/Counter

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
PIR	TMR1IE (1)								0	0
PIE	TMR1IE (1)								0	0
IPR	TMR1IP (1)								0	0
TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 register								xxxx xxxx	uuuu uuuu
TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1 register								xxxx xxxx	uuuu uuuu
T1CON	RD16	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Legend: x = unknown,u = unchanged,- = unimplemented read as '0'.
Shaded cells are not used by the Timer1 module.

Note 1: The placement of this bit is device dependent.

14.13 Initialization

Since Timer1 has a software programmable clock source, there are three examples to show the initialization of each mode. Example 14-4 shows the initialization for the internal clock source, Example 14-5 shows the initialization for the external clock source, and Example 14-6 shows the initialization of the external oscillator mode.

Example 14-4: Timer1 Initialization (Internal Clock Source)

```
    CLRF   T1CON         ; Stop Timer1, Internal Clock Source,
                         ;   T1 oscillator disabled,
                         ;   prescaler = 1:1
    CLRF   TMR1H         ; Clear Timer1 High byte register
    CLRF   TMR1L         ; Clear Timer1 Low byte register
    CLRF   INTCON        ; Disable interrupts
    CLRF   PIE1          ; Disable peripheral interrupts
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x30          ; Internal Clock source
                         ;   with 1:8 prescaler
    MOVWF  T1CON         ;   Timer1 is stopped and
                         ;      T1 osc is disabled
    BSF    T1CON, TMR1ON ; Timer1 starts to increment
;
; The Timer1 interrupt is disabled, do polling on the overflow bit
;
T1_OVFL_WAIT
    BTFSS  PIR1, TMR1IF
    GOTO   T1_OVFL_WAIT
;
; Timer has overflowed
;
    BCF    PIR1, TMR1IF
```

Example 14-5: Timer1 Initialization (External Clock Source)

```
    CLRF   T1CON         ; Stop Timer1, Internal Clock Source,
                         ;   T1 oscillator disabled,
                         ;   prescaler = 1:1
    CLRF   TMR1H         ; Clear Timer1 High byte register
    CLRF   TMR1L         ; Clear Timer1 Low byte register
    CLRF   INTCON        ; Disable interrupts
    CLRF   PIE1          ; Disable peripheral interrupts
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x32          ; External Clock source
                         ;   with 1:8 prescaler
    MOVWF  T1CON         ;   Clock source is
                         ;     synchronized to device
                         ;   Timer1 is stopped
                         ;     and T1 osc is disabled
    BSF    T1CON, TMR1ON ; Timer1 starts to increment
;
; The Timer1 interrupt is disabled, do polling on the overflow bit
;
T1_OVFL_WAIT
    BTFSS  PIR1, TMR1IF
    GOTO   T1_OVFL_WAIT
;
; Timer has overflowed
;
    BCF    PIR1, TMR1IF
```

Example 14-6: Timer1 Initialization (External Oscillator Clock Source)

```
    CLRF   T1CON         ; Stop Timer1, Internal Clock Source,
                         ;   T1 oscillator disabled,
                         ;   prescaler = 1:1
    CLRF   TMR1H         ; Clear Timer1 High byte register
    CLRF   TMR1L         ; Clear Timer1 Low byte register
    CLRF   INTCON        ; Disable interrupts
    CLRF   PIE1          ; Disable peripheral interrupts
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x3E          ; External Clock source
                         ;   with oscillator
    MOVWF  T1CON         ;   circuitry, 1:8 prescaler,
                         ;   Clock source is
                         ;   asynchronous to device
                         ;   Timer1 is stopped
    BSF    T1CON, TMR1ON ; Timer1 starts to increment
;
; The Timer1 interrupt is disabled, do polling on the overflow bit
;
T1_OVFL_WAIT
    BTFSS  PIR1, TMR1IF
    GOTO   T1_OVFL_WAIT
;
; Timer has overflowed
;
    BCF    PIR1, TMR1IF
```


14.14 Design Tips

Question 1: ***Timer1 does not seem to be keeping accurate time.***

Answer 1:

There are a few reasons that this could occur:

1. You should never write to Timer1 where that could cause the loss of time. In most cases, that means you should not write to the TMR1L register, but if the conditions are OK, you may write to the TMR1H register. Normally, you write to the TMR1H register if you want the Timer1 overflow interrupt to be sooner than the full 16-bit time-out.
2. You should ensure that your layout uses good PCB layout techniques so noise does not couple onto the Timer1/Timer3 oscillator lines.

14.15 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is they may be written for the Baseline, the Midrange, or High-end families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to Timer1 are:

Title	Application Note #
Using Timer1 in Asynchronous Clock Mode	AN580
Low Power Real Time Clock	AN582
Yet another Clock using the PIC16C92X	AN649

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

14.16 Revision History

Revision A

This is the initial released revision of the Timer1 module description.

Section 15. Timer2

HIGHLIGHTS

This section of the manual contains the following major topics:

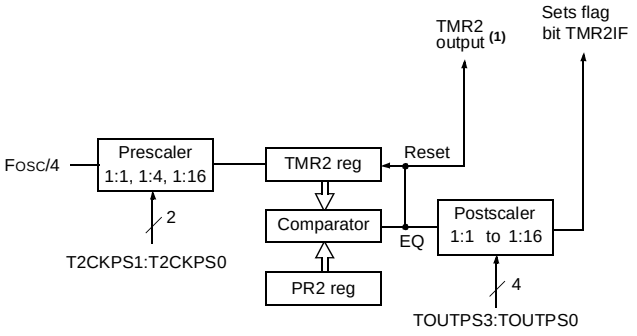
15.1 Introduction

Timer2 is an 8-bit timer with a prescaler, a postscaler and a period register. Using the prescaler and postscaler at their maximum settings, the overflow time is the same as a 16-bit timer.

Timer2 is the PWM time-base when the CCP module(s) is used in the PWM mode.

Figure 15-1 shows a block diagram of Timer2. The postscaler counts the number of times that the TMR2 register matched the PR2 register. This can be useful in reducing the overhead of the interrupt service routine on the CPU performance.

Figure 15-1: Timer2 Block Diagram



Note 1: TMR2 register output can be software selected by the SSP Module as a baud clock.

15.2 Control Register

Register 15-1 shows the Timer2 control register. The prescaler and postscaler selection of Timer2 are controlled by this register.

Register 15-1: T2CON: Timer2 Control Register

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits

0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON**: Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

15.3 Timer Clock Source

The Timer2 module has one source of input clock, the device clock (FOSC/4). A prescale option of 1:1, 1:4 or 1:16 is software selected by control bits T2CKPS1:T2CKPS0 (T2CON register).

15.4 Timer (TMR2) and Period (PR2) Registers

The TMR2 register is readable and writable, and is cleared on all device resets. Timer2 increments from 00h until it matches PR2 and then resets to 00h on the next increment cycle. PR2 is a readable and writable register.

The TMR2 register is cleared and the PR2 register is set when a WDT, POR, $\overline{\text{MCLR}}$ or a BOR reset occurs.

Timer2 can be shut off (disabled from incrementing) by clearing the TMR2ON control bit (T2CON register). This minimizes the power consumption of the module.

Note: If the PR2 register = 00h, the TMR2 register will not increment (Timer2 cleared).

15.5 TMR2 Match Output

The match output of TMR2 goes to two sources:

1. Timer2 Postscaler
2. SSP Clock Input

There are 4-bits which select the postscaler. This allows the postscaler a 1:1 to 1:16 scaling (inclusive). After the postscaler overflows, the TMR2 interrupt flag bit (TMR2IF) is set to indicate the Timer2 overflow. This is useful in reducing the software overhead of the Timer2 interrupt service routine, since it will only execute once every postscaler # of matches.

The match output of TMR2 is also routed to the Synchronous Serial Port module, which may select this via software, as the clock source for the shift clock.

15.6 Clearing the Timer2 Prescaler and Postscaler

The prescaler and postscaler counters are cleared when any of the following occurs:

- a write to the TMR2 register
- a write to the T2CON register

Note: When T2CON is written, TMR2 does not clear.

- any device reset (Power-on Reset, $\overline{\text{MCLR}}$ reset, Watchdog Timer Reset, Brown-out Reset)

15.7 Sleep Operation

During sleep, TMR2 will not increment. The prescaler will retain the last prescale count, ready for operation to resume after the device wakes from sleep.

Table 15-1: Registers Associated with Timer2

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
PIR	TMR2IF (1)								0	0
PIE	TMR2IE (1)								0	0
IPR	TMR21P (1)								0	0
TMR2	Timer2 module's register								0000 0000	0000 0000
T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
PR2	Timer2 Period Register								1111 1111	1111 1111

Legend: x = unknown,u = unchanged,- = unimplemented read as '0'.
Shaded cells are not used by the Timer2 module.

Note 1: The position of this bit is device dependent.

15.8 Initialization

Example 15-1 shows how to initialize the Timer2 module, including specifying the Timer2 prescaler and postscaler.

Example 15-1:Timer2 Initialization

```
     CLRF   T2CON         ; Stop Timer2, Prescaler = 1:1,
                          ;   Postscaler = 1:1
     CLRF   TMR2          ; Clear Timer2 register
     CLRF   INTCON        ; Disable interrupts
     LRF    PIE1          ; Disable peripheral interrupts
     CLRF   PIR1          ; Clear peripheral interrupts Flags
     MOVLW  0x72          ; Postscaler = 1:15, Prescaler = 1:16
     MOVWF  T2CON         ;   Timer2 is off
     MOVLW  PR2VALUE      ; This is the value
     MOVWF  PR2           ;   to load into the PR2 register.
     BSF    T2CON, TMR2ON ; Timer2 starts to increment
;
; The Timer2 interrupt is disabled, do polling on the overflow bit
;
T2_OVFL_WAIT
       BTFSS  PIR1, TMR2IF   ; Has TMR2 interrupt occurred?
       GOTO   T2_OVFL_WAIT   ; NO, continue loop
;
; Timer has overflowed
;
       BCF    PIR1, TMR2IF   ; YES, clear flag and continue.
```

15.9 Design Tips

Question 1: ***Timer2 never seems to increment?***

Answer 1:

Ensure that the Timer2 Period register (PR2) is not 0h. This is because when a period match occurs, the TMR2 register is cleared on the next cycle so Timer2 will never increment.

15.10 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is, they may be written for the Base-Line, the Mid-Range or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the Timer2 Module are:

Title	Application Note #
Using the CCP Module	AN594
Air Flow Control using Fuzzy Logic	AN600
Adaptive Differential Pulse Code Modulation using the PIC16/17	AN643

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

15.11 Revision History

Revision A

This is the initial released revision of the TImer2 module description.

Section 16. Timer3

HIGHLIGHTS

This section of the manual contains the following major topics:

16.1 Introduction

The Timer3 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR3H and TMR3L) that are readable and writable. The TMR3 register pair (TMR3H:TMR3L) increments from 0000h to FFFFh and rolls over to 0000h. The Timer3 Interrupt, if enabled, is generated on overflow, which is latched in the TMR3IF interrupt flag bit. This interrupt can be enabled/disabled by setting/clearing the TMR3IE interrupt enable bit.

Timer3 can operate in one of three modes:

- As a synchronous timer
- As a synchronous counter
- As an asynchronous counter

Some features of Timer3 include:

- TMR3 also has an internal “reset input”, which can be generated by a CCP module.
- TMR3 has the capability to operate off an external crystal/clock.
- TMR3 is the alternate time base for capture/compare

In timer mode, Timer3 increments every instruction cycle. In counter mode, it increments on every rising edge of the external clock input.

The Timer3 increment can be enabled/disabled by setting/clearing control bit TMR3ON (T3CON register).

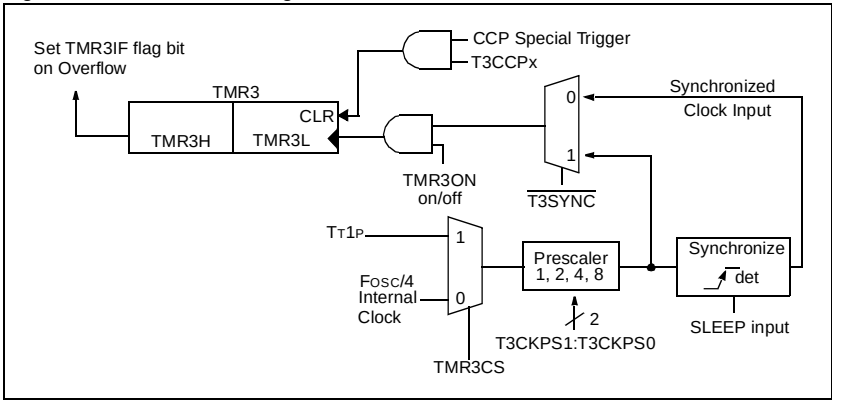
Timer3 also has an internal “reset input”. This reset can be generated by a CCP special event trigger (Capture/Compare/PWM) module. See the CCP (Capture/Compare/PWM) section for details.

When the Timer1 oscillator is enabled (T1OSCEN, in T1CON, is set), the T1OSCI1 and T1OSO2 pins are configured as oscillator input and output, so the corresponding values in the TRIS register are ignored.

The Timer3 module also has a software programmable prescaler.

The operating mode is determined by clock select bit, TMR3CS (T3CON register), and the synchronization bit, $\overline{\text{T3SYNC}}$ (Figure 16-1).

Figure 16-1: Timer3 Block Diagram



16.2 Control Registers

Register 16-1 shows the Timer3 control register. This register controls the operating mode of the Timer3 module and contains the function of the CCP Special Event Trigger.

Register 16-1: T3CON: Timer3 Control Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
bit 7							bit 0

bit 7 **RD16:** 16-bit Read/Write Mode Enable
1 = Enables register Read/Write of Timer3 in one 16-bit operation
0 = Enables register Read/Write of Timer3 in two 8-bit operations

bit 6,3 **T3CCP2:T3CCP1:** Timer3 and Timer1 to CCPx Enable bits
1x = Timer3 is the clock source for compare/capture of the CCP modules
01 = Timer3 is the clock source for compare/capture of CCP2,
Timer1 is the clock source for compare/capture of CCP1
00 = Timer1 is the clock source for compare/capture of the CCP modules

bit 5:4 **T3CKPS1:T3CKPS0**: Timer3 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 2 **$\overline{\text{T3SYNC}}$:** Timer3 External Clock Input Synchronization Control bit
(Not usable if the system clock comes from Timer1/Timer3)

When TMR3CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input

When TMR3CS = 0:
This bit is ignored. Timer3 uses the internal clock when TMR3CS = 0.

bit 1 **TMR3CS:** Timer3 Clock Source Select bit
1 = External clock input from T1OSI or T1CKI (on the rising edge after the first falling edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR3ON:** Timer3 On bit

1 = Enables Timer3
0 = Stops Timer3

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

16.3 Timer3 Operation in Timer Mode

Timer mode is selected by clearing the TMR3CS (T3CON register) bit. In this mode, the input clock to the timer is FOSC/4. The synchronize control bit, T3SYNC (T3CON register), has no effect since the internal clock is always synchronized.

16.4 Timer3 Operation in Synchronized Counter Mode

Counter mode is selected by setting bit TMR3CS. In this mode, the timer increments on every rising edge of input on the T1OSI pin (when enable bit T1OSCEN is set) or the T13CKI pin (when bit T1OSCEN is cleared).

Note:	Timer3 gets its external clock input from the same source as Timer1. The configuration of the Timer1 and Timer3 clock input will be controlled by the T1OSCEN bit in the Timer1 control register.

If the T3SYNC bit is cleared, then the external clock input is synchronized with internal phase clocks. The synchronization is done after the prescaler stage. The prescaler operates asynchronously.

The timer increments at the Q4:Q1 edge.

In this configuration, during SLEEP mode, Timer3 will not increment even if an external clock is present, since the synchronization circuit is shut off. The prescaler, however, will continue to increment.

16.4.1 External Clock Input Timing for Synchronized Counter Mode

When an external clock input is used for Timer3 in synchronized counter mode, it must meet certain requirements. The external clock requirement is due to internal phase clock (TSCLK) synchronization. Also, there is a delay in the actual incrementing of TMR3 after synchronization.

When the prescaler is 1:1, the external clock input is the same as the prescaler output. There is synchronization of T1OSI/T13CKI with the internal phase clocks. Therefore, it is necessary for (T1OSI/T13CKI to be high for at least 2TSCLK (and a small RC delay) and low for at least 2TSCLK (and a small RC relay). Refer to parameters 45, 46, and 47 in the **"Electrical Specifications"** section.

When a prescaler other than 1:1 is used, the external clock input is divided by the asynchronous prescaler, so that the prescaler output is symmetrical. In order for the external clock to meet the sampling requirement, the prescaler counter must be taken into account. Therefore, it is necessary for T1OSI/T1CKI to have a period of at least 4TSCLK (and a small RC delay) divided by the prescaler value. The only requirement on T1OSI/T1CKI high and low time is that they do not violate the minimum pulse width requirements. Refer to parameters 40, 42, 45, 46, and 47 in the **"Electrical Specifications"** section.

16.5 Timer3 Operation in Asynchronous Counter Mode

If $\overline{\text{T3SYNC}}$ (T3CON register) is set, the external clock input is not synchronized. The timer continues to increment asynchronously to the internal phase clocks. The timer will continue to run during SLEEP and can generate an interrupt on overflow that will wake-up the processor. However, special precautions in software are needed to read/write the timer (Subsection **16.6.4 "Reading and Writing Timer3 in Asynchronous Counter Mode with RD16 = 0"**). Since the counter can operate in sleep, Timer1 can be used to implement a true real-time clock.

The timer increments at the Q4:Q1 and Q2:Q3 edges.

In asynchronous counter mode, Timer3 cannot be used as a time-base for capture or compare operations.

Note:	The control bit $\overline{\text{T3SYNC}}$ is not usable when the system clock source comes from the same source as the Timer1/Timer3 clock input, because the T1CKI input will be sampled at one quarter the frequency of the incoming clock.

16.5.1 External Clock Input Timing with Unsynchronized Clock

If the $\overline{\text{T3SYNC}}$ control bit is set, the timer will increment completely asynchronously. The input clock must meet certain minimum high time and low time requirements. Refer to the Device Data Sheet **"Electrical Specifications"** section, timing parameters 45, 46, and 47.

16.6 Reading and Writing of Timer3

Timer3 has modes that allow the 16-bit timer register to be read/written as two 8-bit registers or one 16-bit register. The mode depends on the state of the RD16 bit. The follow subsections discuss this operation.

16.6.1 Timer3 and 16-bit Read/Write Modes

Timer3 can be configured for 16-bit reads. When the RD16 control bit (T3CON register) is set, the address for TMR3H is mapped to a buffer register for the high byte of Timer3. A read from TMR3L will load the contents of the high byte of Timer3 into the Timer3 high byte buffer. This provides the user with the ability to accurately read all 16-bits of Timer3 without having to determine whether a read of the high byte followed by a read of the low byte is valid due to a rollover between reads.

16.6.2 16-bit Mode Timer Write

A write to the high byte of Timer3 must also take place through the TMR3H buffer register. Timer3 high byte is updated with the contents of TMR3H when a write occurs to TMR3L. This allows a user to write all 16 bits to both the high and low bytes of Timer3 at once (See Figure 16-2).

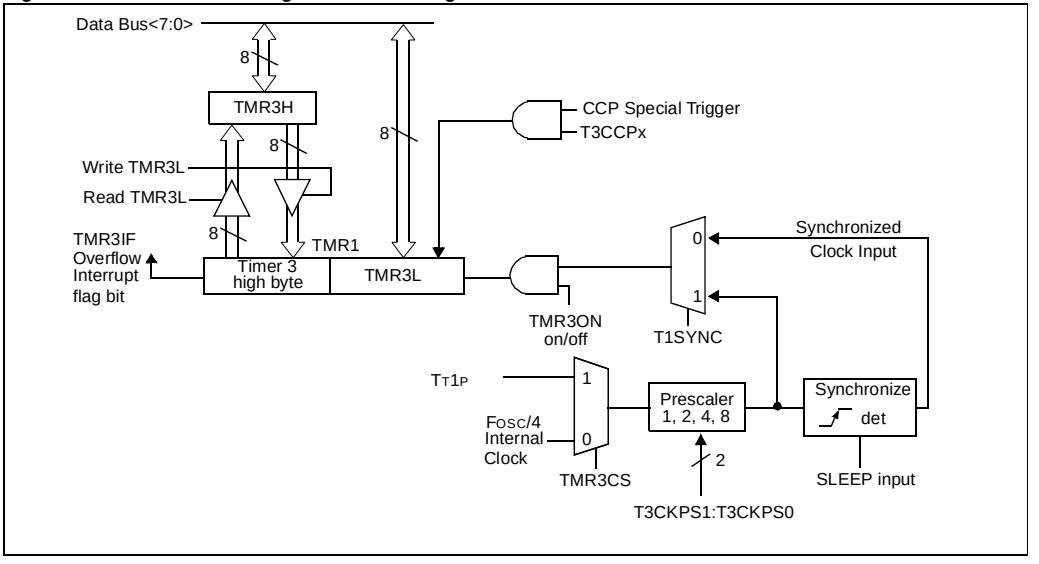
The high byte of Timer3 is not directly readable or writable in this mode. All reads and writes must take place through the Timer3 high byte buffer register.

Writes to TMR3H do not clear the Timer3 prescaler. The prescaler is only cleared on writes to TMR3L.

16.6.3 16-bit Read-Modify-Write

Read-modify-write instructions like BSF or BCF will read the contents of a register, make the appropriate changes, and place the result back into the register. In the case of Timer3 when configured in 16-bit mode, the read portion of a read-modify-write instruction of TMR3L will not update the contents of the TMR3H buffer. The TMR3H buffer will remain unchanged. When the write of TMR3L portion of the instruction takes place, the contents of TMR3H will be placed into the high byte of Timer3.

Figure 16-2: Timer3 Block Diagram When Configured in 16-bit Read/Write Mode



16.6.4 Reading and Writing Timer3 in Asynchronous Counter Mode with RD16 = 0

Reading TMR3H or TMR3L while the timer is running from an external asynchronous clock will ensure a valid read (taken care of in hardware). However, the user should keep in mind that reading the 16-bit timer in two 8-bit values poses certain problems, since the timer may overflow between the reads.

For writes, it is recommended that the user simply stop the timer and write the desired values. A write contention may occur by writing to the timer registers, while the register is incrementing. This may produce an unpredictable value in the timer register.

Reading the 16-bit value requires some care, since two separate reads are required to read the entire 16-bits. Example 16-1 shows why this may not be a straightforward read of the 16-bit register.

Example 16-1:Reading 16-bit Register Issues

TMR3	Sequence 1		Sequence 2	
	Action	TMPH:TMPL	Action	TMPH:TMPL
04FFh	READ TMR3L	xxxxh	READ TMR3H	xxxxh
0500h	Store in TMPL	xxFFh	Store in TMPH	04xxh
0501h	READ TMR3H	xxFFh	READ TMR3L	04xxh
0502h	Store in TMPH	05FFh	Store in TMPL	0401h

Example 16-2 shows a routine to read the 16-bit timer value without experiencing the issues shown in Example 16-1. This is useful if the timer cannot be stopped.

Example 16-2:Reading a 16-bit Free-Running Timer

```
; All interrupts are disabled
       MOVF   TMR3H, W    ; Read high byte
       MOVWF  TMPH        ;
       MOVF   TMR3L, W    ; Read low byte
       MOVWF  TMPL        ;
       MOVF   TMR3H, W    ; Read high byte
       SUBWF  TMPH,  W    ; Sub 1st read with 2nd read
       BTFSC  STATUS,Z    ; Is result = 0
       GOTO   CONTINUE    ; Good 16-bit read
;
; TMR3L may have rolled over between the read of the high and low bytes.
; Reading the high and low bytes now will read a good value.
;
       MOVF   TMR3H, W    ; Read high byte
       MOVWF  TMPH        ;
       MOVF   TMR3L, W    ; Read low byte
       MOVWF  TMPL        ;
; Re-enable the Interrupt (if required)
CONTINUE                  ; Continue with your code
```

Writing a 16-bit value to the 16-bit TMR3 register is straight forward. First the TMR3L register is cleared to ensure that there are many Timer3 clock/oscillator cycles before there is a rollover into the TMR3H register. The TMR3H register is then loaded, and finally, the TMR3L register is loaded. Example 16-3 shows a routine that accomplishes this:

Example 16-3:Writing a 16-bit Free Running Timer

```
; All interrupts are disabled
       CLRF   TMR3L       ; Clear Low byte, Ensures no
                          ;   rollover into TMR3H
       MOVLW  HI_BYTE     ; Value to load into TMR3H
       MOVWF  TMR3H, F    ; Write High byte
       MOVLW  LO_BYTE     ; Value to load into TMR3L
       MOVWF  TMR3H, F    ; Write Low byte
; Re-enable the Interrupt (if required)
CONTINUE                  ; Continue with your code
```

16.7 Timer3 using the Timer1 Oscillator

An alternate crystal oscillator circuit is built into the device. The output of this oscillator can be selected as the input into Timer3. The Timer1 oscillator is primarily intended to operate as a time-base for the timer modules; therefore, the oscillator is primarily intended for a 32 kHz crystal, which is an ideal frequency for real-time keeping. In real-time applications, the timer needs to increment during SLEEP, so SLEEP does not disable the Timer1 oscillator. For many applications, power consumption is also an issue, so the oscillator is designed to minimize consumption.

The Timer1 oscillator is enabled by setting the T1OSCEN control bit (T1CON register). After the Timer1 oscillator is enabled, the user must provide a software time delay to ensure proper oscillator start-up.

Table 16-1 shows the capacitor selection for the Timer1 oscillator.

Note:	The Timer1 oscillator allows the counter to operate (increment) when the device is in sleep mode. This allows Timer1 to be used as a real-time clock.

Table 16-1: Capacitor Selection for the Timer1 oscillator

Osc Type	Freq	C1	C2
LP	32 kHz	33 pF	33 pF
	100 kHz	15 pF	15 pF
	200 kHz	15 pF	15 pF
Crystals Tested:			
32.768 kHz	Epson C-001R32.768K-A	± 20 PPM	
100 kHz	Epson C-2 100.00 KC-P	± 20 PPM	
200 kHz	STD XTL 200.000 kHz	± 20 PPM	

Note 1: Higher capacitance increases the stability of oscillator, but also increases the start-up time.

2: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.

16.8 Timer3 and CCPx Enable

Timer3 can be configured as the time base for capture and compare for either CCP2 or both CCP1 and CCP2. Timer3 cannot be used as the time base for CCP1 only. Timer1 can be used as the time base for CCP1 or both CCP1 and CCP2, but not CCP2 only. Control for the assignment of each of the time bases is given by configuring the corresponding T3CCP2 and T3CCP1 bits in the Timer3 control register, and is described in Table 16-2.

Table 16-2: T3CCPx, TMR1, and TMR3

T3CCP2:T3CCP1	Time base for CCP1	Time base for CCP2
00	TMR1	TMR1
01	TMR1	TMR3
10	TMR3	TMR3
11	TMR3	TMR3

After reset, Timer1 defaults as the time base for compare and capture for both CCP's.

16.8.1 Resetting Timer3 Using a CCP Trigger Output

If the T3CCP2 or T3CCP1 bit is set and CCP1 or CCP2 is configured in Compare mode to generate a "Special Event Trigger" (CCPxM3:CCPxM0 = 1011), this signal will reset Timer3.

Note: The "Special Event Trigger" from the CCP1 and CCP2 modules will not set interrupt flag bit TMR3IF.

Timer3 must be configured for either timer or synchronized counter mode to take advantage of this feature. If Timer3 is running in asynchronous counter mode, this reset operation may not work.

In the event that a write to Timer3 coincides with a special event trigger from a CCP module, the write will take precedence.

In this mode of operation, the CCPRxH:CCPRxL register pair effectively becomes the period register for the Timer3 module.

16.8.2 Resetting of TMR3 Register Pair (TMR3H:TMR3L)

TMR3H and TMR3L registers are not cleared on any reset, only by the CCP special event triggers.

T1CON register is reset to 00h on a Power-on Reset or a Brown-out Reset. In any other reset, the register is unaffected.

Timer3 is the default time base for the CCP1 and CCP2 modules. The timer can be disabled as the time base for either CCP1, CCP2, or both, and Timer3 can be substituted. This is achieved by setting control bits in the Timer3 control register.

This is explained in Section 16.8 - Timer3 and CCPx Enable.

When Timer3 is disabled as a the time base for a CCP, the reset on compare for that particular CCP will have no effect on Timer3.

16.9 Timer3 Prescaler

The prescaler counter is cleared on writes to the TMR3H or TMR3L registers.

16.9.1 Timer3 Prescaler 16-bit Read/Write Mode

Writes to TMR3H do not clear the Timer3 prescaler in 16-bit read/write mode, because a write to TMR3H only modifies the Timer3 latch and does not change the contents of Timer3. The prescaler is only cleared on writes to TMR3L.

16.10 16-bit Mode Timer Reads/Writes

Timer3 has modes that allow the 16-bit time register to be read as two 8-bit registers or one 16-bit register depending on the state of the RD16 bit.

When the RD16 control bit is set, the address for TMR3H is mapped to a buffer register for the high byte of Timer3. A read from TMR3L will load the contents of the high byte of Timer3 into the Timer3 high byte buffer. This provides the user with the ability to accurately read all 16 bits of Timer3 without having to determine whether a read of the high byte followed by a read of the low byte is valid due to a rollover between reads.

16.10.1 16-bit Mode Timer Write

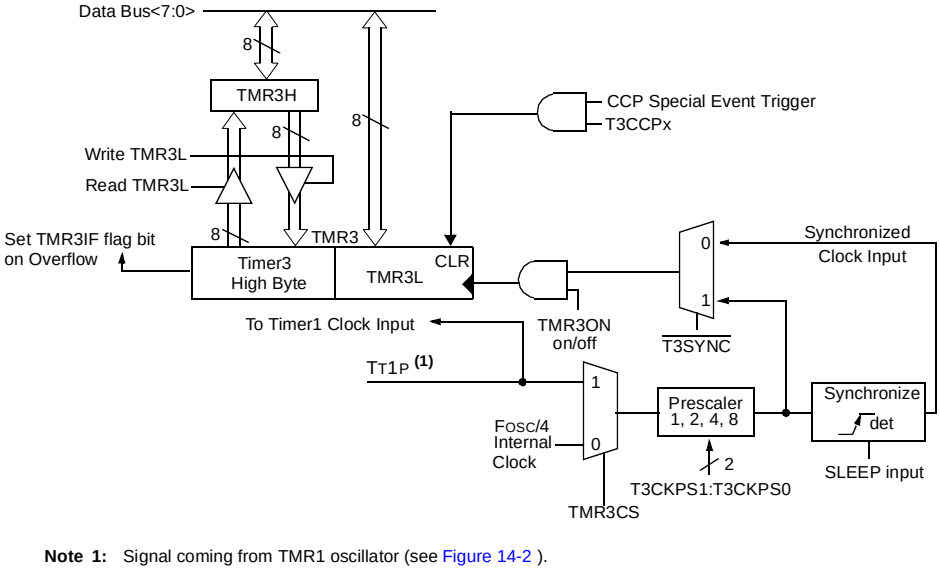
A write to the high byte of Timer3 must also take place through the TMR3H buffer register. Timer3 high byte is updated with the contents of TMR3H when a write occurs to TMR3L. This allows a user to write all 16 bits to both the high and low bytes of Timer3 at once (See Figure 16-3).

The high byte of Timer3 is not directly readable or writable in this mode. All reads and writes must take place through the Timer3 high byte buffer register.

16.10.2 16-bit Read/Modify Write

Read modify write instructions like BSF or BCF will read the contents of a register, make the appropriate changes, and place the result back into the register. In the case of Timer3 when configured in 16-bit Read/Write mode, the read portion of a read-modify-write instruction of TMR3L will not update the contents of the TMR3H buffer. The TMR3H buffer will remain unchanged. When the write of TMR3L portion of the instruction takes place, the contents of TMR3H will be placed into the high byte of Timer3.

Figure 16-3: Timer3 Block Diagram Configured in 16-bit Read/Write Mode

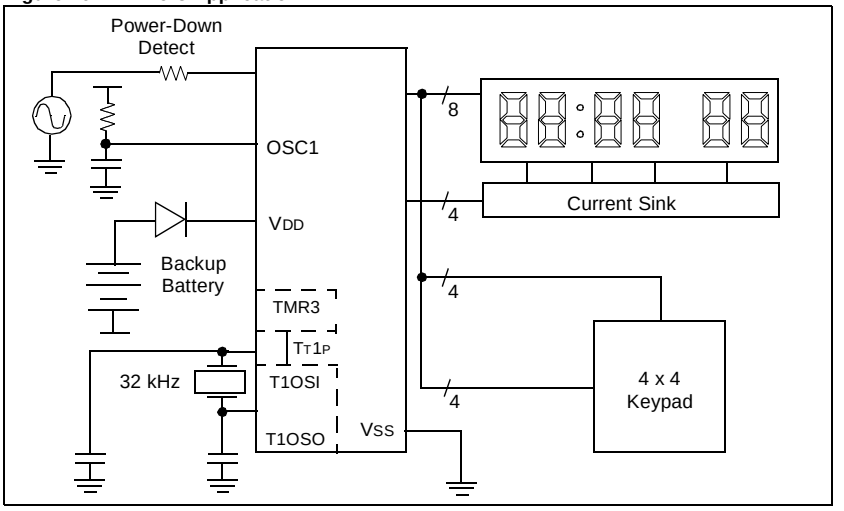


Note 1: Signal coming from TMR1 oscillator (see Figure 14-2).

16.11 Typical Application

The external oscillator clock input feature is typically used in applications where real-time needs to be kept, but it is also desirable to have the lowest possible power consumption. The Timer1 oscillator allows the device to be placed in sleep, while the timer continues to increment. When Timer3 overflows the interrupt wakes up the device so that the appropriate registers can be updated.

Figure 16-4: Timer3 Application



In this example, a 32kHz crystal is used as the time base for the Real Time Clock. If the clock needs to be updated at 1 second intervals, then the Timer1 must be loaded with a value to allow the Timer1 to overflow at the desired rate. In the case of a 1 second Timer1 overflow, the TMR1H register should be loaded with a value of 80k after each overflow.

Note:	The TMR3L register should never be modified, since an external clock is asychronous to the system clock. Writes to the TRM3L register may corrupt the real time counter value causing inaccuracies.

16.12 Sleep Operation

When Timer3 is configured for asynchronous operation, the TMR3 registers will continue to increment for each timer clock (or prescale multiple of clocks). When the TMR3 register overflows, the TMR3IF bit will get set, and if enabled, generate an interrupt that will wake the processor from sleep mode.

The Timer1 oscillator will add a delta current, due to the operation of this circuitry. That is, the power-down current will no longer only be the leakage current of the device, but also the active current of the Timer1 oscillator and other Timer1 circuitry.

16.13 Timer3 Prescaler

The prescaler counter is cleared on writes to the TMR3H or TMR3L registers.

Table 16-3: Registers Associated with Timer3 as a Timer/Counter

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
PIR	TMR3IF (1)								0	0
PIE	TMR3IE (1)								0	0
IPR	TMR3IP (1)								0	0
TMR3L	Holding register for the Least Significant Byte of the 16-bit TMR3 register								xxxx xxxx	uuuu uuuu
TMR3H	Holding register for the Most Significant Byte of the 16-bit TMR3 register								xxxx xxxx	uuuu uuuu
T3CON	RD16		T3CKPSI	T3CKPS0	T30SCEN	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON	--00 0000	--uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'.
Shaded cells are not used by the Timer1 module.

Note 1: The placement of this bit is device dependent.

16.14 Initialization

Since Timer3 has a software programmable clock source, there are three examples to show the initialization of each mode. Example 16-4 shows the initialization for the internal clock source, Example 16-5 shows the initialization for the external clock source, and Example 16-6 shows the initialization of the external oscillator mode.

Example 16-4:Timer3 Initialization (Internal Clock Source)

```
    CLRF   T3CON         ; Stop Timer3, Internal Clock Source,
                         ;   T1 oscillator disabled,
                         ;   prescaler = 1:1
    CLRF   TMR3H         ; Clear Timer3 High byte register
    CLRF   TMR3L         ; Clear Timer3 Low byte register
    CLRF   INTCON        ; Disable interrupts
    CLRF   PIE1          ; Disable peripheral interrupts
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x30          ; Internal Clock source
                         ;   with 1:8 prescaler
    MOVWF  T3CON         ;   Timer3 is stopped and
                         ;   T1 osc is disabled
    BSF    T3CON, TMR3ON ; Timer3 starts to increment
;
; The Timer3 interrupt is disabled, do polling on the overflow bit
;
T3_OVFL_WAIT
    BTFSS  PIR1, TMR3IF
    GOTO   T3_OVFL_WAIT
;
; Timer has overflowed
;
    BCF    PIR1, TMR3IF
```

Example 16-5:Timer3 Initialization (External Clock Source)

```
    CLRF   T3CON         ; Stop Timer3, Internal Clock Source,
                         ;   T1 oscillator disabled,
                         ;   prescaler = 1:1
    CLRF   TMR3H         ; Clear Timer3 High byte register
    CLRF   TMR3L         ; Clear Timer3 Low byte register
    CLRF   INTCON        ; Disable interrupts
    CLRF   PIE1          ; Disable peripheral interrupts
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x32          ; External Clock source
                         ;   with 1:8 prescaler
    MOVWF  T3CON         ;   Clock source is
                         ;   synchronized to device
                         ;   Timer3 is stopped and
                         ;     T1 osc is disabled
    BSF    T3CON, TMR3ON ; Timer3 starts to increment
;
; The Timer3 interrupt is disabled, do polling on the overflow bit
;
T3_OVFL_WAIT
    BTFSS  PIR1, TMR3IF
    GOTO   T3_OVFL_WAIT
;
; Timer has overflowed
;
    BCF    PIR1, TMR3IF
```

Example 16-6:Timer3 Initialization (External Oscillator Clock Source)

```
    CLRF   T3CON         ; Stop Timer3, Internal Clock Source,
                         ;   T1 oscillator disabled,
                         ;   prescaler = 1:1
    CLRF   TMR3H         ; Clear Timer3 High byte register
    CLRF   TMR3L         ; Clear Timer3 Low byte register
    CLRF   INTCON        ; Disable interrupts
    CLRF   PIE1          ; Disable peripheral interrupts
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x3E          ; External Clock source
                         ;   with oscillator
    MOVWF  T3CON         ;   circuitry, 1:8 prescaler,
                         ;   Clock source is
                         ;   asynchronous to device
                         ;   Timer3 is stopped
    BSF    T3CON, TMR3ON ; Timer3 starts to increment
;
; The Timer3 interrupt is disabled, do polling on the overflow bit
;
T3_OVFL_WAIT
    BTFSS  PIR1, TMR3IF
    GOTO   T3_OVFL_WAIT
;
; Timer has overflowed
;
    BCF    PIR1, TMR3IF
```

16.15 Design Tips

Question 1: ***Timer3 does not seem to be keeping accurate time.***

Answer 1:

There are a few reasons that this could occur:

1. You should never write to Timer3, where that could cause the loss of time. In most cases, that means you should not write to the TMR3L register, but if the conditions are ok, you may write to the TMR3H register. Normally you write to the TMR3H register if you want the Timer3 overflow interrupt to be sooner then the full 16-bit time-out.
2. You should ensure the your layout uses good PCB layout techniques so that noise does not couple onto the Timer1/Timer3 oscillator lines.

16.16 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhance family (that is they may be written for the Baseline, the Midrange, or High-end families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to Timer1 are:

Title	Application Note #
Using Timer1 in Asynchronous Clock Mode	AN580
Low Power Real Time Clock	AN582
Yet another Clock using the PIC16C92X	AN649

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

16.17 Revision History

Revision A

This is the initial released revision of the Timer3 module description.

Section 17. Compare/Capture/PWM (CCP)

HIGHLIGHTS

This section of the manual contains the following major topics:

17.1 Introduction

Each CCP (Capture/Compare/PWM) module has three 8-bit registers. These are:

- An 8-bit control register (CCPxCON)
- A 16-bit register (CCPRxH:CCPRxL) that operates as:
  - a 16-bit capture register
  - a 16-bit compare register
  - a 10-bit PWM master/slave duty cycle register

Multiple CCP modules may exist on a single device. The CCP modules are identical in operation, with the exception of the operation of the special event trigger.

Throughout this section, we use generic names for the CCP registers. These generic names are shown in Table 17-1.

Table 17-1: Specific to Generic CCP Nomenclature

Generic Name	CCP1	CCP2	Comment
CCPxCON	CCP1CON	CCP2CON	CCP Control Register
CCPRxH	CCPR1H	CCPR2H	CCP high byte
CCPRxL	CCPR1L	CCPR2L	CCP low byte
CCPx	CCP1	CCP2	CCP pin

17.1.1 Timer Resources

Table 17-2 shows the resources of the CCP modules, in each of its modes. Table 17-3 shows the interactions between the CCP modules, where CCPx is one CCP module and CCPy is another CCP module.

Table 17-2: CCP Mode - Timer Resource

CCP Mode	Timer Resource
Capture	Timer1 or Timer3
Compare	Timer1 or Timer3
PWM	Timer2

Table 17-3: Interaction of Two CCP Modules

CCPx Mode	CCPy Mode	Interaction
Capture	Capture	TMR1 or TMR3 time-base. Time base can be different for each CCP.
Capture	Compare	The compare could be configured for the special event trigger, which clears either TMR1 or TMR3 depending upon which time base is used.
Compare	Compare	The compare(s) could be configured for the special event trigger, which clears TMR1 or TMR3 depending upon which time base is used.
PWM	PWM	The PWMs will have the same frequency, and update rate (TMR2 interrupt).
PWM	Capture	None
PWM	Compare	None

17.2 CCP Control Register

Register 17-1 shows the CCP Control Register. This register selects the mode of operation of the CCP module, as well as contains the 2-LSb of the PWM Duty Cycle.

Register 17-1: CCPxCON Register

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **DCxB<1:0>**: PWM Duty Cycle bit1 and bit0

Capture Mode:

Unused

Compare Mode:

Unused

PWM Mode:

These bits are the two LSbs (bit1 and bit0) of the 10-bit PWM duty cycle. The upper eight bits (DCx<9:2>) of the duty cycle are found in CCPRxL.

bit 3-0 **CCPxM<3:0>**: CCPx Mode Select bits

0000 = Capture/Compare/PWM off (resets CCPx module)
0001 = Reserved
0010 = Compare mode, toggle output on match (CCPxIF bit is set)
0011 = Reserved
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode,
Initialize CCP pin Low, on compare match force CCP pin High (CCPIF bit is set)
1001 = Compare mode,
Initialize CCP pin High, on compare match force CCP pin Low (CCPIF bit is set)
1010 = Compare mode,
Generate software interrupt on compare match
(CCPIF bit is set, CCP pin is unaffected)
1011 = Compare mode,
Trigger special event (CCPIF bit is set)
11xx = PWM mode

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

17.3 Capture Mode

In Capture mode, CCPRxH:CCPRxL captures the 16-bit value of the TMR1 or TMR3 register when an event occurs on the CCPx pin. An event is defined as:

- Every falling edge
- Every rising edge
- Every 4th rising edge
- Every 16th rising edge

An event is selected by control bits CCPxM3:CCPxM0 (CCPxCON<3:0>). When a capture is made, the interrupt request flag bit, CCPxIF, is set. The CCPxIF bit must be cleared in software. If another capture occurs before the value in register CCPRx is read, the old captured value will be lost.

Note:	The dedicated time base (Timer1 or Timer3) must be running in Timer mode or Synchronized Counter mode for the CCP module to use the capture feature. In Asynchronous Counter mode, the capture operation may not work.

When the Capture mode is changed, a false capture interrupt may be generated. The user should keep bit CCPxIE clear to avoid false interrupts and should clear flag bit CCPxIF following any such change in operating mode.

Figure 17-1 shows that a capture does not modify (clear) the 16-bit timer register. This is so the timer (Timer1 or Timer3) can also be used as the time-base for other operations.

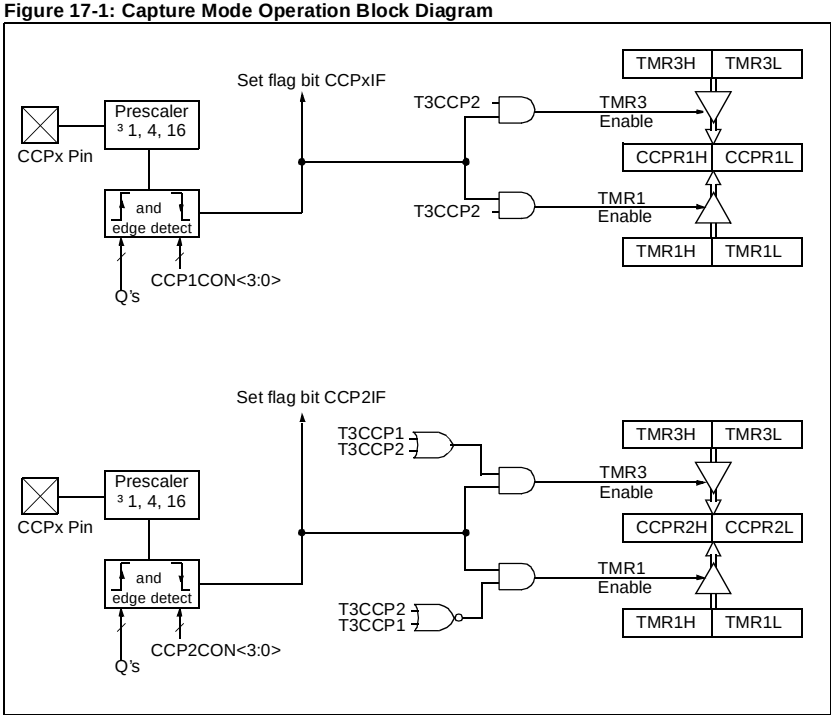
The time between two captures can easily be computed as the difference between the value of the 2nd capture and that of the 1st capture. When the timer overflows, the timer interrupt bit, TMRxIF will be set. If enabled, an interrupt will occur, allowing the time-base to be extended to greater than 16 bits.

17.3.1 CCP Pin Configuration

In Capture mode, the CCPx pin should be configured as an input by setting its corresponding TRIS bit. The prescaler can be used to get a very fine average resolution on a constant input frequency. For example, if we have a stable input frequency and we set the prescaler to 1:16, then the total error for those 16 periods is 1 TCY. This gives an effective resolution of TCY/16, which at 40 MHz is 6.25 ns. This technique is only valid where the input frequency is "stable" over the 16 samples. Without using the prescaler (1:1), each sample would have a resolution of TCY.

Note: If the CCPx pin is configured as an output, a write to the port can cause a capture condition.

Figure 17-1: Capture Mode Operation Block Diagram



17.3.2 Changing Between Capture Modes

When the Capture mode is changed, a capture interrupt may be generated. The user should keep the CCPxIE bit clear to disable these interrupts and should clear the CCPxIF flag bit following any such change in operating mode.

17.3.2.1 CCP Prescaler

There are four prescaler settings, specified by the CCPxM3:CCPxM0 bits. Whenever the CCP module is turned off, or the CCP module is not in capture mode, the prescaler counter is cleared. This means that any reset will clear the prescaler counter.

Switching from one capture prescale setting to another may generate an interrupt. Also, the prescaler counter will not be cleared, therefore the first capture may be from a non-zero prescaler. Example 17-1 shows the recommended method for switching between capture prescale settings. This example uses CCP1 and clears the prescaler counter so not to generate an unintended interrupt.

Example 17-1:Changing Between Capture Prescalers

```
CLRF    CCP1CON         ; Turn CCP module off
MOVLW   NEW_CAPT_PS     ; Load the W reg with the new prescaler
                        ;   mode value and CCP ON
MOVWF   CCP1CON         ; Load CCP1CON with this value
```

To clear the Capture prescaler count, the CCP module must be configured into any non-capture CCP mode (Compare, PWM, or CCP off modes).

17.3.3 Sleep Operation

When the device is placed in SLEEP, the timer will not increment (since it is in synchronous mode), but the prescaler will continue to count events (not synchronized). When a specified capture event occurs, the CCPxIF bit will be set, but the capture register will not be updated. If the CCP interrupt is enabled, the device will wake-up from SLEEP. The value in the 16-bit TMR1 register is **not** transferred to the 16-bit capture register. Effectively, this allows the CCP pin to be used as another external interrupt.

17.3.4 Effects of a Reset

The CCP module is off, and the value in the capture prescaler is cleared.

17.4 Compare Mode

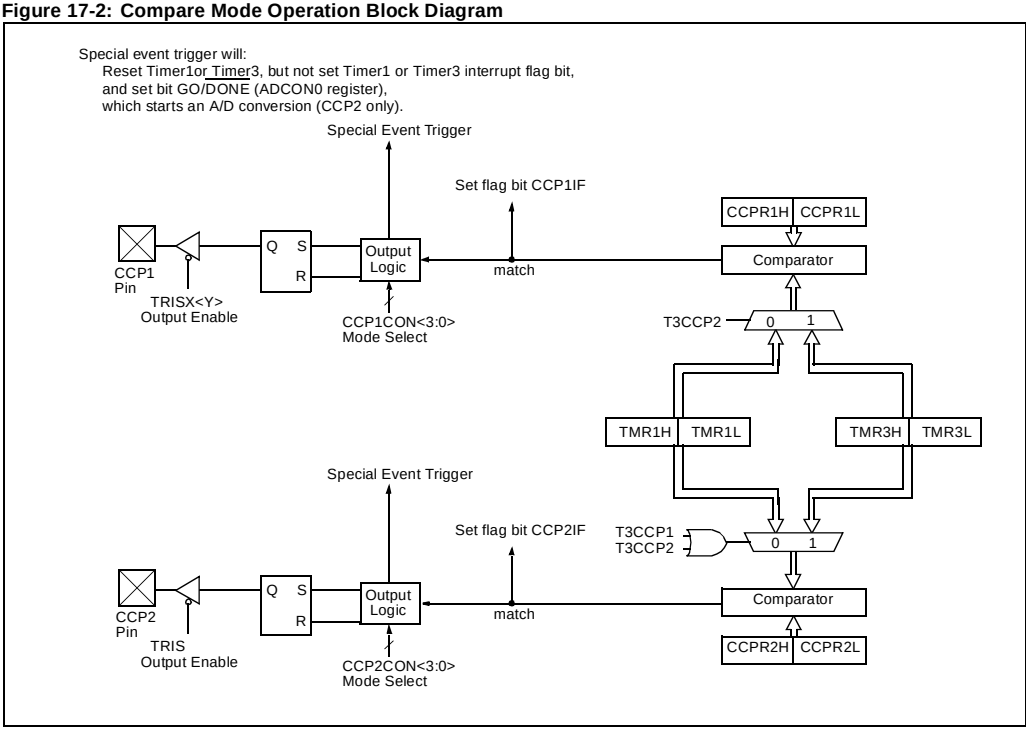
In Compare mode, the 16-bit CCPRx register value is constantly compared against the TMR1 or TMR3 register pair value. When a match occurs, the CCPx pin is:

- Driven High
- Driven Low
- Toggle output (Low to High or High to Low)
- Not affected (remains unchanged) and configured as I/O pin

The action on the pin is based on the value of control bits CCPxM3:CCPxM0 (CCPxCON3:CCPxCON0). At the same time, interrupt flag bit CCPxIF is set.

Note: The dedicated time base (Timer1 or Timer3) must be running in Timer mode or Synchronized Counter mode if the CCP module is using the compare feature. In Asynchronous Counter mode, the compare operation may not work.

Figure 17-2: Compare Mode Operation Block Diagram



17.4.1 CCP Pin Operation in Compare Mode

The user must configure the CCPx pin as an output by clearing the appropriate TRIS bit.

Note:	Clearing the CCPxCON register will force the CCPx compare output latch to the default low level. This is not the Port I/O data latch.

Selecting the compare output mode, forces the state of the CCP pin to the state that is opposite of the match state. So if the Compare mode is selected to force the output pin low on match, then the output will be forced high until the match occurs (or the mode is changed). In the compare toggle mode, the CCPx pin output is initially forced to the low state.

17.4.2 Software Interrupt Mode

When Generate Software Interrupt mode is chosen, the CCPx pin is not affected. Only a CCP interrupt is generated (if enabled).

17.4.3 Special Event Trigger

In this mode, an internal hardware trigger is generated that may be used to initiate an action.

The special event trigger output of CCPx resets the assigned timer register pair (TMR1 or TMR3 depending upon the state of the T3CCPx bits). This allows the CCPRxH:CCPRxL registers to effectively be a 16-bit programmable period register for the timer (Timer1 or Timer3).

For some devices, the special trigger output of the CCP module resets the timer (TMR1 or TMR3) register pair (depending upon the state of the T3CCPx bits), and starts an A/D conversion (if the A/D module is enabled).

Note:	The special event trigger will not set the Timers interrupt flag bit, TMRxIF.

17.4.4 Sleep Operation

When the device is placed in SLEEP, the timer will not increment (since it is in Synchronous mode), and the state of the module will not change. If the CCP pin is driving a value, it will continue to drive that value. When the device wakes-up, it will continue from this state.

17.4.5 Effects of a Reset

The CCP module is off.

Table 17-4: Registers Associated with Capture, Compare, Timer1 and Timer3

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other resets
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1	PSPIF (1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE (1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP (1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	0000 0000	0000 0000
TRISC	PORTC Data Direction Register								1111 1111	1111 1111
TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 register								xxxx xxxx	uuuu uuuu
TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1register								xxxx xxxx	uuuu uuuu
T1CON	RD16	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu
CCPR1L	Capture/Compare/PWM register1 (LSB)								xxxx xxxx	uuuu uuuu
CCPR1H	Capture/Compare/PWM register1 (MSB)								xxxx xxxx	uuuu uuuu
CCP1CON	—	—	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
CCPR2L	Capture/Compare/PWM register2 (LSB)								xxxx xxxx	uuuu uuuu
CCPR2H	Capture/Compare/PWM register2 (MSB)								xxxx xxxx	uuuu uuuu
CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000
PIR2	—	—	—	—	BCLIF	LVDIF	TMR3IF	CCP2IF	0000 0000	0000 0000
PIE2	—	—	—	—	BCLIE	LVDIE	TMR3IE	CCP2IE	0000 0000	0000 0000
IPR2	—	—	—	—	BCLIP	LVDIP	TMR3IP	CCP2IP	0000 0000	0000 0000
TMR3L	Holding register for the Least Significant Byte of the 16-bit TMR3 register								xxxx xxxx	uuuu uuuu
TMR3H	Holding register for the Most Significant Byte of the 16-bit TMR3 register								xxxx xxxx	uuuu uuuu
T3CON	RD16	T3CCP2	T3CKPS1	T3CKPS0	T3CCP1	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON	-000 0000	-uuu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'.
Shaded cells are not used by Capture, Compare, Timer1 and Timer3.

Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18C2x2 devices. Always maintain these bits clear.

17.5 PWM Mode

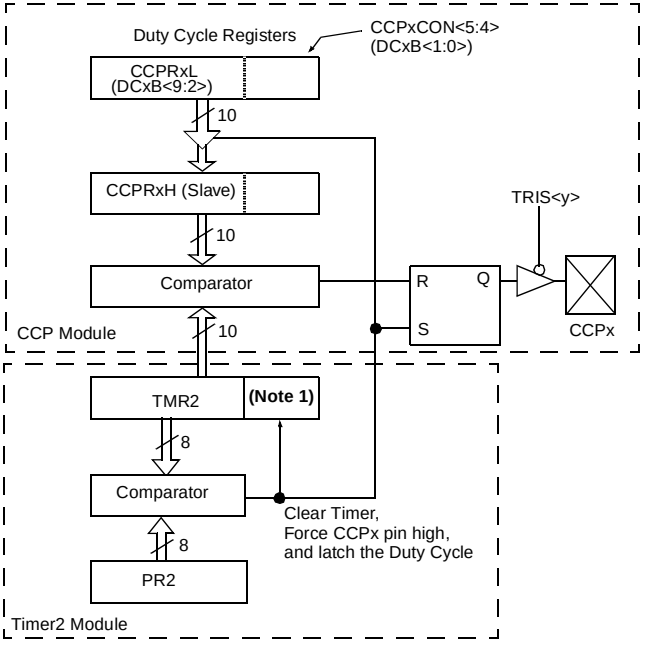
In Pulse Width Modulation (PWM) mode, the CCPx pin produces up to a 10-bit resolution PWM output. Since the CCPx pin is multiplexed with the PORT data latch, the corresponding TRIS bit must be cleared to make the CCPx pin an output.

Note:	Clearing the CCPxCON register will force the CCPx PWM output latch to the default low level. This is not the Port I/O data latch.

Figure 17-3 shows a simplified block diagram of one CCP module in PWM mode. Depending on the device there can be more than one CCP module connected to Timer2. Each CCP module can support one Pulse Width Modulation (PWM) output signal. This PWM signal can attain a resolution of up to 10-bits, from the 8-bit Timer2 module. Two extra bits are used to extend Timer2 to 10 bits (see Section **17.5.1**). A PWM output waveform is shown in Figure 17-4.

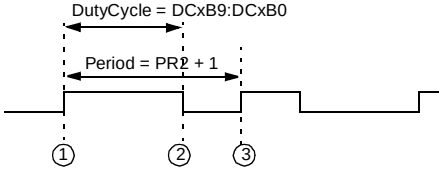
For a step-by-step procedure on how to set up the CCP module for PWM operation, see Section **17.5.4**.

Figure 17-3: Simplified PWM Block Diagram



Note 1: For 10-bit time base generation see Section **17.5.1**.

Figure 17-4: PWM Output Waveform

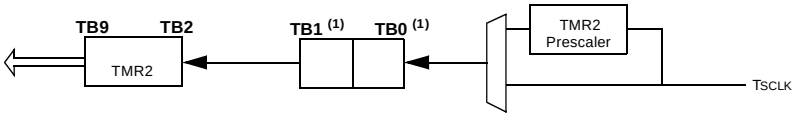


1. Timer2 is cleared and new duty cycle value is loaded from the Duty Cycle latch into the Duty Cycle Slave register
2. Timer2 value equals to value in Duty Cycle Latch register, CCP Pin is driven low
3. Timer2 overflow, value from Duty Cycle Latch is loaded into Slave Register, CCP Pin driven high

17.5.1 10-bit Time Base Generation

The PWM output has up to 10-bits of resolution. This is achieved by creating a 10-bit PWM Time base (TB9:TB0). Figure 17-5 shows the block diagram of this 10-bit PWM Time base. When TSCLK is the clock source to the 10-bit counter, the counter increments on each Tsclk. If a prescaler is selected, the 10-bit counter increments every prescale would by TSCLK.

Figure 17-5: 10-bit Time Base Block Diagram



Note 1: These two bits are not readable or writable and are not mapped into the data memory.

17.5.2 PWM Period

The PWM period is specified by writing to the PR2 register. The PWM period can be calculated using Equation 17-1.

Equation 17-1:Calculation for PWM Period

$$T_{PWM\ period} = [(PR2) + 1] \bullet 4 \bullet T_{SCLK} \bullet (\text{TMR2 prescale value})$$

Where PR2 = Value in PR2 Register

T_{SCLK} = Oscillator Clock

When TMR2 is equal to PR2, the following three events occur on the next increment cycle:

- TMR2 is cleared
- The CCPx pin is set (exception: if PWM duty cycle = 0%, the CCPx pin will not be set)
- The PWM duty cycle is latched from CCPRxL into CCPRxH

Note: The Timer2 postscaler is not used in the determination of the PWM frequency. The postscaler could be used to generate TMR2 interrupts at a different frequency than the PWM output.

17.5.3 PWM Duty Cycle

The PWM duty cycle is specified by writing to the CCPRxL register and to the DCxB1:DCxB0 (CCPxCON<5:4>) bits, if 10-bit resolution is desired. The CCPRxL contains the eight MSbs and CCPxCON<5:4> contains the two LSbs. This 10-bit value is represented by DCxB9:DCxB0. Equation 17-2 is used to calculate the PWM duty cycle.

Equation 17-2:Equation for calculating the PWM Duty Cycle

PWM Duty Cycle = (DCxB<9:0> bits value) • TSLCK • (TMR2 prescale value)

Where PWM Duty Cycle = PWM Duty Cycle Time
TSCLK = Oscillator Clock

The DCxB<9:0> bits can be written to at any time, but the duty cycle value is not latched into CCPRxH until after a match between PR2 and TMR2 occurs (which is the end of the current period). In PWM mode, CCPRxH is a read-only register.

The CCPRxH register and a 2-bit internal latch are used to double buffer the PWM duty cycle. This double buffering is essential for glitchless PWM operation.

When CCPRxH and a 2-bit latch match the value of TMR2 concatenated with the internal 2-bit Q clock (or two bits of the TMR2 prescaler), the CCPx pin is cleared. This is the end of the duty cycle. Equation 17-3 is used to calculate the maximum PWM resolution in bits for a given PWM frequency.

Equation 17-3:Calculation for Maximum PWM Resolution

$$\text{Maximum PWM Resolution (bits)} = \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)} \text{ bits}$$

Note: If the PWM duty cycle value is longer than the PWM period, the CCPx pin will not be cleared. This allows a duty cycle of 100%.

17.5.3.1 Minimum Resolution

The minimum resolution (in time) of each bit of the PWM duty cycle depends on the prescaler of Timer2. Table 17-5 shows the selections for the minimum resolution time.

Table 17-5: Minimum Duty Cycle Bit Time

Prescaler Value	T2CKPS1:T2CKPS0	Minimum Resolution (Time)
1	0 0	TSCLK
4	0 1	TCY
16	1 x	4 TCY

17.5.3.2 Example Calculation for PWM Period and Duty Cycle

This section shows an example calcuation for the PWM Period and Duty Cycle. Furthermore example PWM frequencies based upon different oscillator frequencies are given.

Example 17-2:PWM Period and Duty Cycle Calculation

Desired PWM frequency is 78.125 kHz,
Fosc = 20 MHz
TMR2 prescale = 1

1 / 78.125 kHz = [(PR2) + 1] • 4 • 1/20 MHz • 1

12.8 ms = [(PR2) + 1] • 4 • 50 ns • 1

PR2 = 63

Find the maximum resolution of the duty cycle that can be used with a 78.125 kHz frequency and 20 MHz oscillator:

1 / 78.125 kHz = $2^{\text{PWM RESOLUTION}}$ • 1/20 MHz • 1

12.8 ms = $2^{\text{PWM RESOLUTION}}$ • 50 ns • 1

256 = $2^{\text{PWM RESOLUTION}}$

log(256) = (PWM Resolution) • log(2)

PWM Resolution= 8.0

At most, an 8-bit resolution duty cycle can be obtained from a 78.125 kHz frequency and a 20 MHz oscillator (i.e., 0 ≤ DCxB9:DCxB0 ≤ 255). Any value greater than 255 will result in a 100% duty cycle.

In order to achieve higher resolution, the PWM frequency must be decreased. In order to achieve higher PWM frequency, the resolution must be decreased.

Table 17-6 lists example PWM frequencies and resolutions for FOSC = 20 MHz. Table 17-7 lists example PWM frequencies and resolutions for FOSC = 40 MHz. The TMR2 prescaler and PR2 values are also shown.

Table 17-6: Example PWM Frequencies and Bit Resolutions at 20 MHz

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12 kHz	156.3 kHz	208.3 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	5.5

Table 17-7: Example PWM Frequencies and Bit Resolutions at 40 MHz

PWM Frequency	2.44 kHz	9.76 kHz	39.06 kHz	78.12 kHz	208.3 kHz	416.6 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	5.5

17.5.4 Set-up for PWM Operation

The following steps configure the CCP module for PWM operation:

1. Establish the PWM period by writing to the PR2 register.
2. Establish the PWM duty cycle by writing to the DCxB9:DCxB0 bits.
3. Make the CCPx pin an output by clearing the appropriate TRIS bit.
4. Establish the TMR2 prescale value and enable Timer2 by writing to T2CON.
5. Configure the CCP module for PWM operation.

17.5.5 Sleep Operation

When the device is placed in sleep, Timer2 will not increment, and the state of the module will not change. If the CCP pin is driving a value, it will continue to drive that value. When the device wakes-up, it will continue from this state.

17.5.6 Effects of a Reset

The CCP module is off.

Table 17-8: Registers Associated with PWM and Timer2

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other resets
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1							TMR2IF (1)		0000 0000	0000 0000
PIE1							TMR2IE (1)		0000 0000	0000 0000
IPR1							TMR2IP (1)		0000 0000	0000 0000
TRISC	PORTC Data Direction Register								1111 1111	1111 1111
TMR2	Timer2 module's register								0000 0000	0000 0000
PR2	Timer2 module's period register								1111 1111	1111 1111
T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
CCPR1L	Capture/Compare/PWM register1 (LSB)								xxxx xxxx	uuuu uuuu
CCPR1H	Capture/Compare/PWM register1 (MSB)								xxxx xxxx	uuuu uuuu
CCP1CON	—	—	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
CCPR2L	Capture/Compare/PWM register2 (LSB)								xxxx xxxx	uuuu uuuu
CCPR2H	Capture/Compare/PWM register2 (MSB)								xxxx xxxx	uuuu uuuu
CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, — = unimplemented read as '0'.
Shaded cells are not used by PWM and Timer2.

Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18C2X2 devices. Always maintain these bits clear.

17.6 Initialization

The CCP module has three modes of operation. Example 17-3 shows the initialization of capture mode, Example 17-4 shows the initialization of compare mode and Example 17-5 shows the initialization of PWM mode.

Example 17-3:Capture Initialization

```
    CLRF   CCP1CON       ; CCP Module is off
    CLRF   TMR1H         ; Clear Timer1 High byte
    CLRF   TMR1L         ; Clear Timer1 Low byte
    CLRF   INTCON        ; Disable interrupts and clear T0IF
    BSF    TRISC, CCP1   ; Make CCP pin input
    CLRF   PIE1          ; Disable peripheral interrupts
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x06          ; Capture mode, every 4th rising edge
    MOVWF  CCP1CON       ;
    BSF    T1CON, TMR1ON ; Timer1 starts to increment
;
; The CCP1 interrupt is disabled,
; do polling on the CCP Interrupt flag bit
;
Capture_Event
    BTFSS  PIR1, CCP1IF
    GOTO   Capture_Event
;
; Capture has occured
;
    BCF    PIR1, CCP1IF  ; This needs to be done before
                         ;   next compare
```

Example 17-4:Compare Initialization

```
    CLRF   CCP1CON       ; CCP Module is off
    CLRF   TMR1H         ; Clear Timer1 High byte
    CLRF   TMR1L         ; Clear Timer1 Low byte
    CLRF   INTCON        ; Disable interrupts and clear T0IF
    MOVLW  0x80          ; Load 0x80 (Example Value)
                         ;   into W-Register
    MOVWF  CCPRIH        ; Load value to compare into CCPRIH
    MOVWF  CCPRIL        ; Load value to compare into CCPRIL
    BCF    TRISC, CCP1   ; Make CCP pin output if controlling
                         ;    state of pin
    CLRF   PIE1          ; Disable peripheral interrupts
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x08          ; Compare mode, set CCP1 pin on match
    MOVWF  CCP1CON       ;
    BSF    T1CON, TMR1ON ; Timer1 starts to increment
;
; The CCP1 interrupt is disabled,
; do polling on the CCP Interrupt flag bit
;
Compare_Event
    BTFSS  PIR1, CCP1IF
    GOTO   Compare_Event
;
; Compare has occured
;
      BCF   PIR1, CCP1IF ; This needs to be done before
                         ;  next compare
```

Example 17-5:PWM Initialization

```
    CLRF   CCP1CON       ; CCP Module is off
    CLRF   TMR2          ; Clear Timer2
    MOVLW  0x7F          ;
    MOVWF  PR2           ;
    MOVLW  0x1F          ;
    MOVWF  CCPR1L        ; Duty Cycle is 25% of PWM Period
    CLRF   INTCON        ; Disable interrupts and clear T0IF
    BCF    TRISC, PWM1   ; Make pin output
    CLRF   PIE1          ; Disable peripheral interrupts
    CLRF   PIR1          ; Clear peripheral interrupts Flags
    MOVLW  0x2C          ; PWM mode, 2 LSbs of
                         ;  Duty cycle = 10
    MOVWF  CCP1CON       ;
    BSF    T2CON, TMR2ON ; Timer2 starts to increment
;
; The CCP1 interrupt is disabled,
; do polling on the TMR2 Interrupt flag bit
;
PWM_Period_Match
    BTFSS  PIR1, TMR2IF
    GOTO   PWM_Period_Match
;
; Update this PWM period and the following PWM Duty cycle
;
    BCF    PIR1, TMR2IF
```

17.7 Design Tips

Question 1: ***What timers can I use for the capture and compare modes?***

Answer 1:

The capture and compare modes are designed around Timer1 and Timer3, so no other timer can be used for these functions. This also means that if multiple CCP modules are being used for a capture or compare function, they can share the same timer.

Question 2: ***What timers can I use with the PWM mode?***

Answer 2:

The PWM mode is designed around Timer2, so no other timer can be used for this function. It is the only timer with a period register associated with it. If multiple CCP modules are doing PWM, they will share the same timer and have the same PWM period and frequency.

Question 3: ***Can I use one CCP module to do capture (or compare) AND PWM at the same time, since they use different timers as their reference?***

Answer 3:

The timers may be different, but other logic functions are shared. However, you can switch from one mode to the other. For a device with 2 CCP modules, you can also have CCP1 set up for PWM and CCP2 set up for capture or compare (or vice versa) since they are two independent modules.

Question 4: ***How does a reset affect the CCP module?***

Answer 4:

Any reset will turn the CCP module off. See the **"Reset"** section to see reset values.

Question 5: ***I am setting up the CCP1CON module for "Compare Mode, trigger special event" (1011) that resets TMR1. When a compare match occurs, will I have both the TMR1 and the CCP1 interrupts pending (TMR1IF is set, CCP1IF is set)?***

Answer 5:

The CCP1IF flag will be set on the match condition. TMR1IF is set when Timer1 overflows, and the special trigger reset of Timer1 is not considered an overflow. However, if both the CCPR1L and CCPR1H registers are set at FFh, then an overflow occurs at the same time as the match, which will then set both CCP1IF and TMR1IF.

Question 6: ***How do I use Timer2 as a general purpose timer, with an interrupt flag on rollover?***

Answer 6:

Timer2 always resets to zero when it equals PR2 and flag bit TMR2IF always gets set at this time. By putting FFh into PR2, you will get an interrupt on overflow at FFh. Quite often it is desirable to have an event occur at a periodic rate, perhaps an interrupt driven event. Normally an initial value would be placed into the timer so that the overflow will occur at the desired time. This value would have to be placed back into the timer every time it overflowed to make the interrupts occur at the same desired rate. The benefit of Timer2 is that a value can be written to PR2 that will cause it to reset at your desired time interval. This means you do not have the housekeeping chore of reloading the timer every time it overflows, since PR2 maintains its value.

Question 7: ***I am using a CCP module in PWM mode. The duty cycle being outputted is almost always 100%, even when my program writes a value like 7Fh to the duty cycle register, which should be 50%. What am I doing wrong?***

Answer 7:

1. The value in CCPRxL is higher than PR2. This happens quite often when a user desires a fast PWM output frequency and writes a small value in the PR2. In this case, if a value of 7Eh were written to PR2, then a value 7Fh in CCPRxL will result in 100% duty cycle.
2. If the TRIS bit corresponding to the CCP output pin you are using is configured as an input, the PWM output cannot drive the pin. In this case, the pin would float and duty cycle may appear to be 0%, 100% or some other floating value.

Question 8: ***I want to determine a signal frequency using the CCP module in capture mode to find the period. I am currently resetting Timer1 on the first edge, then using the value in the capture register on the second edge as the time period. The problem is that my code to clear the timer does not occur until almost twelve instructions after the first capture edge (interrupt latency plus saving of registers in interrupt), so I cannot measure very fast frequencies. Is there a better way to do this?***

Answer 8:

You do not need to zero the counter to find the difference between two pulse edges. Just take the first captured value and put it into another set of registers. Then when the second capture event occurs, subtract the first event from the second. Assuming that your pulse edges are not so far apart that the counter can wrap around past the last capture value, the answer will always be correct. This is illustrated by the following example:

1. First captured value is FFFEh. Store this value in two registers.
2. The second capture value is 0001h (the counter has incremented three times).
3. 0001h - FFFEh = 0003, which is the same as if you had cleared Timer1 to zero and let it count to 3. (Theoretically, except that there was a delay getting to the code that clears Timer1, so actual values would differ).

The interrupt overhead is now less important because the values are captured automatically. For even faster inputs, do not enable interrupts and just test the flag bit in a loop. If you must also capture very long time periods, such that the timer can wrap around past the previous capture value, then consider using an auto-scaling technique that starts with a large prescale, and shorten the prescale as you converge on the exact frequency.

17.8 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is, they may be written for the Base-Line, the Mid-Range or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the CCP modules are:

Title	Application Note #
Using the CCP Modules	AN594
Implementing Ultrasonic Ranging	AN597
Air Flow Control Using Fuzzy Logic	AN600
Adaptive Differential Pulse Code Modulation	AN643

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

17.9 Revision History

Revision A

This is the initial released revision of the Enhanced MCU CCP module description.

Section 18. ECCP

Please check the Microchip web site for Revision B of the ECCP Section.

18.1 Revision History

Revision A

This is the initial released revision of the Enhanced MCU ECCP module description.

Section 19. Synchronous Serial Port (SSP)

HIGHLIGHTS

This section of the manual contains the following major topics:

I²C is a trademark of Philips Corporation.

19.1 Introduction

The Synchronous Serial Port (SSP) module is a serial interface useful for communicating with other peripherals or microcontroller devices. These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D converters, etc. The SSP module can operate in one of two modes:

- Serial Peripheral Interface (SPI™)
- Inter-Integrated Circuit (I^2C™)
  - Slave mode
  - I/O slope control, and Start and Stop bit detection to ease software implementation of Master and Multi-master modes

SPI is a registered trademark of Motorola Corporation.
I^2C is a trademark of Philips Corporation.

Section **19.2** forced to next page for formatting purposes.

19.2 Control Registers

Register 19-1 shows the SSPSTAT register while Register 19-2 shows the SSPCON register.

Register 19-1: SSPSTAT: Synchronous Serial Port Status Register

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** SPI data input sample phase

SPI Master Mode

1 = Input data sampled at end of data output time
0 = Input data sampled at middle of data output time

SPI Slave Mode

SMP must be cleared when SPI is used in slave mode

I2C Mode

This bit must be maintained clear.

bit 6 **CKE**: SPI Clock Edge Select (Figure 19-3, Figure 19-4, and Figure 19-5)

SPI Mode

CKP = 0 (SSPCON<4>)

1 = Data transmitted on rising edge of SCK
0 = Data transmitted on falling edge of SCK

CKP = 1 (SSPCON<4>)

1 = Data transmitted on falling edge of SCK
0 = Data transmitted on rising edge of SCK

I^2C Mode

This bit must be maintained clear.

bit 5 **D/$\overline{A}$**: Data/$\overline{Address}$ bit (I^2C mode only)

1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P**: Stop bit
(I^2C mode only. This bit is cleared when the SSP module is disabled)

1 = Indicates that a stop bit has been detected last (this bit is '0' on RESET)
0 = Stop bit was not detected last

bit 3 **S**: Start bit
(I^2C mode only. This bit is cleared when the SSP module is disabled)

1 = Indicates that a start bit has been detected last (this bit is '0' on RESET)
0 = Start bit was not detected last

bit 2 **R/$\overline{W}$**: Read/$\overline{Write}$ bit information (I^2C mode only)

This bit holds the R/W bit information following the last address match. This bit is only valid from the address match to the next start bit, stop bit, or not $\overline{ACK}$ bit.

1 = Read
0 = Write

bit 1 **UA**: Update Address (10-bit I^2C mode only)

1 = Indicates that the user needs to update the address in the SSPADD register
0 = Address does not need to be updated

bit 0 **BF**: Buffer Full Status bit

Receive (SPI and I^2C modes)

1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty

Transmit (I^2C mode only)

1 = Transmit in progress, SSPBUF is full
0 = Transmit complete, SSPBUF is empty

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 19-2: SSPCON: Synchronous Serial Port Control Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL**: Write Collision Detect bit

1 = The SSPBUF register was written to while the previous word was being transmitted. (must be cleared in software)
0 = No collision

bit 6 **SSPOV**: Receive Overflow Indicator bit

In SPI mode:

1 = A new byte was received while the SSPBUF register was still holding the previous data. In case of overflow, the data in SSPSR is lost and the SSPBUF is no longer updated. Overflow can only occur in slave mode. The user must read the SSPBUF, even if only transmitting data, to avoid setting overflow. In master mode the overflow bit is not set since each new reception (and transmission) is initiated by writing to the SSPBUF register.

0 = No overflow

In I^2C mode:

1 = A byte was received while the SSPBUF register was still holding the previous byte. SSPO is a “don‘t care” in transmit mode. SSPOV must be cleared in software in either mode.

0 = No overflow

bit 5 **SSPEN**: Synchronous Serial Port Enable bit
In both modes, when enabled, these pins must be properly configured as input or output.

In SPI mode:
1 = Enables serial port and configures SCK, SDO, SDI, and $\overline{SS}$ as the source of the serial port pins
0 = Disables serial port and configures these pins as I/O port pins

In I^2C mode:
1 = Enables the serial port and configures the SDA and SCL pins as the source of the serial port pins
0 = Disables serial port and configures these pins as I/O port pins

bit 4 **CKP**: Clock Polarity Select bit

In SPI mode:
1 = Idle state for clock is a high level
0 = Idle state for clock is a low level

In I^2C mode:
SCK release control
1 = Enable clock
0 = Holds clock low (clock stretch) (Used to ensure data setup time)

bit 3:0 **SSPM3:SSPM0**: Synchronous Serial Port Mode Select bits

0000 = SPI master mode, clock = FOSC/4
0001 = SPI master mode, clock = FOSC/16
0010 = SPI master mode, clock = FOSC/64
0011 = SPI master mode, clock = TMR2 output/2
0100 = SPI slave mode, clock = SCK pin. $\overline{SS}$ pin control enabled.
0101 = SPI slave mode, clock = SCK pin. $\overline{SS}$ pin control disabled. $\overline{SS}$ can be used as I/O pin
0110 = I^2C slave mode, 7-bit address
0111 = I^2C slave mode, 10-bit address
1000 = Reserved
1001 = Reserved
1010 = Reserved
1011 = I^2C firmware controlled master mode (slave idle)
1100 = Reserved
1101 = Reserved
1110 = I^2C slave mode, 7-bit address with start and stop bit interrupts enabled
1111 = I^2C slave mode, 10-bit address with start and stop bit interrupts enabled

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Microwire is a trademark of National Semiconductor.

19.3 SPI Mode

The SPI mode allows 8-bits of data to be synchronously transmitted and received simultaneously. All four modes of SPI are supported, as well as Microwire™ (sample edge) when the SPI is in the master mode.

To accomplish communication, typically three pins are used:

- Serial Data Out (SDO)
- Serial Data In (SDI)
- Serial Clock (SCK)

Additionally a fourth pin may be used when in a slave mode of operation:

- Slave Select ($\overline{SS}$)

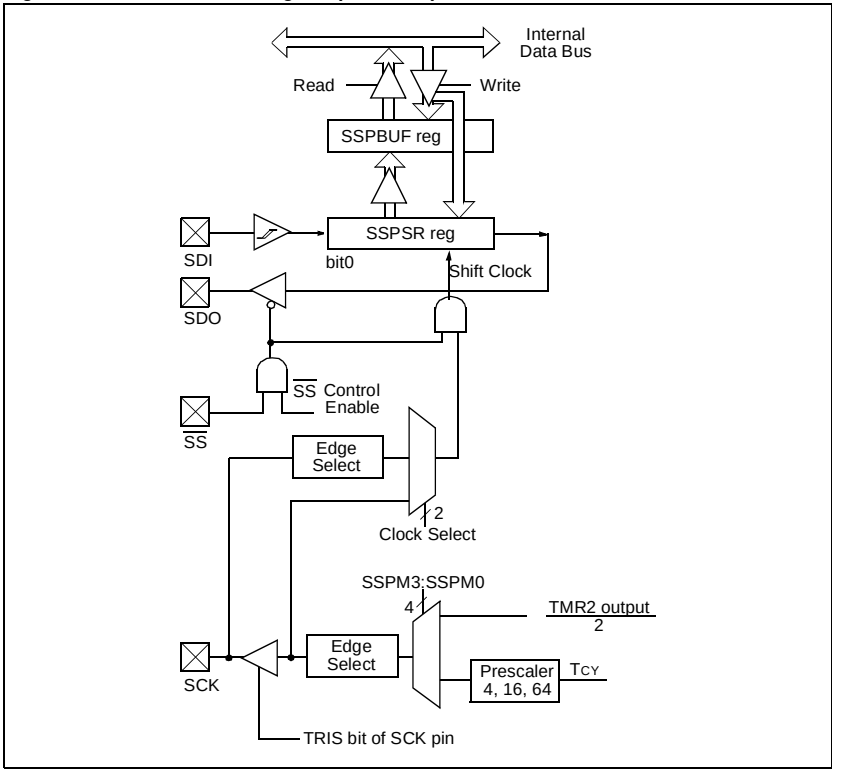
19.3.1 Operation

When initializing the SPI, several options need to be specified. This is done by programming the appropriate control bits in the SSPCON register (SSPCON<5:0>) and SSPSTAT<7:6>. These control bits allow the following to be specified:

- Master Mode (SCK is the clock output)
- Slave Mode (SCK is the clock input)
- Clock Polarity (Idle state of SCK)
- Clock edge (output data on rising/falling edge of SCK)
- Data Input Sample Phase
- Clock Rate (Master mode only)
- Slave Select Mode (Slave mode only)

Figure 19-1 shows the block diagram of the SSP module, when in SPI mode.

Figure 19-1: SSP Block Diagram (SPI Mode)



The SSP consists of a transmit/receive Shift Register (SSPSR) and a buffer register (SSPBUF). The SSPSR shifts the data in and out of the device, MSb first. The SSPBUF holds the data that was written to the SSPSR, until the received data is ready. Once the 8-bits of data have been received, that byte is moved to the SSPBUF register. Then the buffer full detect bit, BF (SSPSTAT<0>), and interrupt flag bit, SSPIF, are set. This double buffering of the received data (SSPBUF) allows the next byte to start reception before reading the data that was just received. Any write to the SSPBUF register during transmission/reception of data will be ignored, and the write collision detect bit, WCOL (SSPCON<7>), will be set. User software must clear the WCOL bit so that it can be determined if the following write(s) to the SSPBUF register completed successfully. When the application software is expecting to receive valid data, the SSPBUF should be read before the next byte of data to transfer is written to the SSPBUF. Buffer full bit, BF (SSPSTAT<0>), indicates when SSPBUF has been loaded with the received data (transmission is complete). When the SSPBUF is read, the BF bit is cleared. This data may be irrelevant if the SPI is only a transmitter. Generally the SSP Interrupt is used to determine when the transmission/reception has completed. The SSPBUF must be read and/or written. If the interrupt method is not going to be used, then software polling can be done to ensure that a write collision does not occur. Example 19-1 shows the loading of the SSPBUF (SSPSR) for data transmission. The shaded instruction is only required if the received data is meaningful (some SPI applications are transmit only).

Example 19-1: Loading the SSPBUF (SSPSR) Register

```
LOOP BTFSS SSPSTAT, BF    ;Has data been received
                          ;  (transmit complete)?
     GOTO  LOOP           ;No
     MOVF  SSPBUF, W      ;W reg = contents of SSPBUF
     MOVWF RXDATA         ;Save in user RAM,
                          ;  if data is meaningful
     MOVFF TXDATA, SSPBUF ;contents of TXDATA
                          ;  is the new data to transmit
```

The SSPSR is not directly readable or writable, and can only be accessed from addressing the SSPBUF register. Additionally, the SSP status register (SSPSTAT) indicates the various status conditions.

19.3.2 Enabling SPI I/O

To enable the serial port the SSP Enable bit, SSPEN (SSPCON<5>), must be set. To reset or reconfigure SPI mode, clear the SSPEN bit which re-initializes the SSPCON register, and then set the SSPEN bit. This configures the SDI, SDO, SCK, and $\overline{SS}$ pins as serial port pins. For the pins to behave as the serial port function, they must have their data direction bits (in the TRIS register) appropriately programmed. That is:

- SDI must have the TRIS bit set
- SDO must have the TRIS bit cleared
- SCK (Master mode) must have the TRIS bit cleared
- SCK (Slave mode) must have the TRIS bit set
- $\overline{SS}$ must have the TRIS bit set

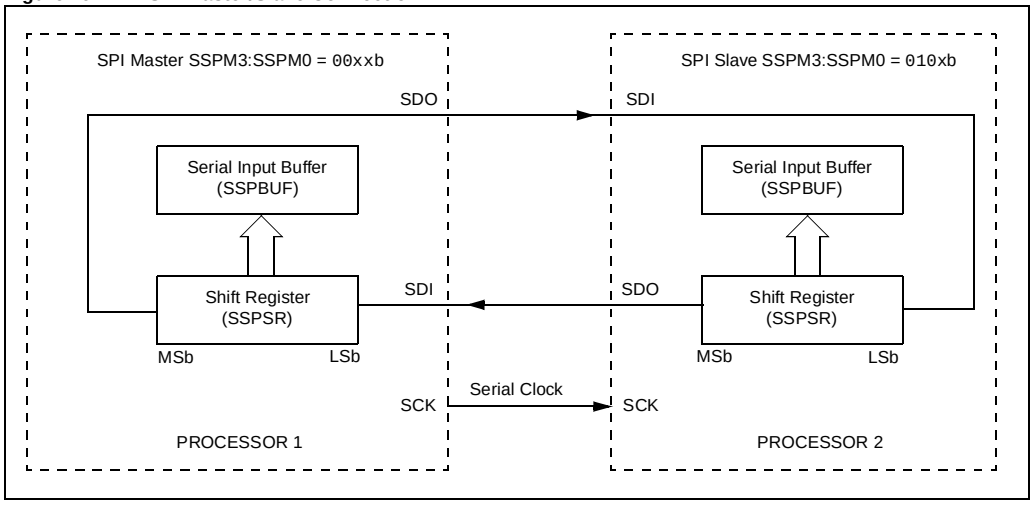
Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value. An example would be in master mode where you are only sending data (to a display driver), then both SDI and $\overline{SS}$ could be used as general purpose outputs by clearing their corresponding TRIS register bits.

19.3.3 Typical Connection

Figure 19-2 shows a typical connection between two microcontrollers. The master controller (Processor 1) initiates the data transfer by sending the SCK signal. Data is shifted out of both shift registers on their programmed clock edge, and latched on the edge of the clock specified by the SMP bit. Both processors should be programmed to same Clock Polarity (CKP), so both controllers would send and receive data at the same time. Whether the data is meaningful (or dummy data) depends on the application software. This leads to three scenarios for data transmission:

- Master sends data — Slave sends dummy data
- Master sends data — Slave sends data
- Master sends dummy data — Slave sends data

Figure 19-2: SPI Master/Slave Connection



19.3.4 Master Operation

The master can initiate the data transfer at any time because it controls the SCK. The master determines when the slave (Processor 2) is to broadcast data by the software protocol.

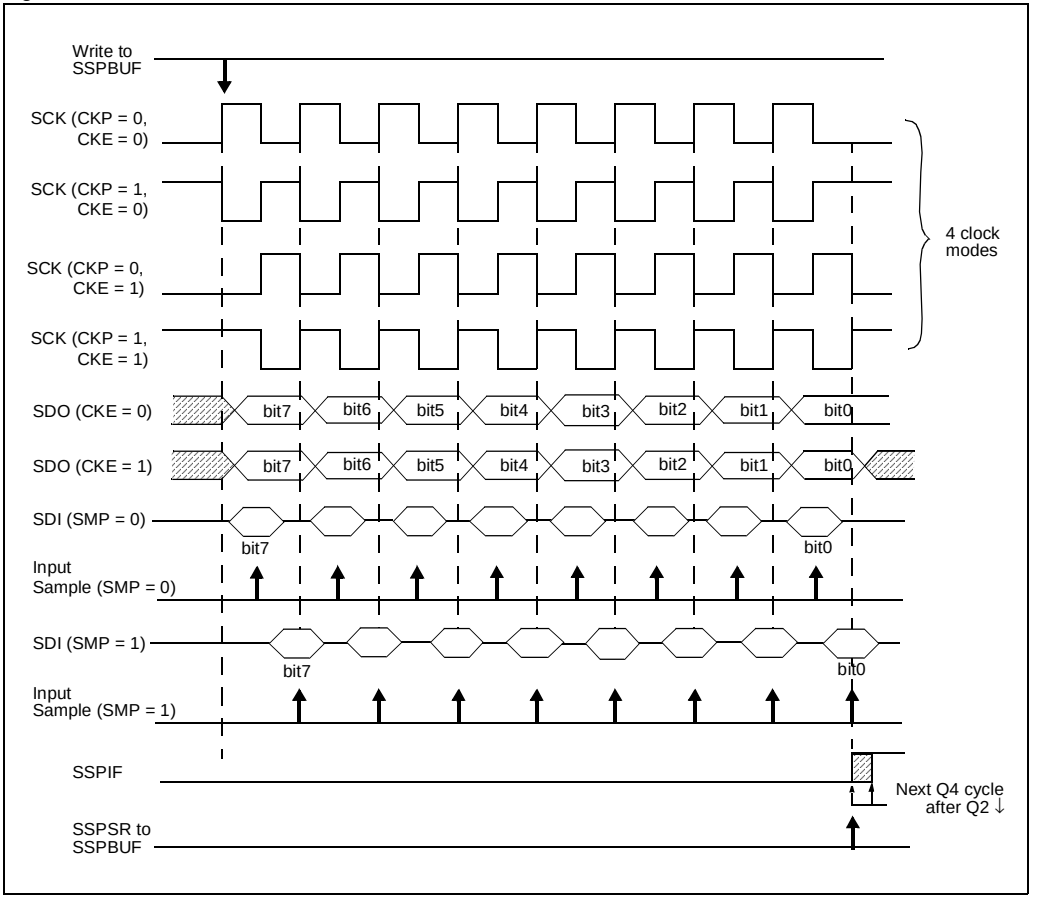
In master mode the data is transmitted/received as soon as the SSPBUF register is written to. If the SPI is only going to receive, the SDO output could be disabled (programmed as an input). The SSPSR register will continue to shift in the signal present on the SDI pin at the programmed clock rate. As each byte is received, it will be loaded into the SSPBUF register as if a normal received byte (interrupts and status bits appropriately set). This could be useful in receiver applications as a “line activity monitor” mode.

The clock polarity is selected by appropriately programming bit CKP (SSPCON<4>). This would give waveforms for SPI communication as shown in Figure 19-3, Figure 19-4, and Figure 19-5 where the MSb is transmitted first. In master mode, the SPI clock rate (bit rate) is user programmable to be one of the following:

- FOSC/4 (or TCY)
- FOSC/16 (or 4 • TCY)
- FOSC/64 (or 16 • TCY)
- Timer2 output/2

This allows a maximum data rate of 5 Mbps (at 20 MHz).

Figure 19-3: SPI Mode Waveform, Master Mode



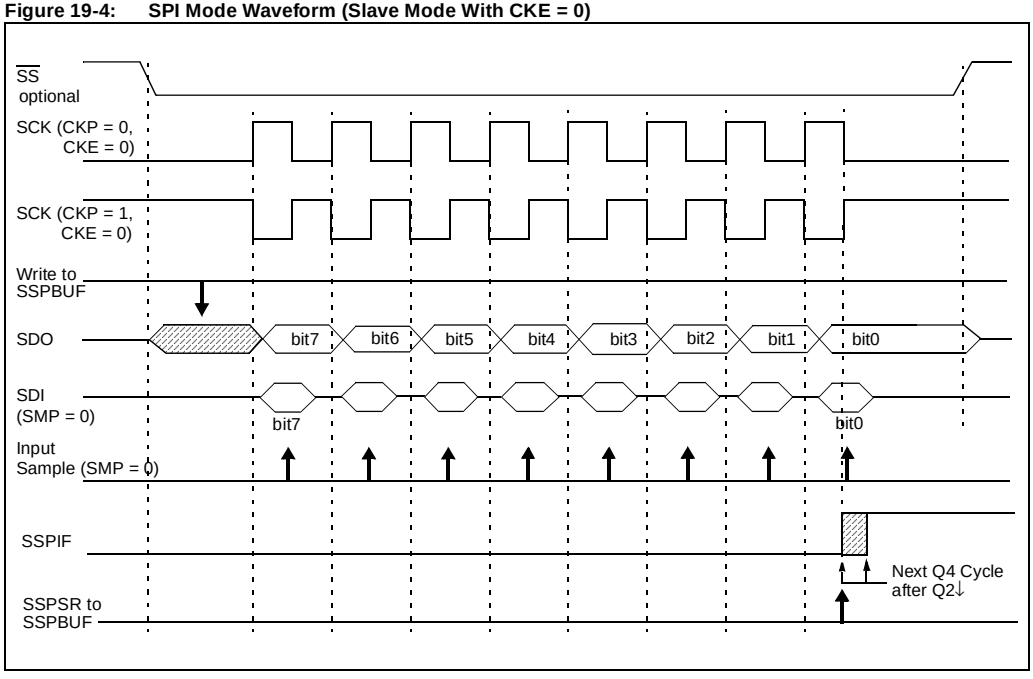
19.3.5 Slave Operation

In slave mode, the data is transmitted and received as the external clock pulses appear on SCK. When the last bit is latched, the interrupt flag bit SSPIF is set.

The clock polarity is selected by appropriately programming bit CKP (SSPCON). This then would give waveforms for SPI communication as shown in Figure 19-3, Figure 19-4, and Figure 19-5 where the MSb is transmitted first. When in slave mode the external clock must meet the minimum high and low times.

In sleep mode, the slave can transmit and receive data. When a byte is received, the device will wake-up from sleep, if the interrupt is enabled.

Figure 19-4: SPI Mode Waveform (Slave Mode With CKE = 0)



19.3.6 Slave Select Mode

When in slave select mode, the $\overline{SS}$ pin allows multi-drop for multiple slaves with a single master. The SPI must be in slave mode (SSPCON<3:0> = 04h) and the TRIS bit, for the $\overline{SS}$ pin, must be set for the slave select mode to be enabled. When the $\overline{SS}$ pin is low, transmission and reception are enabled and the SDO pin is driven. When the $\overline{SS}$ pin goes high, the SDO pin is no longer driven, even if in the middle of a transmitted byte, and becomes a floating output. External pull-up/ pull-down resistors may be desirable, depending on the application.

When the SPI is in Slave Mode with $\overline{SS}$ pin control enabled, (SSPCON<3:0> = 0100) the SPI module will reset if the $\overline{SS}$ pin is set to VDD. If the SPI is used in Slave Mode with the CKE bit is set, then the $\overline{SS}$ pin control must be enabled.

When the SPI module resets, the bit counter is forced to 0. This can be done by either by forcing the $\overline{SS}$ pin to a high level or clearing the SSPEN bit (Figure 19-6).

To emulate two-wire communication, the SDO pin can be connected to the SDI pin. When the SPI needs to operate as a receiver the SDO pin can be configured as an input. This disables transmissions from the SDO. The SDI can always be left as an input (SDI function) since it cannot create a bus conflict.

Figure 19-5: SPI Mode Waveform (Slave Select Mode With CKE = 1)

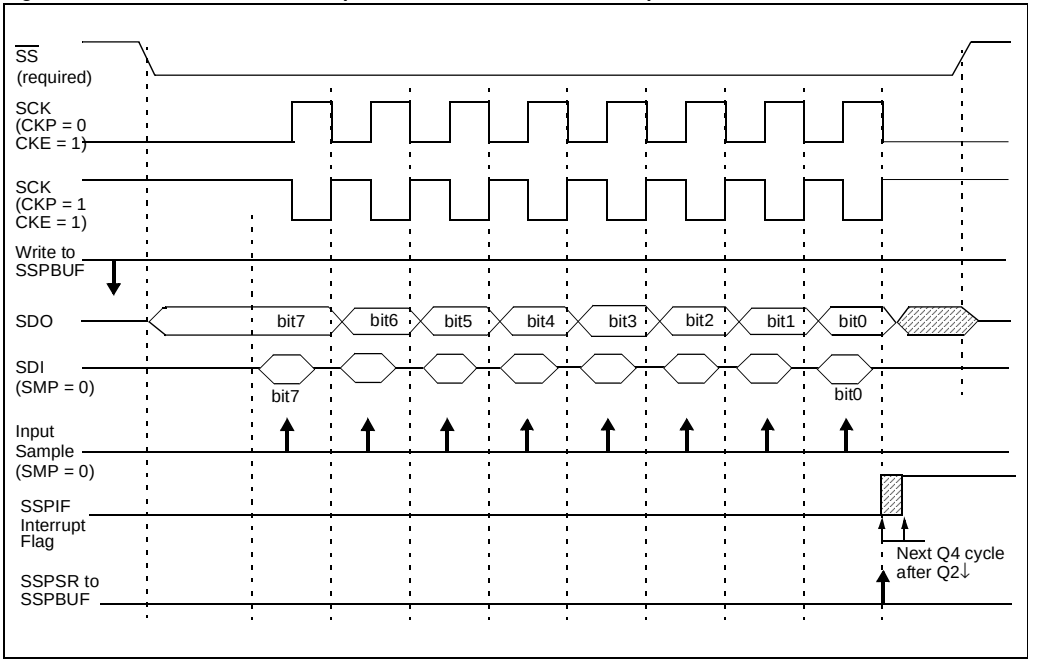
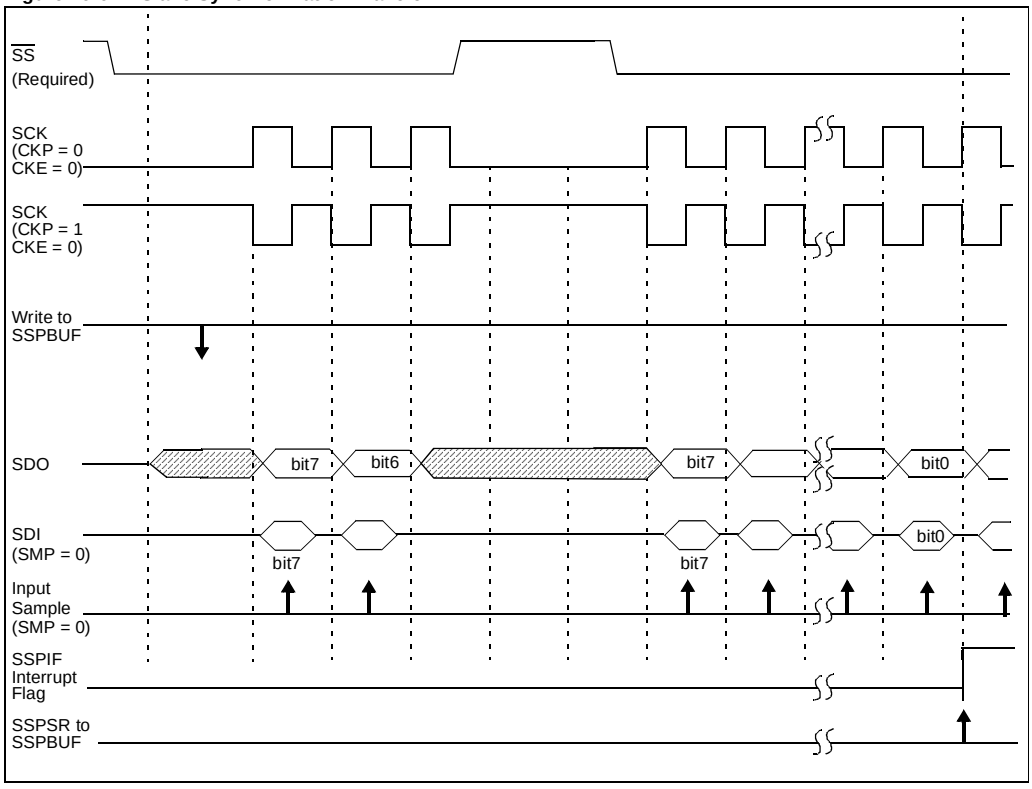


Figure 19-6: Slave Synchronization Waveform



19.3.7 Sleep Operation

In master mode all module clocks are halted, and the transmission/reception will remain in that state until the device wakes from sleep. After the device returns to normal mode, the module will continue to transmit/receive data.

In slave mode, the SPI transmit/receive shift register operates asynchronously to the device. This allows the device to be placed in sleep mode, and data to be shifted into the SPI transmit/receive shift register. When all 8-bits have been received, the SSP interrupt flag bit will be set and if enabled will wake the device from sleep.

19.3.8 Effects of a Reset

A reset disables the SSP module and terminates the current transfer.

Table 19-1: Registers Associated with SPI Operation

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other resets
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR	SSPIF (1)								0	0
IPR	SSPIP (1)								0	0
SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
TRISC	PORTC Data Direction Control Register								1111 1111	1111 1111
SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	0000 0000

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'. Shaded cells are not used by the SSP in SPI mode.

Note 1: The position of this bit is device dependent.

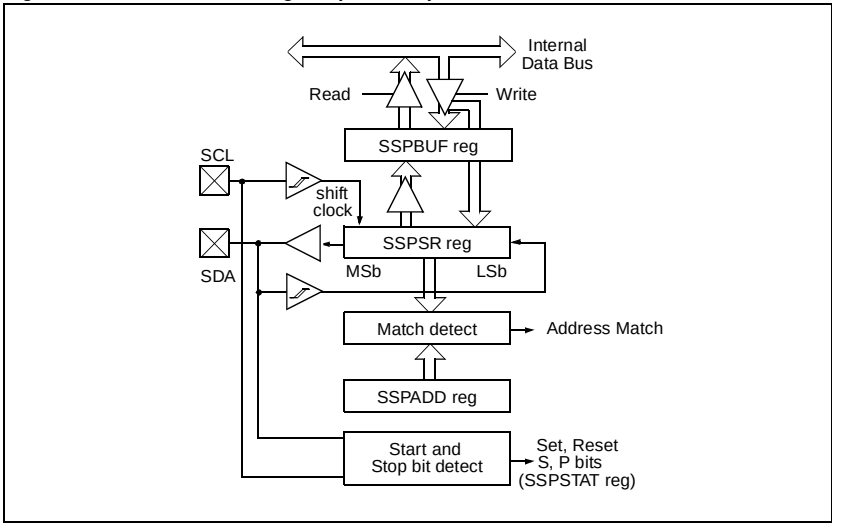
19.4 SSP I²C Operation

The SSP module in I²C mode fully implements all slave functions, except general call support, and provides interrupts on start and stop bits in hardware to facilitate software implementations of the master functions. The SSP module implements the standard mode specifications as well as 7-bit and 10-bit addressing. The **"Appendix"** section gives an overview of the I²C bus specification.

Two pins are used for data transfer. These are the SCL pin, which is the clock, and the SDA pin, which is the data. The user must configure these pins as inputs through the TRIS bits. The SSP module functions are enabled by setting SSP Enable bit, SSPEN (SSPCON).

A "glitch" filter is on the SCL and SDA pins when the pin is an input. This filter operates in both the 100 KHz and 400 KHz modes. In the 100 KHz mode, when these pins are an output, there is a slew rate control of the pin that is independent of device frequency.

Figure 19-7: SSP Block Diagram (I²C Mode)



The SSP module has five registers for I^2C operation. They are:

- SSP Control Register (SSPCON)
- SSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer (SSPBUF)
- SSP Shift Register (SSPSR) - Not directly accessible
- SSP Address Register (SSPADD)

The SSPCON register allows control of the I^2C operation. Four mode selection bits (SSPCON<3:0>) allow one of the following I^2C modes to be selected:

- I^2C Slave mode (7-bit address)
- I^2C Slave mode (10-bit address)
- I^2C Firmware controlled Multi-Master mode (start and stop bit interrupts enabled)
- I^2C Firmware controlled Multi-Master mode (start and stop bit interrupts enabled)
- I^2C Firmware controlled Master mode, slave is idle

Before selecting any I^2C mode, the SCL and SDA pins must be programmed to inputs by setting the appropriate TRIS bits. Selecting an I^2C mode by setting the SSPEN bit enables the SCL and SDA pins to be used as the clock and data lines in I^2C mode.

The SSPSTAT register gives the status of the data transfer. This information includes detection of a START or STOP bit, specifies if the received byte was data or address, if the next byte is the completion of 10-bit address, and if this will be a read or write data transfer.

The SSPBUF is the register to which transfer data is written to or read from. The SSPSR register shifts the data in or out of the device. In receive operations, the SSPBUF and SSPSR create a doubled buffered receiver. This allows reception of the next byte to begin before reading the last byte of received data. When the complete byte is received, it is transferred to the SSPBUF register and flag bit SSPIF is set. If another complete byte is received before the SSPBUF register is read, a receiver overflow has occurred and the SSPOV bit (SSPCON<6>) is set and the byte in the SSPSR is lost.

The SSPADD register holds the slave address. In 10-bit mode, the user needs to write the high byte of the address (`1111 0 A9 A8 0`). Following the high byte address match, the low byte of the address needs to be loaded (A7:A0).

19.4.1 Slave Mode

In slave mode, the SCL and SDA pins must be configured as inputs (TRIS set). The SSP module will override the input state with the output data when required (slave-transmitter).

When an address is matched or the data transfer after an address match is received, the hardware automatically will generate the acknowledge ($\overline{ACK}$) pulse, and then load the SSPBUF register with the received value currently in the SSPSR register.

There are certain conditions that will cause the SSP module not to give this $\overline{ACK}$ pulse. These are if either (or both):

a) The buffer full bit, BF (SSPSTAT<0>), was set before the message completed.

b) The overflow bit, SSPOV (SSPCON<6>), was set before the message completed.

In this case, the SSPSR register value is not loaded into the SSPBUF, but the SSPIF and SSPOV bits are set. Table 19-2 shows what happens when a data transfer byte is received, given the status of bits BF and SSPOV. The shaded cells show the condition where user software did not properly clear the overflow condition. The BF flag bit is cleared by reading the SSPBUF register while bit SSPOV is cleared through software.

The SCL clock input must have a minimum high and low time for proper operation. The high and low times of the I^2C specification as well as the requirement of the SSP module is shown in the Device Data Sheet electrical specifications parameters 100 and 101.

19.4.1.1 Addressing

Once the SSP module has been enabled, it waits for a START condition to occur. Following the START condition, the 8-bits are shifted into the SSPSR register. All incoming bits are sampled with the rising edge of the clock (SCL) line. The value of register SSPSR<7:1> is compared to the value of the SSPADD register. The address is compared on the falling edge of the eighth clock (SCL) pulse. If the addresses match, and the BF and SSPOV bits are clear, the following events occur:

a) The SSPSR register value is loaded into the SSPBUF register on the falling edge of the eighth SCL pulse.

b) The buffer full bit, BF, is set on the falling edge of the eighth SCL pulse.

c) An $\overline{ACK}$ pulse is generated.

d) The SSP interrupt flag bit, SSPIF, is set (and an interrupt is generated if enabled) - on the falling edge of the ninth SCL pulse.

In 10-bit address mode, two address bytes need to be received by the slave. The five Most Significant bits (MSbs) of the first address byte specify if this is a 10-bit address. The R/$\overline{W}$ bit (SSPSTAT) must specify a write so the slave device will receive the second address byte. For a 10-bit address the first byte would equal '1111 0 A9 A8 0', where A9 and A8 are the two MSbs of the address. The sequence of events for a 10-bit address is as follows, with steps 7- 9 for slave-transmitter:

1. Receive first (high) byte of Address (the SSPIF, BF, and UA (SSPSTAT) bits are set).
2. Update the SSPADD register with second (low) byte of Address (clears the UA bit and releases the SCL line).
3. Read the SSPBUF register (clears the BF bit) and clear the SSPIF flag bit.
4. Receive second (low) byte of Address (the SSPIF, BF, and UA bits are set).
5. Update the SSPADD register with the high byte of Address. This will clear the UA bit and releases SCL line.
6. Read the SSPBUF register (clears the BF bit) and clear the SSPIF flag bit.
7. Receive repeated START condition.
8. Receive first (high) byte of Address (the SSPIF and BF bits are set).
9. Read the SSPBUF register (clears the BF bit) and clear the SSPIF flag bit.

Note:	Following the RESTART condition (step 7) in 10-bit mode, the user only needs to match the first 7-bit address. The user does not update the SSPADD for the second half of the address.

Table 19-2: Data Transfer Received Byte Actions

Status Bits as Data Transfer is Received				
BF	**SSPOV**	**SSPSR → SSPBUF**	**Generate $\overline{ACK}$ Pulse**	**Set bit SSPIF (SSP Interrupt occurs if enabled)**
0	0	Yes	Yes	Yes
1	0	No	No	Yes
1	1	No	No	Yes
0	1	Yes	No	Yes

Note: Shaded cells show the conditions where the user software did not properly clear the overflow condition.

19.4.1.2 Reception

When the R/$\overline{W}$ bit of the address byte is clear and an address match occurs, the R/$\overline{W}$ bit of the SSPSTAT register is cleared. The received address is loaded into the SSPBUF register.

When the address byte overflow condition exists, then no acknowledge ($\overline{ACK}$) pulse is given. An overflow condition is defined as either the BF bit (SSPSTAT) is set or the SSPOV bit (SSPCON) is set. When a byte is received with these conditions, and attempts to move from the SSPSR register to the SSPBUF register, no acknowledge pulse is given.

An SSP interrupt is generated for each data transfer byte. The SSPIF flag bit must be cleared in software. The SSPSTAT register is used to determine the status of the receive byte.

Figure 19-8: I²C Waveforms for Reception (7-bit Address)

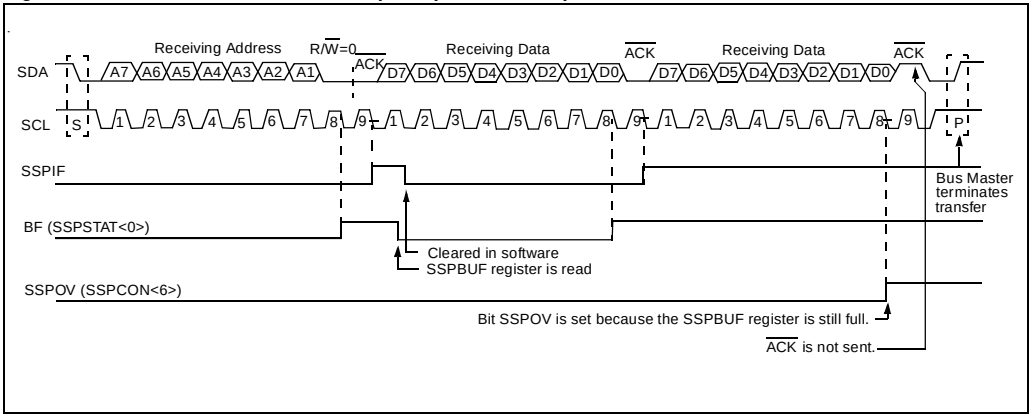
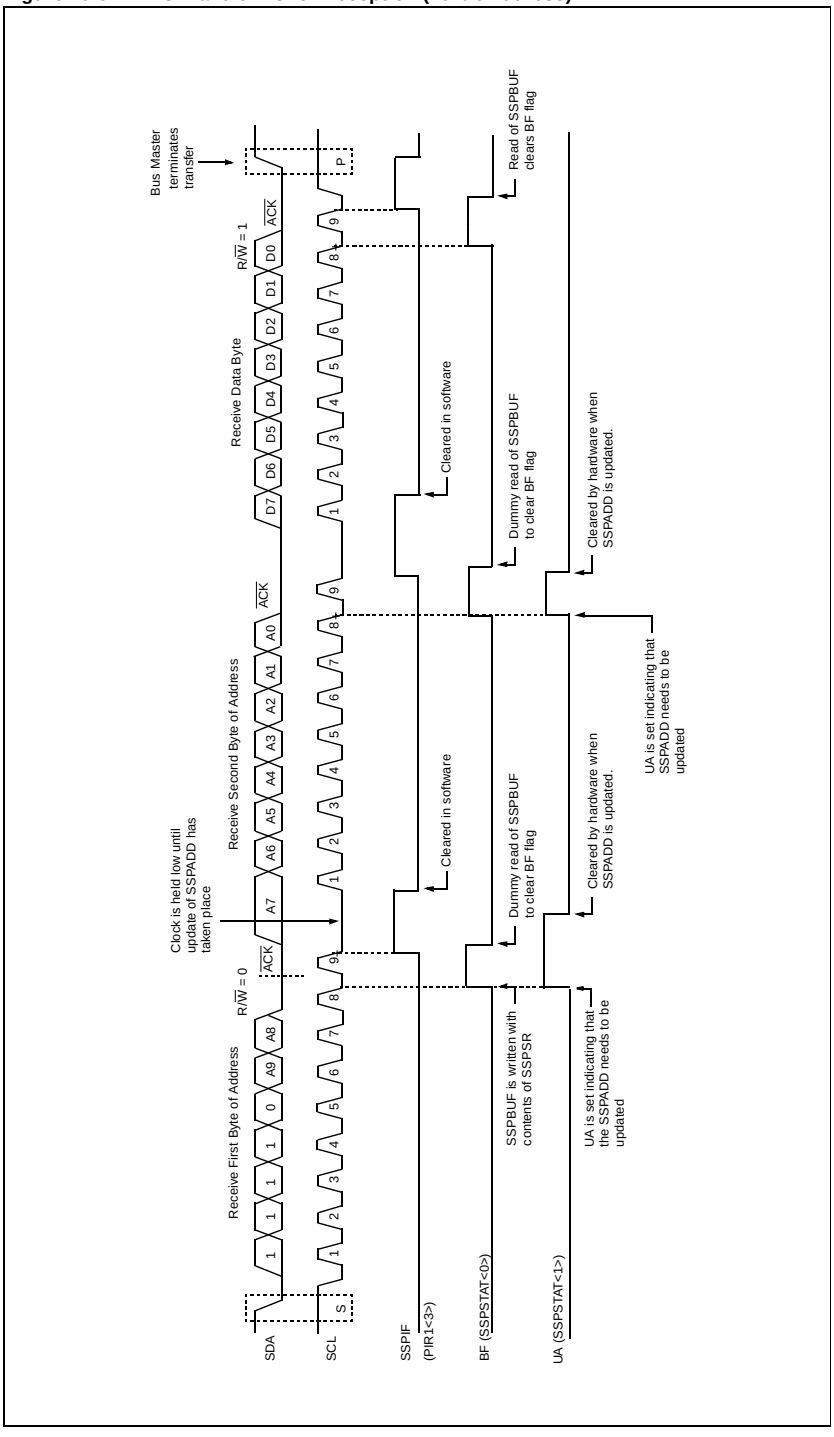


Figure 19-9: I^2C Waveforms for Reception (10-bit Address)



19.4.1.3 Transmission

When the R/$\overline{W}$ bit of the incoming address byte is set and an address match occurs, the R/$\overline{W}$ bit of the SSPSTAT register is set. The received address is loaded into the SSPBUF register. The $\overline{ACK}$ pulse will be sent on the ninth bit, and the SCL pin is held low. The transmit data must be loaded into the SSPBUF register, which also loads the SSPSR register. Then the SCL pin should be enabled by setting the CKP bit (SSPCON<4>). The master must monitor the SCL pin prior to asserting another clock pulse. The slave devices may be holding off the master by stretching the clock. The eight data bits are shifted out on the falling edge of the SCL input. This ensures that the SDA signal is valid during the SCL high time (Figure 19-10).

An SSP interrupt is generated for each data transfer byte. The SSPIF flag bit must be cleared in software, and the SSPSTAT register is used to determine the status of the byte transfer. The SSPIF flag bit is set on the falling edge of the ninth clock pulse.

As a slave-transmitter, the $\overline{ACK}$ pulse from the master-receiver is latched on the rising edge of the ninth SCL input pulse. If the SDA line was high (not $\overline{ACK}$), then the data transfer is complete. When the not $\overline{ACK}$ is latched by the slave, the slave logic is reset and the slave then monitors for another occurrence of the START bit. If the SDA line was low ($\overline{ACK}$), the transmit data must be loaded into the SSPBUF register, which also loads the SSPSR register. Then the SCL pin should be enabled by setting the CKP bit.

Figure 19-10: I²C Waveforms for Transmission (7-bit Address)

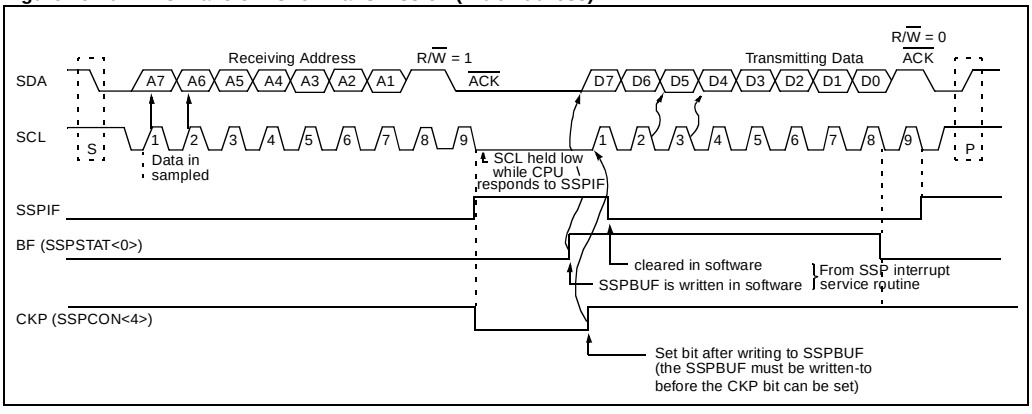
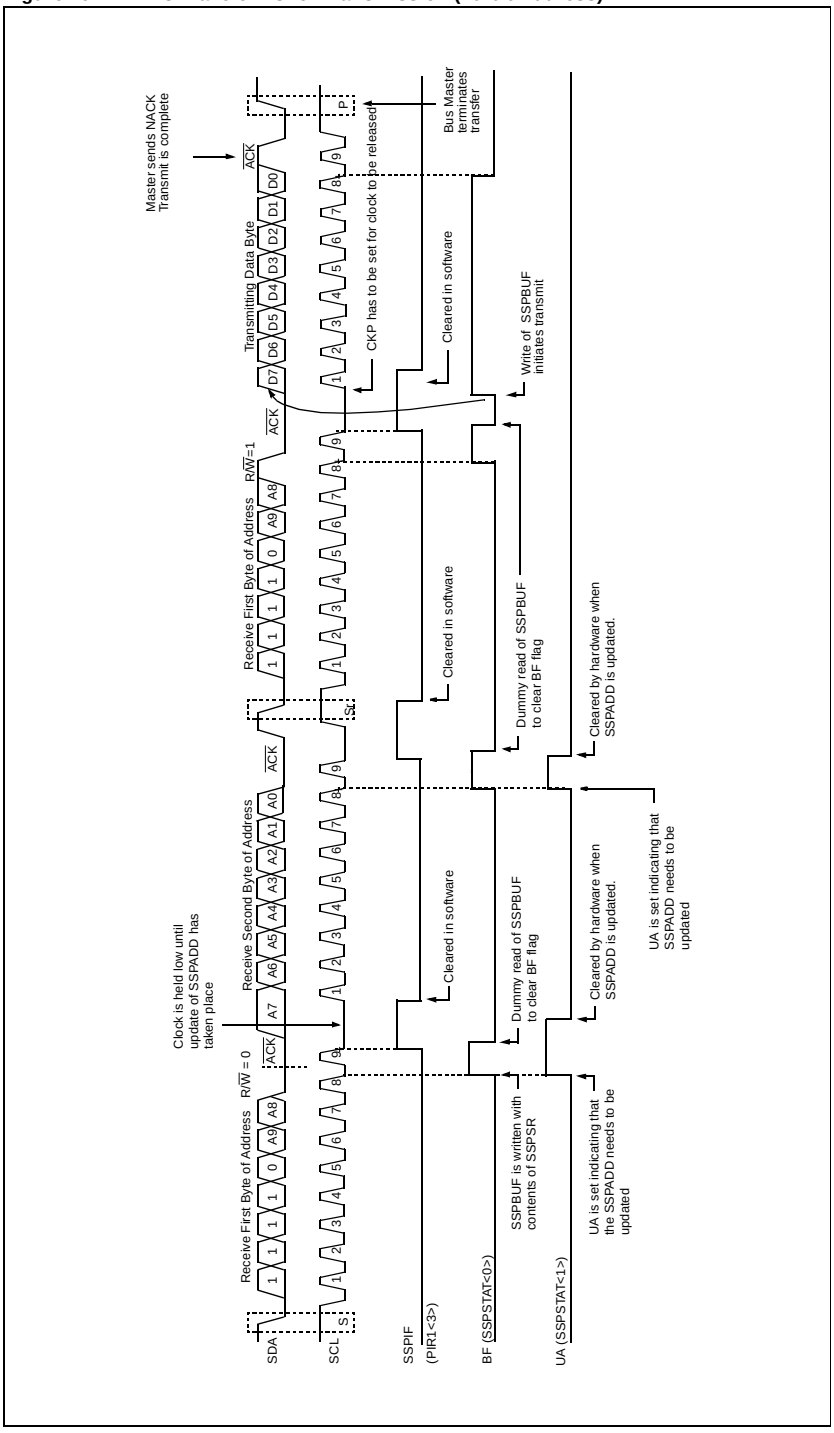


Figure 19-11: I²C Waveforms for Transmission (10-bit Address)



19.4.1.4 Clock Arbitration

Clock arbitration has the SCL pin to inhibit the master device from sending the next clock pulse. The SSP module in I^2C slave mode will hold the SCL pin low when the CPU needs to respond to the SSP interrupt (SSPIF bit is set and the CKP bit is cleared). The data that needs to be transmitted will need to be written to the SSPBUF register, and then the CKP bit will need to be set to allow the master to generate the required clocks.

19.4.2 Master Mode (Firmware)

Master mode of operation is supported by interrupt generation on the detection of the START and STOP conditions. The STOP (P) and START (S) bits are cleared from a reset or when the SSP module is disabled. Control of the I^2C bus may be taken when the P bit is set, or the bus is idle with both the S and P bits clear.

In master mode the SCL and SDA lines are manipulated by clearing the corresponding TRIS bit(s). The output level is always low, irrespective of the value(s) in the PORT register. So when transmitting data, a '1' data bit must have it's TRIS bit set (input) and a '0' data bit must have it's TRIS bit cleared (output). The same scenario is true for the SCL line with the TRIS bit.

The following events will cause SSP Interrupt Flag bit, SSPIF, to be set (SSP Interrupt if enabled):

- START condition
- STOP condition
- Data transfer byte transmitted/received

Master mode of operation can be done with either the slave mode idle (SSPM3:SSPM0 = 1011) or with the slave active (SSPM3:SSP0 = 1110 or 1111). When the slave modes are enabled, the software needs to differentiate the source(s) of the interrupt.

19.4.3 Multi-Master Mode (Firmware)

In multi-Master mode, the interrupt generation on the detection of the START and STOP conditions allows the determination of when the bus is free. The STOP (P) and START (S) bits are cleared from a reset or when the SSP module is disabled. Control of the I^2C bus may be taken when the P bit (SSPSTAT<4>) is set, or the bus is idle with both the S and P bits clear. When the bus is busy, enabling the SSP Interrupt will generate the interrupt when the STOP condition occurs.

In Multi-Master operation, the SDA line must be monitored to see if the signal level is the expected output level. This check only needs to be done when a high level is output. If a high level is expected and a low level is present, the device needs to release the SDA and SCL lines (set the TRIS bits). There are two stages where this arbitration can be lost, they are:

- Address transfer
- Data transfer

When the slave logic is enabled, the slave continues to receive. If arbitration was lost during the address transfer stage, communication to the device may be in progress. If addressed an $\overline{\text{ACK}}$ pulse will be generated. If arbitration was lost during the data transfer stage, the device will need to retransfer the data at a later time.

19.4.4 Sleep Operation

While in sleep mode, the I^2C module can receive addresses or data, and when an address match or complete byte transfer occurs wake the processor from sleep (if the SSP interrupt is enabled).

19.4.5 Effect of a Reset

A reset disables the SSP module and terminates the current transfer.

Table 19-3: Registers Associated with I^2C Operation

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other resets
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR	SSPIF (1)								0	0
IPR	SSPIP (1)								0	0
SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	0000 0000
SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	0000 0000

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'.
Shaded cells are not used by SSP in I^2C mode.

Note 1: The positions of these bits are device dependent.

19.5 Initialization

Example 19-2: SPI Master Mode Initialization

```
    CLRF   SSPSTAT      ; SMP = 0, CKE = 0, and clear status bits
    BSF    SSPSTAT, CKE ; CKE = 1
    MOVLW  0x31         ; Set up SPI port, Master mode, CLK/16,
    MOVWF  SSPCON       ;   Data xmit on falling edge (CKE=1 & CKP=1)
                        ;   Data sampled in middle (SMP=0 & Master mode)
    BSF    PIE, SSPIE   ; Enable SSP interrupt
    BSF    INTCON, GIE  ; Enable, enabled interrupts
    MOVLW  DataByte     ; Data to be Transmitted
                        ;   Could move data from RAM location
    MOVWF  SSPBUF       ; Start Transmission
```

19.5.1 SSP Module / Basic SSP Module Compatibility

When upgrading from the Mid-Range family's basic SSP module, the SSPSTAT register contains two additional control bits. These bits are used only in SPI mode and are:

- SMP, SPI data input sample phase
- CKE, SPI Clock Edge Select

To be compatible with the SPI of the basic SSP module, these bits must be appropriately configured. If these bits are not at the states shown in Table 19-4, improper SPI communication may occur.

Table 19-4: New Bit States for Compatibility

Mid-Range Family's Basic SSP Module	SSP Module		
CKP	CKP	CKE	SMP
1	1	0	0
0	0	0	0

19.6 Design Tips

Question 1: ***Using SPI mode, I do not seem able to talk to an SPI device.***

Answer 1:

Ensure that you are using the correct SPI mode for that device. This SPI supports all four SPI modes so you should be able to get it to function. Check the clock polarity and the clock phase. These settings should match what the SPI is interfacing to.

Question 2: ***Using I^2C mode, I do not seem able to make the master mode work.***

Answer 2:

This SSP module does not have master mode fully automated in hardware, see Application Note AN578 for software which uses the SSP module to implement master mode. If you require a fully automated hardware implementation of I^2C Master Mode, please refer to the Microchip Line Card for devices that have the Master SSP module.

Question 3: ***Using I^2C mode, I write data to the SSPBUF register, but the data did not transmit.***

Answer 3:

Ensure that you set the CKP bit to release the I^2C clock.

19.7 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced MCU family (that is they may be written for the Base-Line, Mid-Range or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the SSP Module are:

Title	Application Note #
Use of the SSP Module in the I ^{2}C Multi-Master Environment.	AN578
Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports	AN613
Software Implementation of I^{2}C Bus Master	AN554
Use of the SSP module in the Multi-master Environment	AN578
Interfacing PIC16C64/74 to Microchip SPI Serial EEPROM	AN647
Interfacing a Microchip PIC16C92x to Microchip SPI Serial EEPROM	AN668

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

19.8 Revision History

Revision A

This is the initial released revision of the SSP module description.

Section 20. Master SSP

HIGHLIGHTS

This section of the manual contains the following major topics:

I²C is a trademark of Philips

20.1 Introduction

The Master Synchronous Serial Port (MSSP) module is a serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, Shift Registers, display drivers, A/D converters, etc. The MSSP module can operate in one of two modes:

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I^2C)
  - Full Master Mode
  - Slave Mode (with general address call)

The I^2C interface supports the following modes in hardware:

- Master Mode
- Multi-Master Mode
- Slave Mode

Figure 20-1 shows a block diagram for the SPI Mode, while Figure 20-2 and Figure 20-3 show the block diagrams for the two different I^2C Modes of operation.

Figure 20-1: SPI Mode Block Diagram

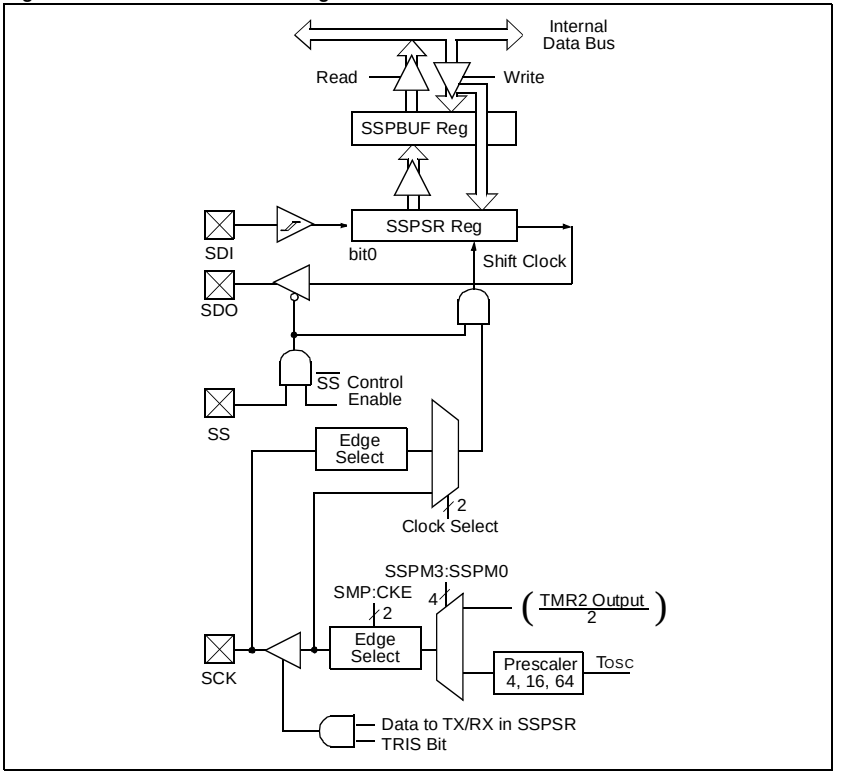


Figure 20-2: I^2C Slave Mode Block Diagram

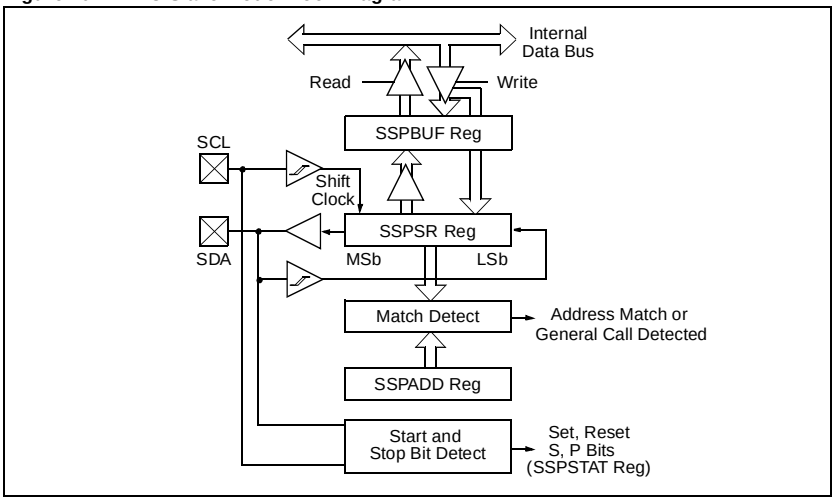
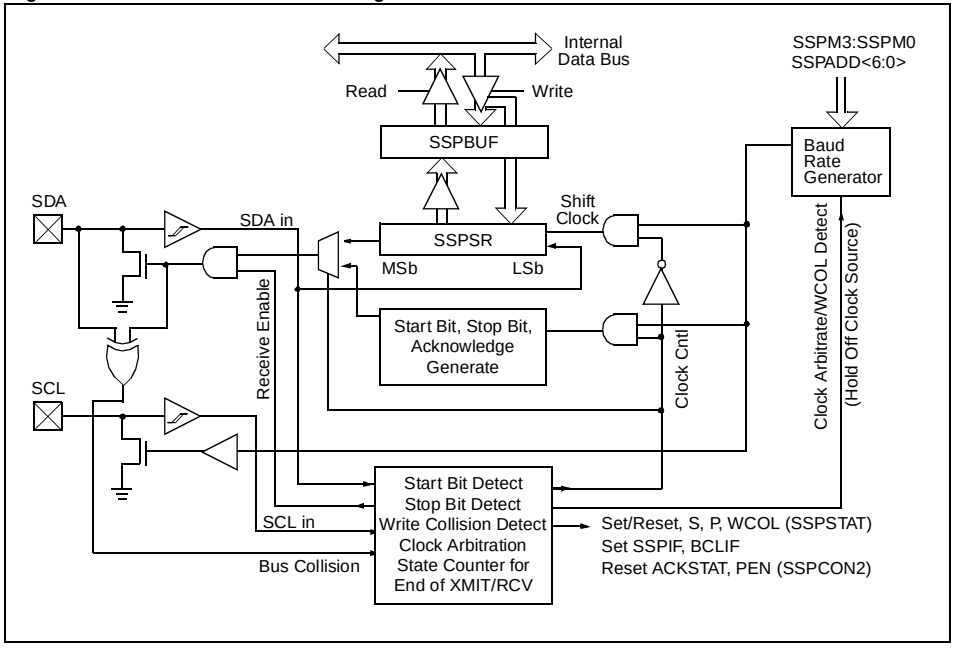


Figure 20-3: I^2C Master Mode Block Diagram



20.2 Control Registers

The Master SSP (MSSP) module has three registers that control the operation and indicate the status of the module. These are the SSPSTAT register (Register 20-1), the SSPCON1register (Register 20-2), and the SSPCON2 register (Register 20-3).

Register 20-1: SSPSTAT: MSSP Status Register

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** Sample bit

SPI Master Mode

1 = Input data sampled at end of data output time
0 = Input data sampled at middle of data output time

SPI Slave Mode

SMP must be cleared when SPI is used in Slave Mode

In I^2C Master or Slave Mode:

1= Slew rate control disabled for standard speed mode (100 kHz and 1 MHz)
0= Slew rate control enabled for high speed mode (400 kHz)

bit 6 **CKE:** SPI Clock Edge Select

CKP = 0
1 = Data transmitted on rising edge of SCK
0 = Data transmitted on falling edge of SCK

CKP = 1
1 = Data transmitted on falling edge of SCK
0 = Data transmitted on rising edge of SCK

bit 5 **D/$\overline{A}$:** Data/$\overline{\text{Address}}$ bit (I^2C Mode only)

1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P:** Stop bit
(I^2C Mode only. This bit is cleared when the MSSP module is disabled, SSPEN is cleared)

1 = Indicates that a Stop bit has been detected last (this bit is '0' on RESET)
0 = Stop bit was not detected last

bit 3 **S:** Start bit
(I^2C Mode only. This bit is cleared when the MSSP module is disabled, SSPEN is cleared)

1 = Indicates that a Start bit has been detected last (this bit is '0' on RESET)
0 = Start bit was not detected last

bit 2 **R/$\overline{W}$:** Read/Write bit information (I^2C Mode only)

This bit holds the R/W bit information following the last address match. This bit is only valid from the address match to the next Start bit, Stop bit, or not $\overline{\text{ACK}}$ bit.

In I^2C Slave Mode:
1 = Read
0 = Write

In I^2C Master Mode:
1 = Transmit is in progress
0 = Transmit is not in progress.
Or'ing this bit with SEN, RSEN, PEN, RCEN, or ACKEN will indicate if the MSSP is in idle Mode.

bit 1 **UA:** Update Address (10-bit I^2C mode only)

1 = Indicates that the user needs to update the address in the SSPADD Register
0 = Address does not need to be updated

bit 0 **BF:** Buffer Full Status bit

Receive (SPI and I^2C Modes)
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty

Transmit (I^2C Mode only)
1 = Data transmit in progress (does not include the $\overline{ACK}$ and Stop bits), SSPBUF is full
0 = Data transmit complete (does not include the $\overline{ACK}$ and Stop bits), SSPBUF is empty

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleard	x = bit is unknown

Register 20-2: SSPCON1: MSSP Control Register1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL:** Write Collision Detect bit

Master Mode:
1 = A write to the SSPBUF Register was attempted while the I^2C conditions were not valid for a transmission to be started
0 = No collision

Slave Mode:
1 = The SSPBUF Register is written while it is still transmitting the previous word (must be cleared in software)
0 = No collision

bit 6 **SSPOV:** Receive Overflow Indicator bit

In SPI Mode:

1 = A new byte is received while the SSPBUF Register is still holding the previous data. In case of overflow, the data in SSPSR is lost. Overflow can only occur in Slave Mode. In Slave Mode, the user must read the SSPBUF, even if only transmitting data, to avoid setting overflow. In Master Mode the overflow bit is not set since each new reception (and transmission) is initiated by writing to the SSPBUF Register. (Must be cleared in software)

0 = No overflow

In I^2C Mode:

1 = A byte is received while the SSPBUF Register is still holding the previous byte. SSPOV is a "don't care" in transmit mode. (Must be cleared in software)

0 = No overflow

bit 5 **SSPEN:** Synchronous Serial Port Enable bit
In both modes, when enabled, the I/O pins must be properly configured as input or output.

In SPI Mode:
1 = Enables serial port and configures SCK, SDO, SDI, and $\overline{SS}$ as the source of the serial port pins
0 = Disables serial port and configures these pins as I/O port pins

In I^2C Mode:
1 = Enables the serial port and configures the SDA and SCL pins as the source of the serial port pins
0 = Disables serial port and configures these pins as I/O port pins

bit 4 **CKP:** Clock Polarity Select bit

In SPI Mode:
1 = Idle state for clock is a high level
0 = Idle state for clock is a low level

In I^2C Slave Mode:
SCK release control

1 = Enable clock
0 = Holds clock low (clock stretch) (Used to ensure data setup time)

In I^2C Master Mode
Unused in this mode

bit 3 - 0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits

0000 = SPI Master Mode, clock = FOSC/4
0001 = SPI Master Mode, clock = FOSC/16
0010 = SPI Master Mode, clock = FOSC/64
0011 = SPI Master Mode, clock = TMR2 output/2
0100 = SPI Slave Mode, clock = SCK pin. $\overline{SS}$ pin control enabled.
0101 = SPI Slave Mode, clock = SCK pin. $\overline{SS}$ pin control disabled. $\overline{SS}$ can be used as I/O pin
0110 = I^2C Slave Mode, 7-bit address
0111 = I^2C Slave Mode, 10-bit address
1000 = I^2C Master Mode, clock = FOSC / (4 * (SSPADD+1))
1001 = Reserved
1010 = Reserved
1011 = I^2C firmware controlled master mode (Slave idle)
1100 = Reserved
1101 = Reserved
1110 = I^2C Slave Mode, 7-bit address with Start and Stop bit interrupts enabled
1111 = I^2C Slave Mode, 10-bit address with Start and Stop bit interrupts enabled

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleard	x = bit is unknown

Register 20-3: SSPCON2: MSSP Control Register2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

bit 7 **GCEN:** General Call Enable bit (In I^2C Slave Mode only)
1 = Enable interrupt when a general call address (0000h) is received in the SSPSR
0 = General call address disabled

bit 6 **ACKSTAT:** Acknowledge Status bit (In I^2C Master Mode only)
In Master Transmit Mode:
1 = Acknowledge was not received from slave
0 = Acknowledge was received from slave

bit 5 **ACKDT:** Acknowledge Data bit (In I^2C Master Mode only)
In Master Receive Mode:
Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.
1 = Not Acknowledge
0 = Acknowledge

bit 4 **ACKEN:** Acknowledge Sequence Enable bit (In I^2C Master Mode only)
In Master Receive Mode:
1 = Initiate Acknowledge sequence on SDA and SCL pins, and transmit ACKDT data bit. Automatically cleared by hardware.
0 = Acknowledge sequence idle

bit 3 **RCEN:** Receive Enable bit (In I^2C Master Mode only)
1 = Enables Receive mode for I^2C
0 = Receive idle

bit 2 **PEN:** Stop condition enable bit (In I^2C Master Mode only)
SCK release control
1 = Initiate Stop condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Stop condition idle

bit 1 **RSEN:** Repeated Start condition enabled bit (In I^2C Master Mode only)
1 = Initiate Repeated Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Repeated Start condition idle.

bit 0 **SEN:** Start condition enabled bit (In I^2C Master Mode only)
1 = Initiate Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Start condition idle

Note: For the ACKEN, RCEN, PEN, RSEN, SEN bits: If the I^2C module is not in the idle mode, the bit may not be set (no spooling) and the SSPBUF may not be written (writes to the SSPBUF are disabled).

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleard	x = bit is unknown

20.3 SPI Mode

The SPI mode allows 8 bits of data to be synchronously transmitted and received simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

- Serial Data Out (SDO)
- Serial Data In (SDI)
- Serial Clock (SCK)

Additionally a fourth pin may be used when in a Slave Mode of operation:

- Slave Select ($\overline{SS}$)

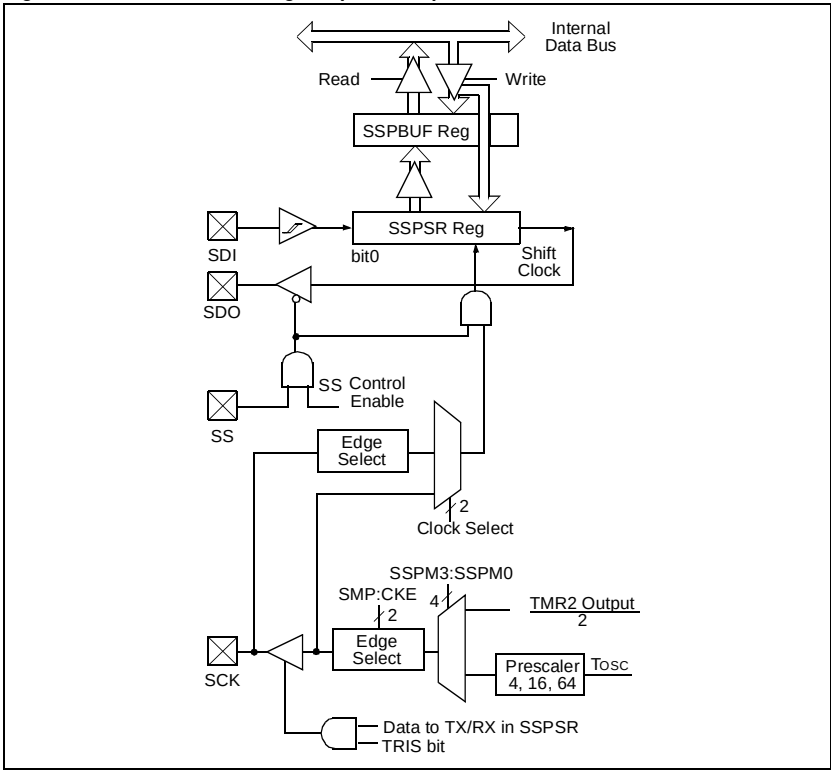
20.3.1 Operation

When initializing the SPI, several options need to be specified. This is done by programming the appropriate control bits (SSPCON1<5:0>) and SSPSTAT<7:6>. These control bits allow the following to be specified:

- Master Mode (SCK is the clock output)
- Slave Mode (SCK is the clock input)
- Clock Polarity (Idle state of SCK)
- Data input sample phase (middle or end of data output time)
- Clock edge (output data on rising/falling edge of SCK)
- Clock Rate (Master Mode only)
- Slave Select Mode (Slave Mode only)

Figure 20-4 shows the block diagram of the MSSP module, when in SPI mode.

Figure 20-4: MSSP Block Diagram (SPI Mode)



The MSSP consists of a transmit/receive Shift Register (SSPSR) and a Buffer Register (SSPBUF). The SSPSR shifts the data in and out of the device, MSb first. The SSPBUF holds the data that was written to the SSPSR, until the received data is ready. Once the 8 bits of data have been received, that byte is moved to the SSPBUF Register. Then the buffer full detect bit, BF (SSPSTAT register), and the interrupt flag bit, SSPIF, are set. This double buffering of the received data (SSPBUF) allows the next byte to start reception before reading the data that was just received. Any write to the SSPBUF Register during transmission/reception of data will be ignored, and the write collision detect bit, WCOL (SSPCON1 register), will be set. User software must clear the WCOL bit so that it can be determined if the following write(s) to the SSPBUF Register completed successfully.

When the application software is expecting to receive valid data, the SSPBUF should be read before the next byte of data to transfer is written to the SSPBUF. Buffer full bit, BF (SSPSTAT register), indicates when SSPBUF has been loaded with the received data (transmission is complete). When the SSPBUF is read, the BF bit is cleared. This data may be irrelevant if the SPI is only a transmitter. Generally the MSSP Interrupt is used to determine when the transmission/reception has completed. The SSPBUF must be read and/or written. If the interrupt method is not going to be used, then software polling can be done to ensure that a write collision does not occur. Example 20-1 shows the loading of the SSPBUF (SSPSR) for data transmission.

Example 20-1:Loading the SSPBUF (SSPSR) Register

```
LOOP BTFSS SSPSTAT, BF    ;Has data been received (transmit complete)?
     GOTO  LOOP           ;No
     MOVF  SSPBUF, W      ;WREG reg = contents of SSPBUF
     MOVWF RXDATA         ;Save in user RAM, if data is meaningful
     MOVF  TXDATA, W      ;W reg = contents of TXDATA
     MOVWF SSPBUF         ;New data to xmit
```

The SSPSR is not directly readable or writable, and can only be accessed by addressing the SSPBUF Register. Additionally, the MSSP Status Register (SSPSTAT) indicates the various status conditions.

20.3.2 Enabling SPI I/O

To enable the serial port, SSP Enable bit, SSPEN, must be set. To reset or reconfigure SPI mode, clear the SSPEN bit, re-initialize the SSPCON Registers, and then set the SSPEN bit. This configures the SDI, SDO, SCK, and $\overline{SS}$ pins as serial port pins. For the pins to behave as the serial port function, some must have their data direction bits (in the TRIS Register) appropriately programmed. That is:

- SDI is automatically controlled by the SPI module
- SDO must have the TRIS bit cleared
- SCK (Master Mode) must have the TRIS bit cleared
- SCK (Slave Mode) must have the TRIS bit set
- $\overline{SS}$ must have the TRIS bit set

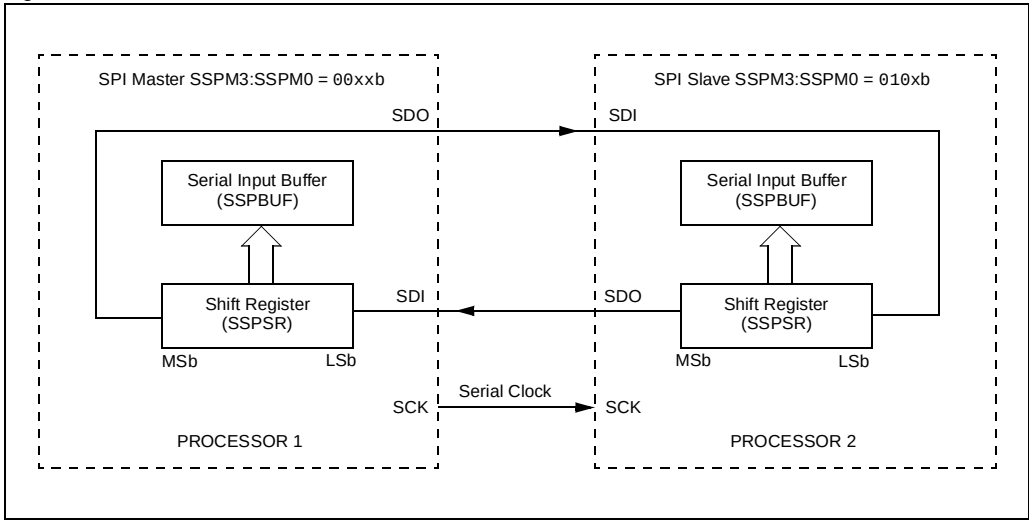
Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) Register to the opposite value.

20.3.3 Typical Connection

Figure 20-5 shows a typical connection between two microcontrollers. The master controller (Processor 1) initiates the data transfer by sending the SCK signal. Data is shifted out of both Shift Registers on their programmed clock edge, and latched on the opposite edge of the clock. Both processors should be programmed to same Clock Polarity (CKP), then both controllers would send and receive data at the same time. Whether the data is meaningful (or dummy data) depends on the application software. This leads to three scenarios for data transmission:

- Master sends data — Slave sends dummy data
- Master sends data — Slave sends data
- Master sends dummy data — Slave sends data

Figure 20-5:SPI Master/Slave Connection



20.3.4 SPI Master Mode

The master can initiate the data transfer at any time because it controls the SCK. The master determines when the slave (Processor 2, Figure 20-5) is to broadcast data by the software protocol.

In Master Mode, the data is transmitted/received as soon as the SSPBUF Register is written to. If the SPI is only going to receive, the SDO output could be disabled (programmed as an input). The SSPSR Register will continue to shift in the signal present on the SDI pin at the programmed clock rate. As each byte is received, it will be loaded into the SSPBUF Register (interrupts and status bits appropriately set). This could be useful in receiver applications as a “line activity monitor” mode.

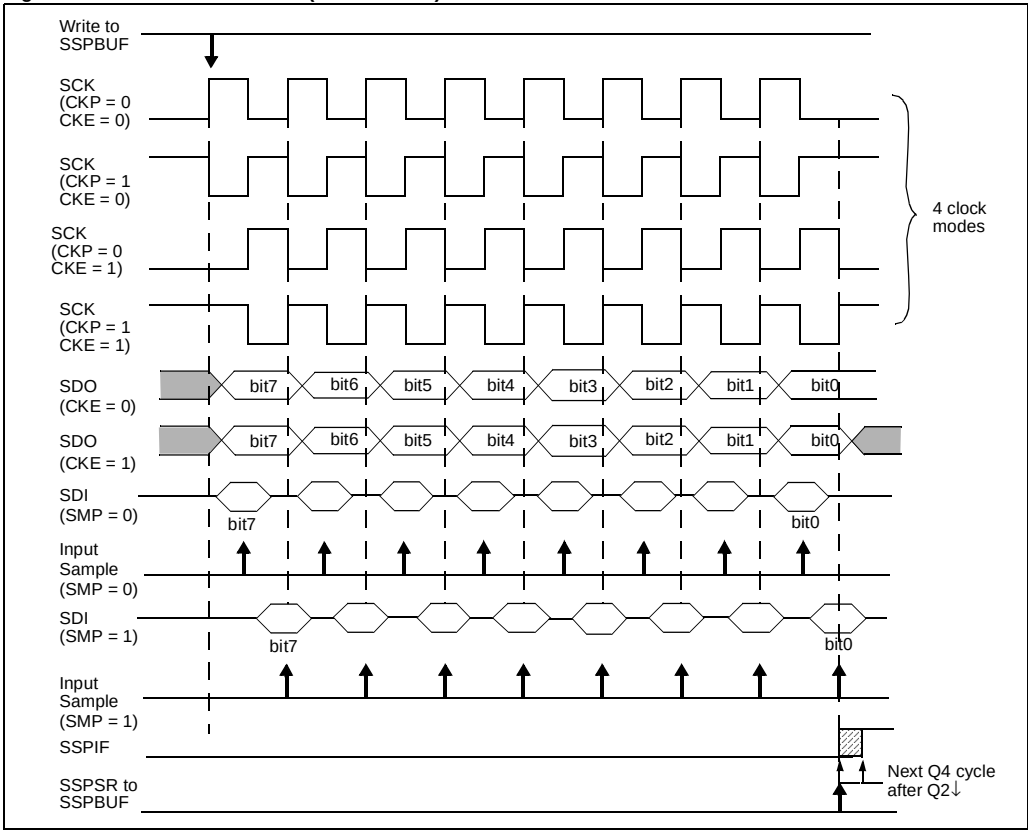
The clock polarity is selected by appropriately programming the CKP bit. This gives waveforms for SPI communication as shown in Figure 20-6, Figure 20-8, and Figure 20-9 where the Msb is transmitted first. In Master Mode, the SPI clock rate (bit rate) is user programmable to be one of the following:

- FOSC/4 (or TCY)
- FOSC/16 (or 4 • TCY)
- FOSC/64 (or 16 • TCY)
- Timer2 output/2

This allows a maximum data rate (at 40 MHz) of 10.00 Mbps.

Figure 20-6 shows the waveforms for Master Mode. When the CKE bit is set, the SDO data is valid before there is a clock edge on SCK. The change of the input sample is shown based on the state of the SMP bit. The time when the SSPBUF is loaded with the received data is shown.

Figure 20-6:SPI Mode Waveform (Master Mode)



20.3.5 SPI Slave Mode

In Slave Mode, the data is transmitted and received as the external clock pulses appear on SCK. When the last bit is latched, the SSPIF interrupt flag bit is set.

While in Slave Mode, the external clock is supplied by the external clock source on the SCK pin. This external clock must meet the minimum high and low times as specified in the electrical specifications.

While in sleep mode, the slave can transmit/receive data. When a byte is received, the device will wake-up from sleep.

20.3.6 Slave Select Synchronization

The $\overline{SS}$ pin is a Slave Select pin, and functions similar to a chip select pin. The SPI must be in Slave Mode with $\overline{SS}$ pin control enabled (SSPCON1<3:0> = 04h). The pin must be configured as an input by setting the corresponding TRIS bit. When the $\overline{SS}$ pin is low, transmission and reception are enabled and the SDO pin is driven. When the $\overline{SS}$ pin goes high, the SDO pin is no longer driven, even if in the middle of a transmitted byte, and becomes a floating output. External pull-up/ pull-down resistors may be desirable, depending on the application.

If the TRIS bit is cleared, making the pin an output, and the pin outputs a high , the SPI receive logic (slave mode) will be in reset. It will remain in reset until either the pin outputs a low, or the pin's TRIS bit is set and external circuits pull the pin low.

Note 1: When the SPI is in Slave Mode with $\overline{SS}$ pin control enabled, (SSPCON<3:0> = 0100) the SPI module will reset if the $\overline{SS}$ pin is set to VDD.

Note 2: If the SPI is used in Slave Mode with CKE set, then the $\overline{SS}$ pin control must be enabled.

When the SPI module resets, the bit counter is forced to 0. This can be done by either by forcing the $\overline{SS}$ pin to a high level or clearing the SSPEN bit.

To emulate two-wire communication, the SDO pin can be connected to the SDI pin. When the SPI needs to operate as a receiver, the SDO pin can be configured as an input. This disables transmissions from the SDO. The SDI can always be left as an input (SDI function) since it cannot create a bus conflict.

Figure 20-7:Slave Synchronization Waveform

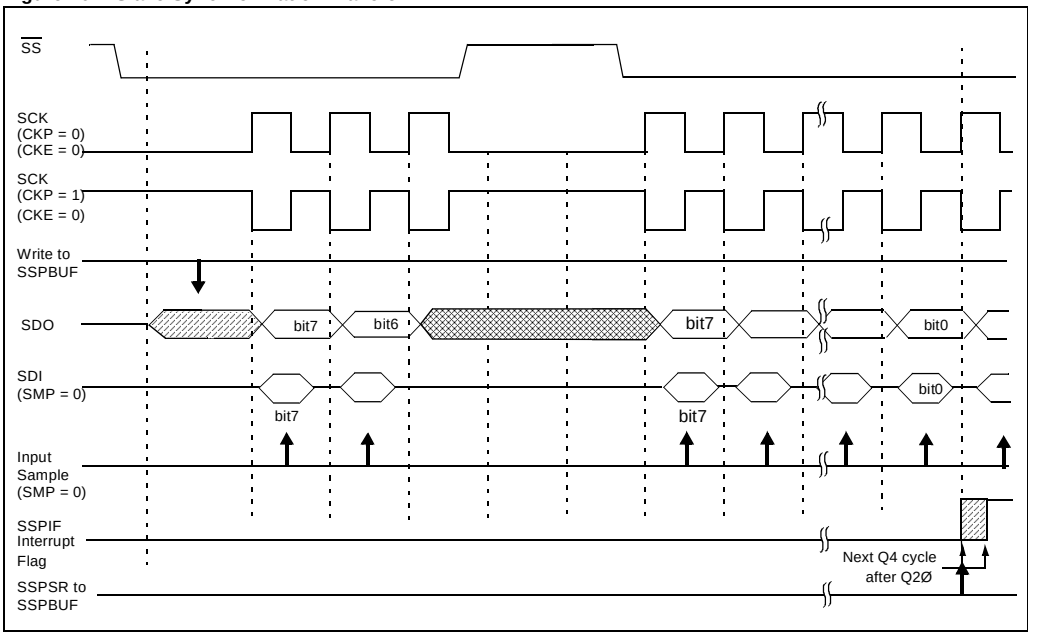


Figure 20-8:SPI Slave Mode Waveform (CKE = 0)

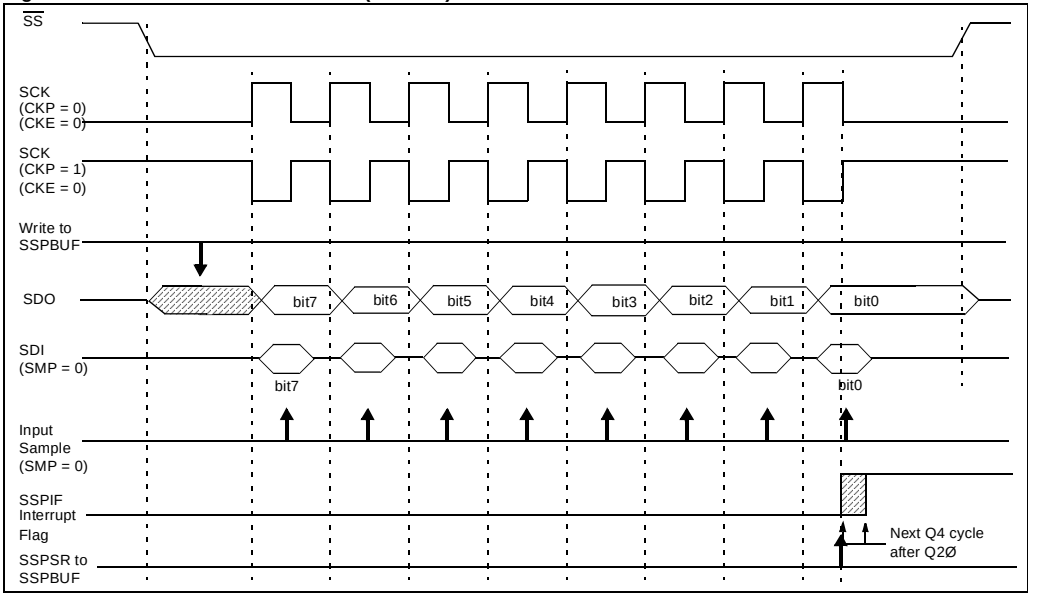
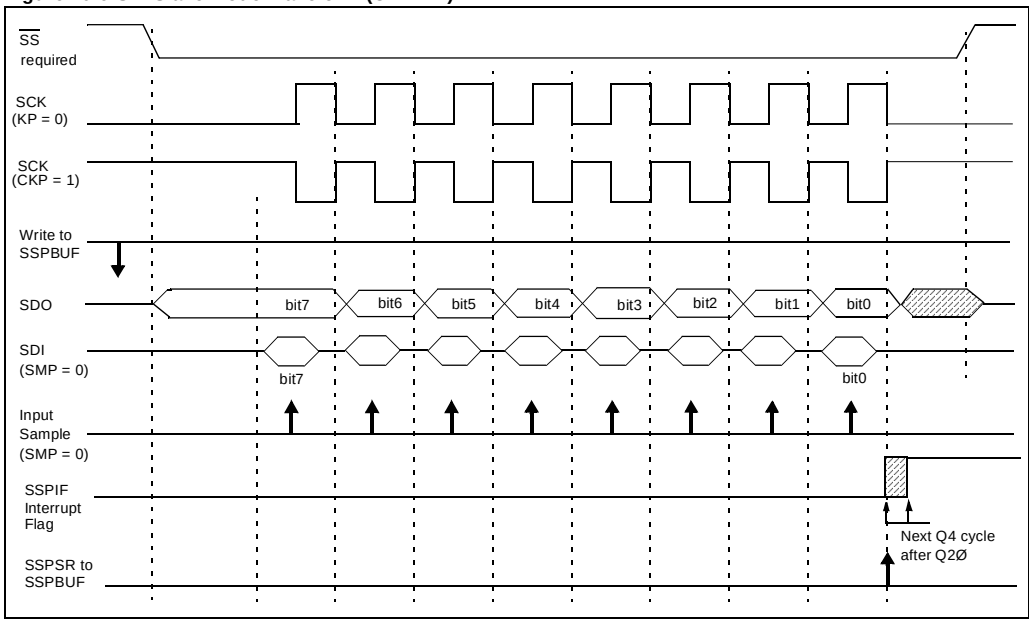


Figure 20-9:SPI Slave Mode Waveform (CKE = 1)



20.3.7 Sleep Operation

In Master Mode, when the SLEEP instruction is executed, all module clocks are halted. The transmission/reception that is in progress will remain in the current state until the device wakes from sleep. After the device returns to normal mode, the module will continue to transmit/receive data.

In Slave Mode, the SPI transmit/receive Shift Register operates asynchronously to the device. This allows the device to be placed in sleep mode, and data to be shifted into the SPI transmit/receive Shift Register. When all 8 bits have been received, the MSSP interrupt flag bit will be set. If the SSPIF is enabled, it will wake the device from sleep.

20.3.8 Effects of a Reset

A reset disables the MSSP module and terminates the current transfer.

20.3.9 Bus Mode Compatibility

Table 20-1 shows the compatibility between the standard SPI modes and the states of the CKP and CKE control bits.

Table 20-1: SPI Bus Modes

Standard SPI Mode Terminology	Control Bits State	
	CKP	CKE
0, 0	0	1
0, 1	0	0
1, 0	1	1
1, 1	1	0

There is also a SMP bit that controls when the data is sampled.

Table 20-2: Registers Associated with SPI Operation

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other resets
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1	PSPIF (1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE (1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP (1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	0000 0000	0000 0000
TRISC	PORTC Data Direction Register								1111 1111	1111 1111
SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
TRISA	—	PORTA Data Direction Register							--11 1111	--11 1111
SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	0000 0000

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'.
Shaded cells are not used by the MSSP in SPI mode.

Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18C2X2 devices. Always maintain these bits clear.

20.4 MSSP I²C Operation

The MSSP module in I²C mode fully implements all master and slave functions (including general call support) and provides interrupts on Start and Stop bits in hardware to determine a free bus (multi-master function). The MSSP module implements the standard mode specifications as well as 7-bit and 10-bit addressing. Appendix A gives an overview of the I²C bus specification.

A "glitch" filter is on the SCL and SDA pins when the pin is an input. This filter operates in both the 100 kHz and 400 kHz modes. In the 100 kHz mode, when these pins are an output, there is a slew rate control of the pin that is independent of device frequency.

Figure 20-10: I²C Slave Mode Block Diagram

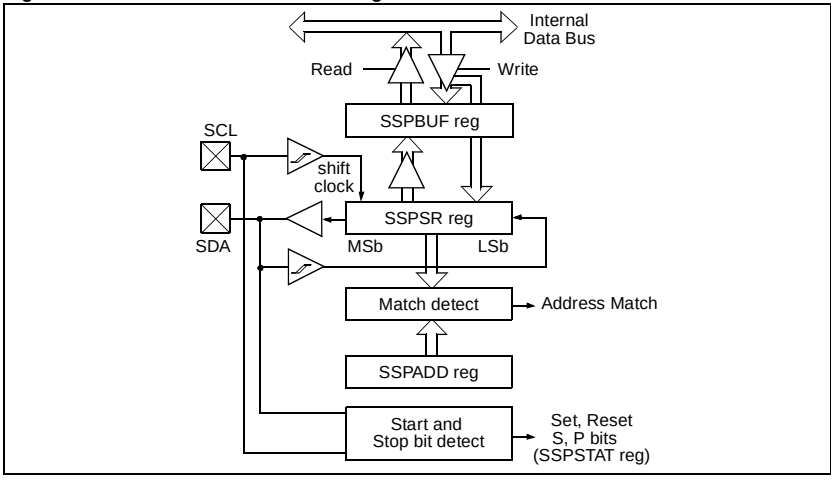
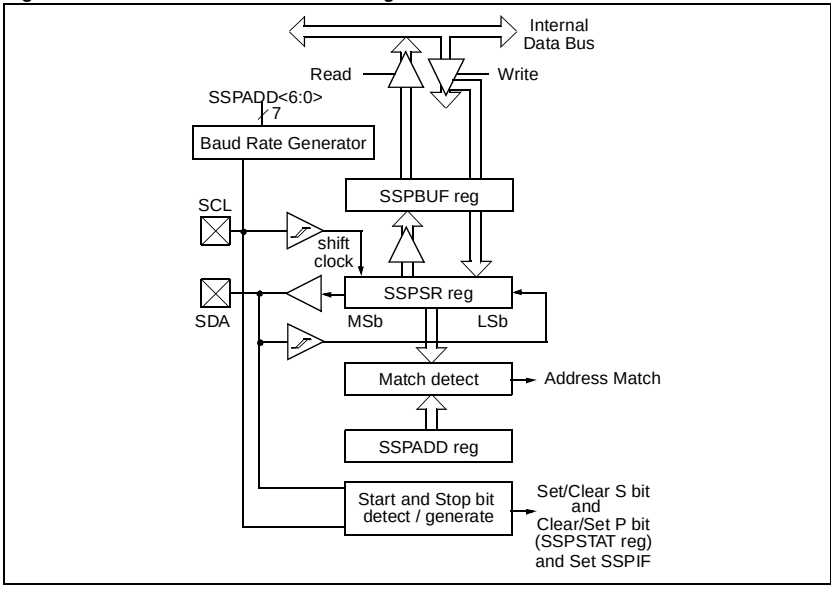


Figure 20-11: I²C Master Mode Block Diagram



Two pins are used for data transfer. These are the SCL pin, which is the clock, and the SDA pin, which is the data. The SDA and SCL pins must be configured as inputs in the corresponding TRIS registers when the I^2C mode is enabled. The MSSP module functions are enabled by setting the MSSP Enable bit, SSPEN (SSPCON register).The MSSP module has six registers for I^2C operation. They are the:

- MSSP Control Register1 (SSPCON1)
- MSSP Control Register2 (SSPCON2)
- MSSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer (SSPBUF)
- MSSP Shift Register (SSPSR) - Not directly accessible
- MSSP Address Register (SSPADD)

The SSPCON1 Register allows control of the I^2C operation. Four mode selection bits (SSPCON1<3:0>) allow one of the following I^2C modes to be selected:

- I^2C Slave Mode (7-bit address)
- I^2C Slave Mode (10-bit address)
- I^2C Master Mode, clock = OSC/4 (SSPADD +1)
- I^2C Slave Mode (7-bit address), with Start and Stop bit interrupts enabled
- I^2C Slave Mode (10-bit address), with Start and Stop bit interrupts enabled
- I^2C Firmware controlled master operation, slave is idle

Before selecting any I^2C mode, the SCL and SDA pins must be programmed to inputs by setting the appropriate TRIS bits. Selecting an I^2C mode, by setting the SSPEN bit, enables the SCL and SDA pins to be used as the clock and data lines in I^2C mode.

The SSPSTAT Register gives the status of the data transfer. This information includes detection of a Start or Stop bit, specifies if the received byte was data or address, if the next byte is the completion of 10-bit address, and if this will be a read or write data transfer.

The SSPBUF is the register to which transfer data is written to or read from. The SSPSR Register shifts the data in or out of the device. In receive operations, the SSPBUF and SSPSR create a double buffered receiver. This allows reception of the next byte to begin before reading the current byte of received data. When the complete byte is received, it is transferred to the SSPBUF Register and the SSPIF bit is set. If another complete byte is received before the SSPBUF Register is read, a receiver overflow has occurred and the SSPOV bit (SSPCON1 register) is set and the byte in the SSPSR is lost.

The SSPADD Register holds the slave address. In 10-bit mode, the user needs to write the high byte of the address (`1111 0 A9 A8 0`). Following the high byte address match, the low byte of the address needs to be loaded (A7:A0).

20.4.1 Slave Mode

In Slave Mode, the SCL and SDA pins must be configured as inputs. The MSSP module will override the input state with the output data when required (slave-transmitter).

When an address is matched or the data transfer after an address match is received, the hardware automatically generates the acknowledge ($\overline{ACK}$) pulse, and loads the SSPBUF Register with the received value currently in the SSPSR Register.

There are certain conditions that will cause the MSSP module not to give this $\overline{ACK}$ pulse. These are if either (or both):

a) The buffer full bit, BF (SSPSTAT register), was set before the transfer was received.

b) The overflow bit, SSPOV (SSPCON1 register), was set before the transfer was received.

If the BF bit is set, the SSPSR Register value is not loaded into the SSPBUF, but the SSPIF and SSPOV bits are set. Table 20-3 shows what happens when a data transfer byte is received, given the status of the BF and SSPOV bits. The shaded cells show the condition where user software did not properly clear the overflow condition. The BF bit is cleared by reading the SSPBUF register while bit SSPOV is cleared through software.

The SCL clock input must have a minimum high and low time for proper operation. The high and low times of the I^2C specification as well as the requirement of the MSSP module is shown in timing parameters 100 and 101 of the **"Electrical Specifications"** section.

20.4.1.1 Addressing

Once the MSSP module has been enabled, it waits for a Start condition to occur. Following the Start condition, the 8 bits are shifted into the SSPSR Register. All incoming bits are sampled with the rising edge of the clock (SCL) line. The value of register SSPSR<7:1> is compared to the value of the SSPADD Register (bits 7:1). The address is compared on the falling edge of the eighth clock (SCL) pulse. If the addresses match, and the BF and SSPOV bits are clear, the following events occur:

a) The SSPSR Register value is loaded into the SSPBUF Register on the falling edge of the eighth SCL pulse.

b) The buffer full bit, BF, is set on the falling edge of the eighth SCL pulse.

c) An $\overline{ACK}$ pulse is generated.

d) MSSP interrupt flag bit, SSPIF, is set (interrupt is generated if enabled) - on the falling edge of the ninth SCL pulse.

In 10-bit address mode, two address bytes need to be received by the slave. The five Most Significant bits (MSbs) of the first address byte specify if this is a 10-bit address. The R/$\overline{W}$ bit (SSPSTAT<2>) must specify a write so the slave device will receive the second address byte. For a 10-bit address the first byte would equal '`1111 0 A9 A8 0`', where A9 and A8 are the two MSbs of the address. The sequence of events for a 10-bit address is as follows (with steps 7- 9 for a slave-transmitter):

1. Receive first (high) byte of the address (the SSPIF, BF, and UA (SSPSTAT register) bits are set).
2. Update the SSPADD Register with second (low) byte of the address (clears the UA bit and releases the SCL line).
3. Read the SSPBUF Register (clears the BF bit) and clear flag bit SSPIF.
4. Receive second (low) byte of the address (the SSPIF, BF, and UA bits are set).
5. Update the SSPADD Register with the first (high) byte of the address. This will clear the UA bit and release the SCL line.
6. Read the SSPBUF Register (clears the BF bit) and clear the SSPIF flag bit.
7. Receive repeated Start condition.
8. Receive first (high) byte of the address (the SSPIF and BF bits are set).
9. Read the SSPBUF Register (clears the BF bit) and clear the SSPIF flag bit.

Note:	Following the Repeated Start condition (step 7) in 10-bit mode, the user only needs to match the first 7-bit address. The user does not update the SSPADD for the second half of the address.

Table 20-3: Data Transfer Received Byte Actions

Status Bits as Data Transfer is Received				
BF	**SSPOV**	**SSPSR → SSPBUF**	**Generate $\overline{ACK}$ Pulse**	**Set bit SSPIF (SSP Interrupt occurs if enabled)**
0	0	Yes	Yes	Yes
1	0	No	No	Yes
1	1	No	No	Yes
0	1	Yes	No	Yes

Note: Shaded cells show the conditions where the user software did not properly clear the overflow condition

20.4.1.2 Slave Reception

When the R/$\overline{W}$ bit of the address byte is clear and an address match occurs, the R/$\overline{W}$ bit of the SSPSTAT Register is cleared. The received address is loaded into the SSPBUF Register.

When the address byte overflow condition exists, then no acknowledge ($\overline{ACK}$) pulse is given. An overflow condition is defined as either the BF bit (SSPSTAT register) is set or the SSPOV bit (SSPCON1 register) is set.

An MSSP interrupt is generated for each data transfer byte. The SSPIF flag bit must be cleared in software. The SSPSTAT Register is used to determine the status of the received byte.

Note:	The SSPBUF will be loaded if the SSPOV bit is set and the BF flag bit is cleared. If a read of the SSPBUF was performed, but the user did not clear the state of the SSPOV bit before the next receive occurred. The ACK is not sent and the SSPBUF is updated.

20.4.1.3 Slave Transmission

When the R/$\overline{W}$ bit of the incoming address byte is set and an address match occurs, the R/$\overline{W}$ bit of the SSPSTAT Register is set. The received address is loaded into the SSPBUF Register. The $\overline{ACK}$ pulse will be sent on the ninth bit, and the SCL pin is held low. The transmit data must be loaded into the SSPBUF Register, which also loads the SSPSR Register and sets the BF bit. Then the SCL pin should be enabled by setting the CKP bit (SSPCON1 register). The master should monitor the SCL pin prior to asserting another clock pulse. The slave devices may be holding off the master by stretching the clock. The eight data bits are shifted out on the falling edge of the SCL input. This ensures that the SDA signal is valid during the SCL high time (Figure 20-13). When all eight bits have been shifted out, the BF bit will be cleared.

An MSSP interrupt is generated for each data transfer byte. The SSPIF flag bit must be cleared in software, and the SSPSTAT Register is used to determine the status of the byte transfer. The SSPIF flag bit is set on the falling edge of the ninth clock pulse.

As a slave-transmitter, the $\overline{ACK}$ pulse from the master-receiver is latched on the rising edge of the ninth SCL input pulse. If the SDA line was high (not $\overline{ACK}$), then the data transfer is complete. When the not $\overline{ACK}$ is latched by the slave, the slave logic is reset and the slave then monitors for another occurrence of the Start bit. If the SDA line was low ($\overline{ACK}$), the transmit data must be loaded into the SSPBUF Register, which also loads the SSPSR Register and sets the BF bit. Then the SCL pin should be enabled by setting the CKP bit.

Figure 20-12: I²C Slave Mode Waveforms for Reception (7-bit Address)

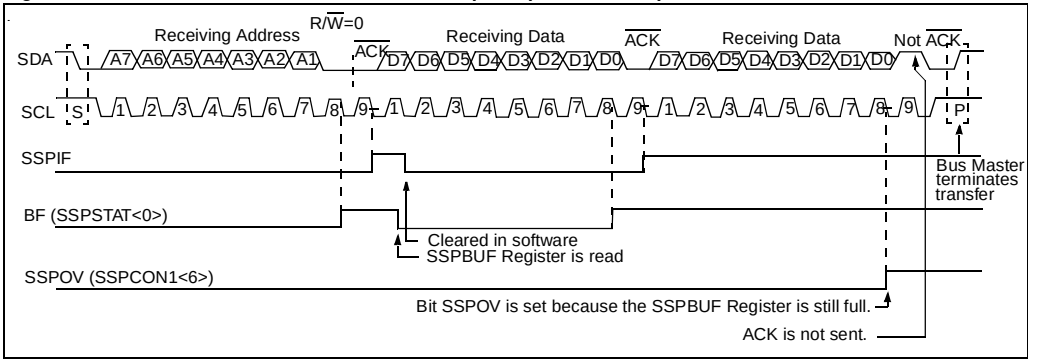


Figure 20-13: I²C Slave Mode Waveforms for Transmission (7-bit Address)

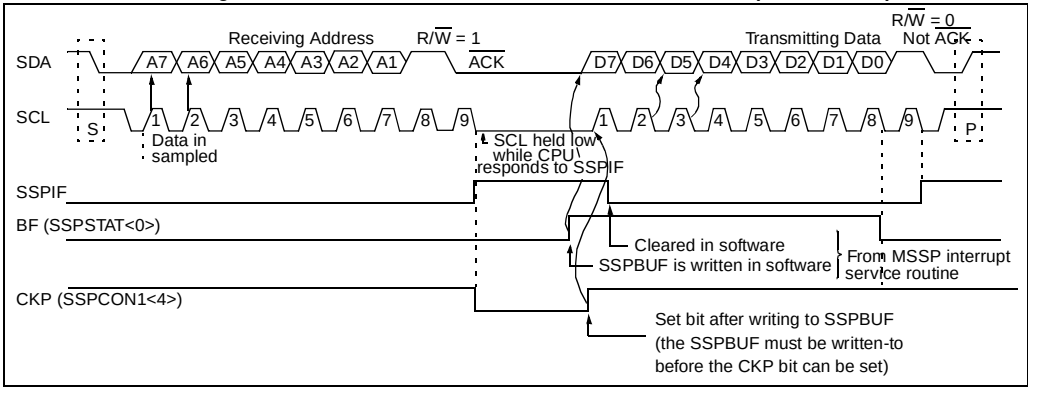


Figure 20-14: I^2C Slave Mode Waveform (Transmission 10-bit Address)

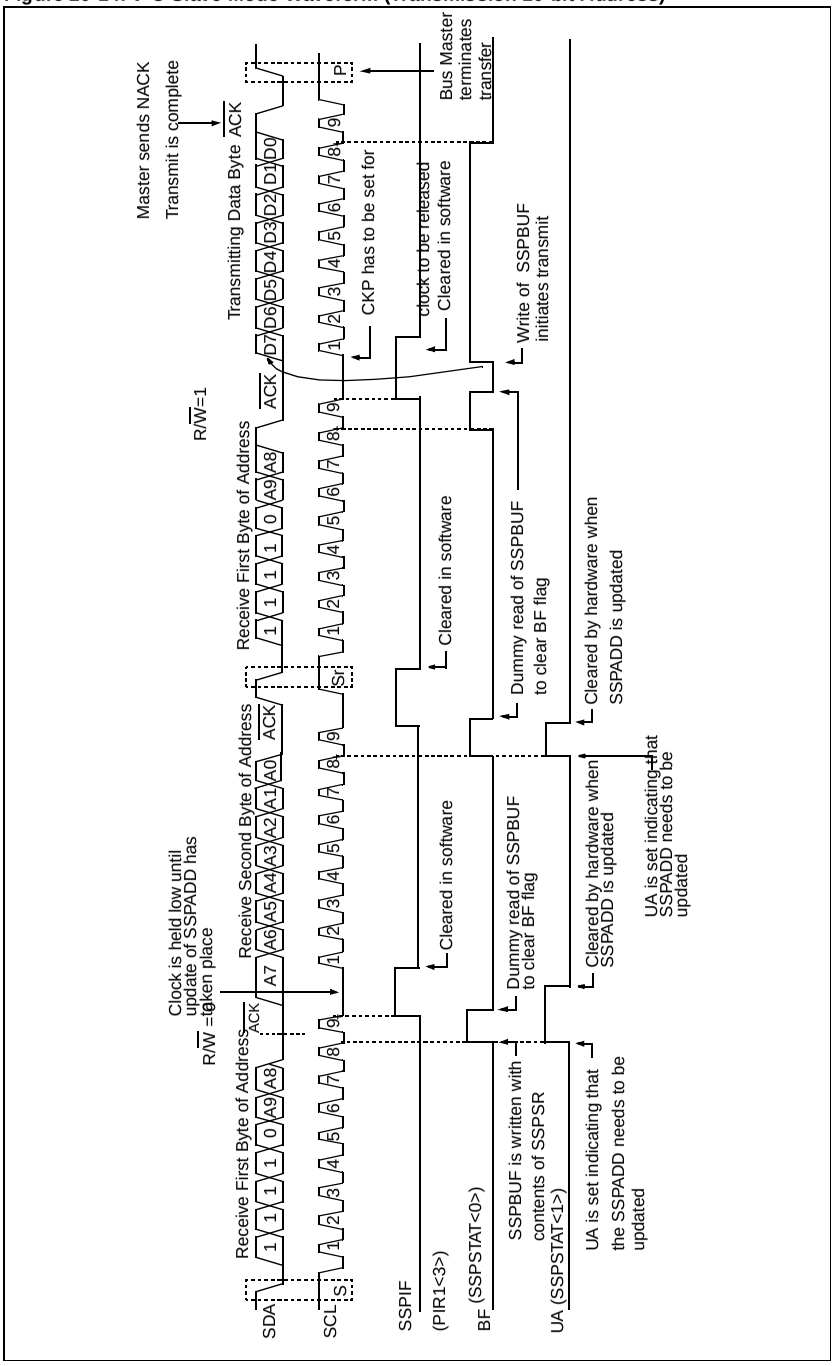
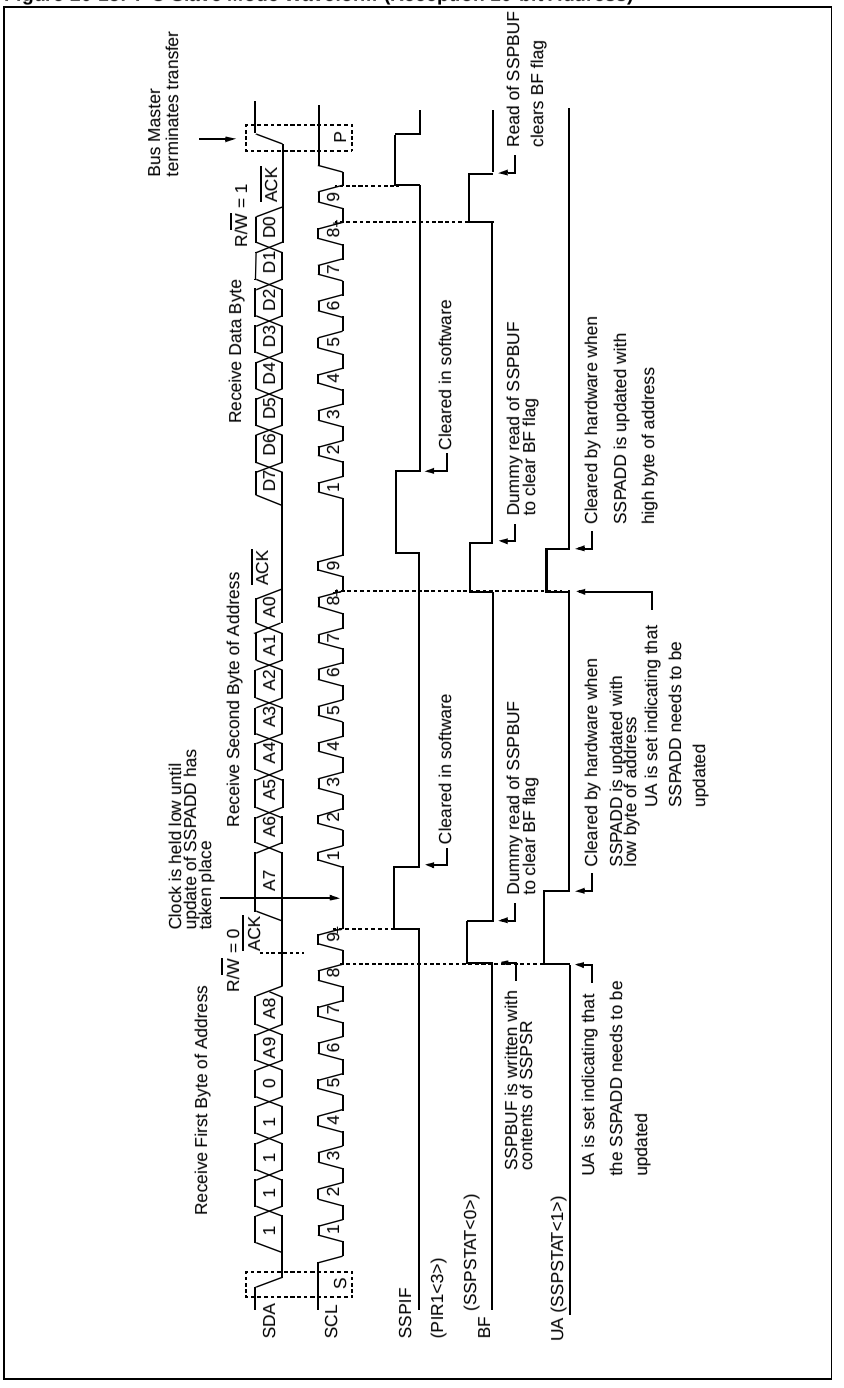


Figure 20-15: I^2C Slave Mode Waveform (Reception 10-bit Address)



20.4.2 General Call Address Support

The addressing procedure for the I^2C bus is such that the first byte after the Start condition usually determines which device will be the slave addressed by the master. The exception is the general call address, which can address all devices. When this address is used, all devices should respond with an acknowledge.

The general call address is one of eight addresses reserved for specific purposes by the I^2C protocol. It consists of all 0's with R/$\overline{W}$ = 0.

The general call address is recognized when the General Call Enable bit (GCEN) is set. Following a Start bit detect, 8 bits are shifted into the SSPSR and the address is compared against the SSPADD, and is also compared to the general call address, fixed in hardware.

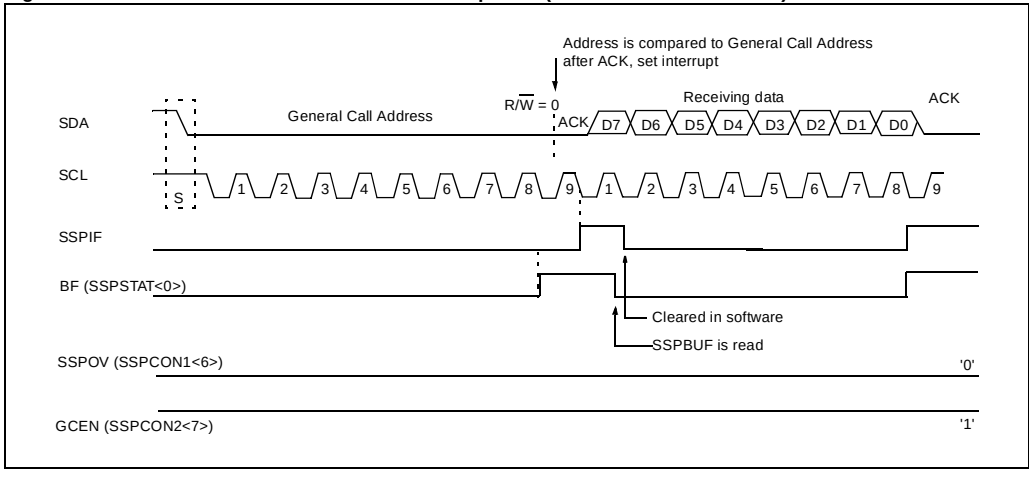
If the general call address matches, the SSPSR is transferred to the SSPBUF, the BF flag bit is set (during the eighth bit), and on the falling edge of the ninth bit (the $\overline{ACK}$ bit) the SSPIF interrupt flag bit is set.

When the interrupt is serviced. The source for the interrupt can be checked by reading the contents of the SSPBUF to determine if the address was device specific or a general call address.

In 10-bit address mode, SSPADD must be updated for the second half of the address to match and the UA bit to be set.

If the general call address is sampled when the GCEN bit is set, then the second half of the address is not necessary. The UA bit will not be set, and the slave (configured in 10-bit address mode) will begin receiving data after the acknowledge (Figure 20-16).

Figure 20-16: Slave Mode General Call Address Sequence (7 or 10-bit Address Mode)



20.4.3 Sleep Operation

While in sleep mode, the I^{2}C module can receive addresses or data. When an address match or complete byte transfer occurs, the processor will wake-up from sleep (if the MSSP interrupt is enabled).

20.4.4 Effect of a Reset

A reset disables the MSSP module and terminates the current transfer.

Table 20-4: Registers Associated with I^{2}C Operation

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other resets
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR	SSPIF, BCLIF (1)								0, 0	0, 0
PIE	SSPIE, BCLIF (1)								0, 0	0, 0
SSPADD	Synchronous Serial Port (I^{2}C mode) Address Register (Slave Mode)/Baud Rate Generator (Master Mode)								0000 0000	0000 0000
SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
SSPCON1	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	0000 0000
SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	0000 0000

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'.
Shaded cells are not used by the MSSP in I^{2}C mode.

Note 1: The position of these bits is device dependent.

20.4.5 Master Mode

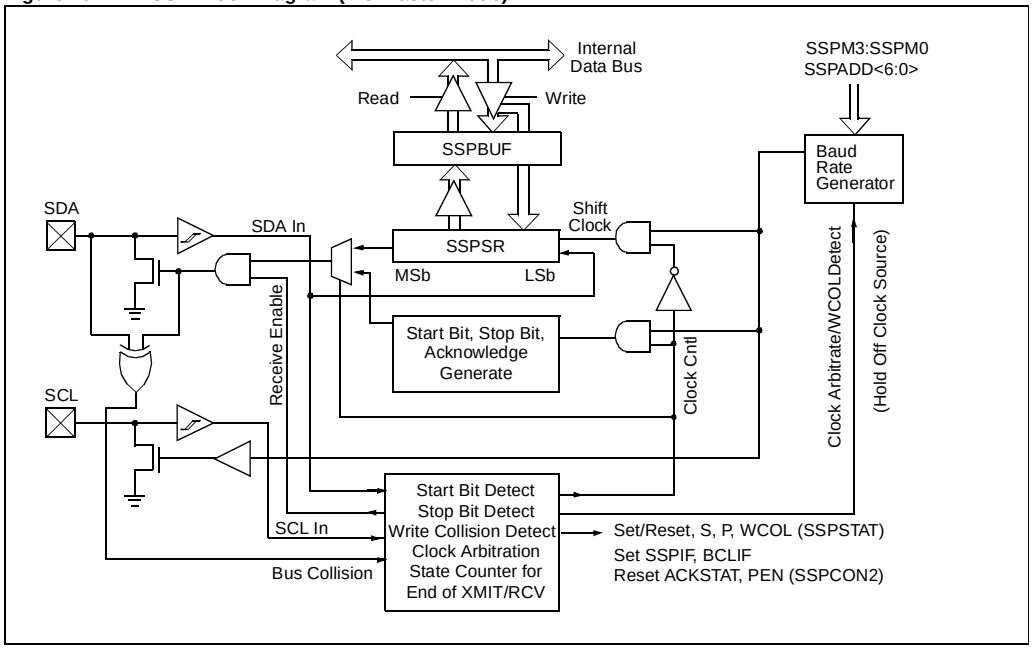
Master Mode of operation is supported by interrupt generation on the detection of the Start and Stop conditions. The Stop (P) and Start (S) bits are cleared when a reset occurs or when the MSSP module is disabled. Control of the I^2C bus may be taken when the P bit is set, or the bus is idle with both the S and P bits clear.

In Master Mode, the SCL and SDA lines are manipulated by the MSSP hardware.

The following events will cause SSP Interrupt Flag bit, SSPIF, to be set (SSP Interrupt if enabled):

- Start condition
- Stop condition
- Data transfer byte transmitted/received
- Acknowledge Transmit
- Repeated Start

Figure 20-17: MSSP Block Diagram (I^2C Master Mode)



20.4.6 Multi-Master Mode

In Multi-Master Mode, the interrupt generation on the detection of the Start and Stop conditions allows the determination of when the bus is free. The Stop (P) and Start (S) bits are cleared from a reset or when the MSSP module is disabled. Control of the I^2C bus may be taken when the P bit (SSPSTAT register) is set, or the bus is idle with both the S and P bits clear. When the bus is busy, enabling the MSSP Interrupt will generate the interrupt when the Stop condition occurs.

In multi-master operation, the SDA line must be monitored, for arbitration, to see if the signal level is the expected output level. This check is performed in hardware, with the result placed in the BCLIF bit.

The states where arbitration can be lost are:

- Address transfer
- Data transfer
- A Start condition
- A Repeated Start condition
- An Acknowledge condition

20.4.7 I^2C Master Mode Support

Master Mode is enabled by setting and clearing the appropriate SSPM bits in SSPCON1 and by setting the SSPEN bit. Once Master Mode is enabled, the user has six options.

1. Assert a Start condition on SDA and SCL.
2. Assert a Repeated Start condition on SDA and SCL.
3. Write to the SSPBUF Register initiating transmission of data/address.
4. Generate a Stop condition on SDA and SCL.
5. Configure the I^2C port to receive data.
6. Generate an acknowledge condition at the end of a received byte of data.

Note:	The MSSP Module when configured in I^2C Master Mode does not allow queueing of events. For instance: The user is not allowed to initiate a Start condition, and immediately write the SSPBUF Register to imitate transmission before the Start condition is complete. In this case the SSPBUF will not be written to, and the WCOL bit will be set, indicating that this write to the SSPBUF did not occur.

20.4.7.1 I²C Master Mode Operation

The master device generates all of the serial clock pulses and the Start and Stop conditions. A transfer is ended with a Stop condition or with a Repeated Start condition. Since the Repeated Start condition is also the beginning of the next serial transfer, the I²C bus will not be released.

In master transmitter mode, serial data is output through SDA, while SCL outputs the serial clock. The first byte transmitted contains the slave address of the receiving device (7 bits), and the Read/Write (R/$\overline{W}$) bit. In this case, the R/$\overline{W}$ bit will be logic '0'. Serial data is transmitted 8 bits at a time. After each byte is transmitted, an Acknowledge bit is received. Start and Stop conditions are output to indicate the beginning and the end of a serial transfer.

In master receive mode, the first byte transmitted contains the slave address of the transmitting device (7 bits), and the R/$\overline{W}$ bit. In this case, the R/$\overline{W}$ bit will be logic '1'. Thus, the first byte transmitted is a 7-bit slave address followed by a '1' to indicate receive bit. Serial data is received via the SDA pin, while the SCL pin outputs the serial clock. Serial data is received 8 bits at a time. After each byte is received, an Acknowledge bit is transmitted. Start and Stop conditions indicate the beginning and end of transmission.

The baud rate generator used for SPI mode operation is now used to set the SCL clock frequency for either 100 kHz, 400 kHz, or 1 MHz I²C operation. The baud rate generator reload value is contained in the lower 7 bits of the SSPADD Register. The baud rate generator will automatically begin counting on a write to the SSPBUF. Once the given operation is complete (i.e., transmission of the last data bit is followed by ACK), the internal clock will automatically stop counting and the SCL pin will remain in its last state.

A typical transmit sequence would go as follows:

a) The user generates a Start condition by setting the Start enable bit, SEN (SSPCON2 register).
b) SSPIF is set. The MSSP module will wait the required start time before any other operation takes place.
c) The user loads the SSPBUF with the address to transmit.
d) Address is shifted out the SDA pin until all 8 bits are transmitted.
e) The MSSP module shifts in the ACK bit from the slave device, and writes its value into the SSPCON2 Register (SSPCON2 register).
f) The MSSP module generates an interrupt at the end of the ninth clock cycle by setting the SSPIF bit.
g) The user loads the SSPBUF with eight bits of data.
h) DATA is shifted out the SDA pin until all 8 bits are transmitted.
i) The MSSP module shifts in the ACK bit from the slave device, and writes its value into the SSPCON2 Register (SSPCON2 register).
j) The MSSP module generates an interrupt at the end of the ninth clock cycle by setting the SSPIF bit.
k) The user generates a Stop condition by setting the Stop enable bit, PEN (SSPCON2 register).
l) Interrupt is generated once the Stop condition is complete.

20.4.8 Baud Rate Generator

In I^2C Master Mode, the reload value for the BRG is located in the lower 7 bits of the SSPADD Register (Figure 20-18). When the BRG is loaded with this value, the BRG counts down to 0 and stops until another reload has taken place. The BRG count is decremented twice per instruction cycle (TCY) on the Q2 and Q4 clocks. In I^2C Master Mode, the BRG is reloaded automatically. If clock arbitration is taking place for instance, the BRG will be reloaded when the SCL pin is sampled high (Figure 20-19).

Figure 20-18: Baud Rate Generator Block Diagram

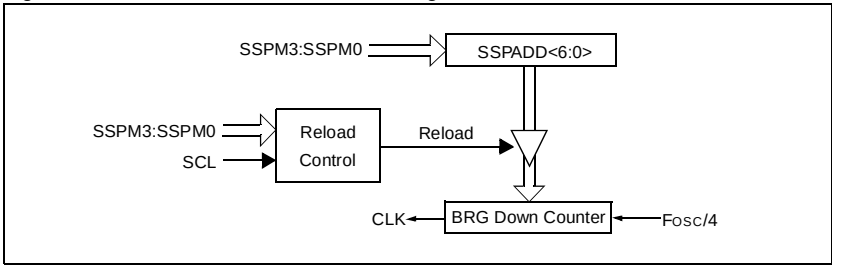
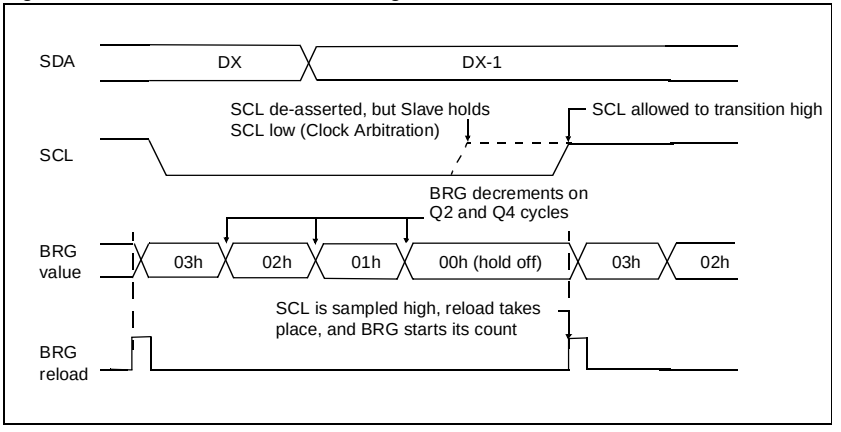


Figure 20-19: Baud Rate Generator Timing With Clock Arbitration



20.4.9 I²C Master Mode Start Condition Timing

To initiate a Start condition, the user sets the Start condition enable bit, SEN (SSPCON2 register). If the SDA and SCL pins are sampled high, the baud rate generator is re-loaded with the contents of SSPADD<6:0>, and starts its count. If the SCL and SDA pins are both sampled high when the baud rate generator times out (TBRG), the SDA pin is driven low. The action of the SDA pin being driven low while the SCL pin is high in the Start condition, and causes the S bit (SSPSTAT register) to be set. Following this, the baud rate generator is reloaded with the contents of SSPADD<6:0> and resumes its count. When the baud rate generator times out (TBRG) the SEN bit (SSPCON2 register) will be automatically cleared by hardware, the baud rate generator is suspended leaving the SDA line held low, and the Start condition is complete.

Note:	If at the beginning of Start condition, the SDA and SCL pins are already sampled low, or if during the Start condition the SCL pin is sampled low before the SDA pin is driven low, a bus collision occurs. The Bus Collision Interrupt Flag, BCLIF, is set, the Start condition is aborted, and the I²C module is reset into its idle state.

20.4.9.1 WCOL Status Flag

If the user writes the SSPBUF when an Start sequence is in progress, then WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

Note:	Because queueing of events is not allowed, writing to the lower 5 bits of SSPCON2 is disabled until the Start condition is complete.

Figure 20-20: First Start Bit Timing

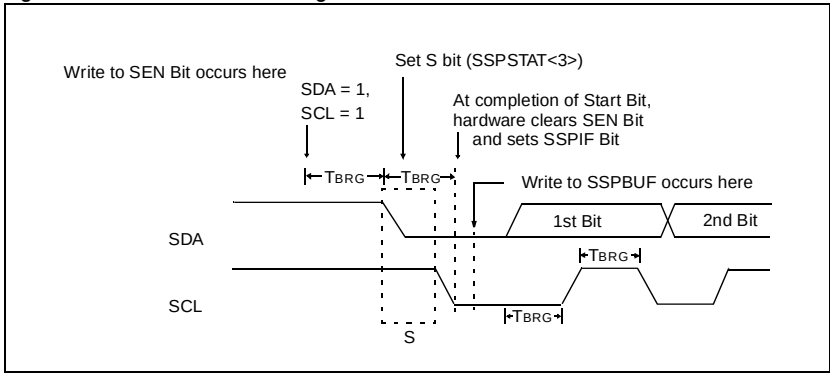
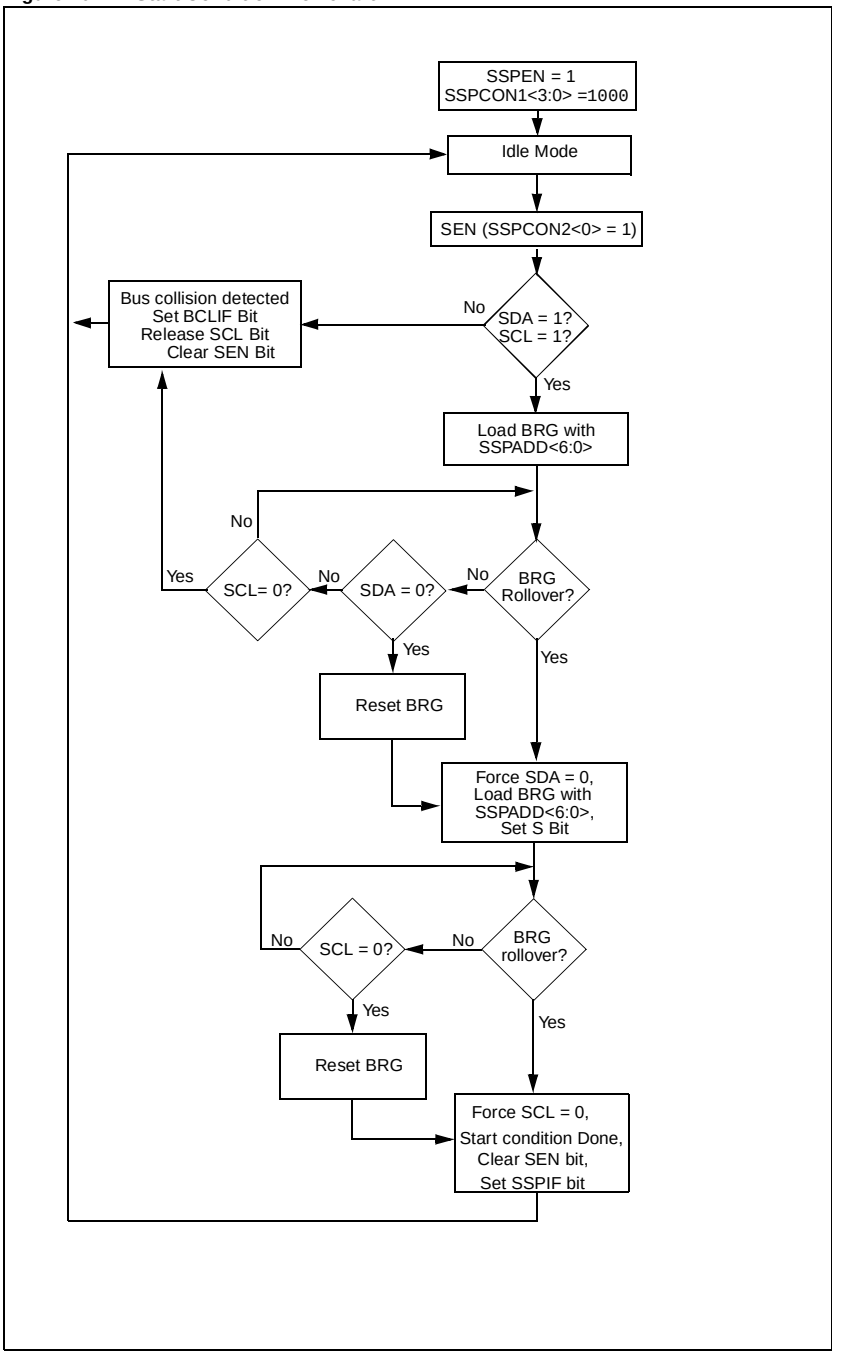


Figure 20-21: Start Condition Flowchart



20.4.10 I²C Master Mode Repeated Start Condition Timing

A Repeated Start condition occurs when the RSEN bit (SSPCON2 register) is programmed high and the I²C logic module is in the idle state. When the RSEN bit is set, the SCL pin is asserted low. When the SCL pin is sampled low, the baud rate generator is loaded with the contents of SSPADD<5:0>, and begins counting. The SDA pin is released (brought high) for one baud rate generator count (TBRG). When the baud rate generator times out, if SDA is sampled high, the SCL pin will be de-asserted (brought high). When the SCL pin is sampled high, the baud rate generator is re-loaded with the contents of SSPADD<6:0> and begins counting. SDA and SCL must be sampled high for one TBRG. This action is then followed by assertion of the SDA pin (SDA = 0) for one TBRG while SCL is high. Following this, the RSEN bit (SSPCON2 register) will be automatically cleared and the baud rate generator is not reloaded, leaving the SDA pin held low. As soon as a Start condition is detected on the SDA and SCL pins, the S bit (SSPSTAT register) will be set. The SSPIF bit will not be set until the baud rate generator has timed-out.

Note 1: If RSEN is programmed while any other event is in progress, it will not take effect.

Note 2: A bus collision during the Repeated Start condition occurs if:

- SDA is sampled low when SCL goes from low to high.
- SCL goes low before SDA is asserted low. This may indicate that another master is attempting to transmit a data "1".

Immediately following the SSPIF bit getting set, the user may write the SSPBUF with the 7-bit address in 7-bit mode, or the default first address in 10-bit mode. After the first eight bits are transmitted and an ACK is received, the user may then transmit an additional eight bits of address (10-bit mode) or eight bits of data (7-bit mode).

20.4.10.1 WCOL Status Flag

If the user writes the SSPBUF when a Repeated Start sequence is in progress, then WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

Note:	Because queueing of events is not allowed, writing of the lower 5 bits of SSPCON2 is disabled until the Repeated Start condition is complete.

Figure 20-22: Repeat Start Condition Waveform

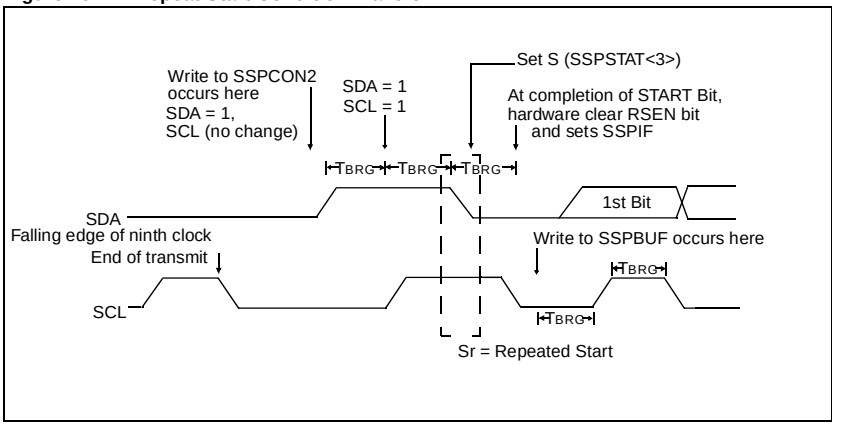


Figure 20-23: Repeated Start Condition Flowchart (part 1 of 2)

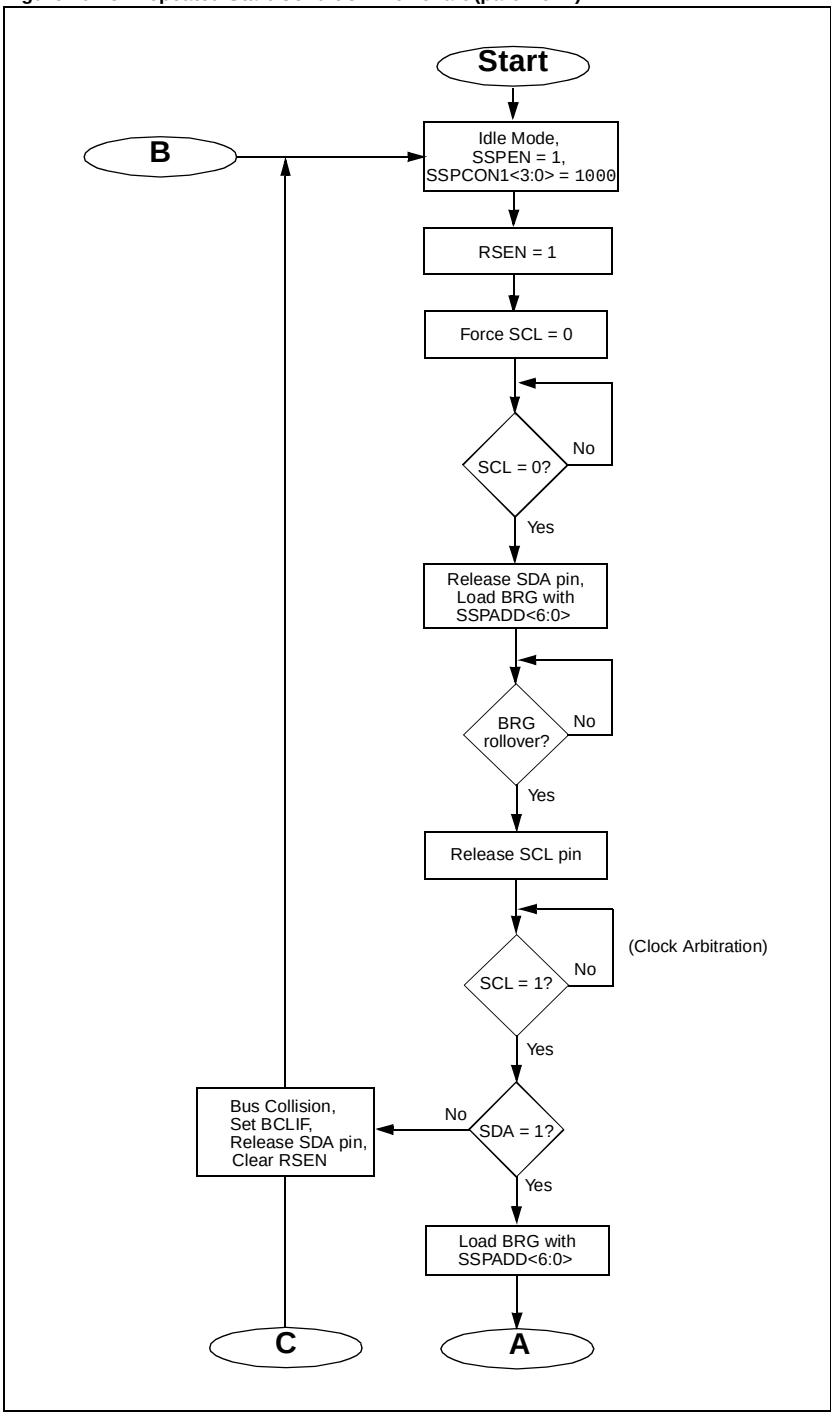
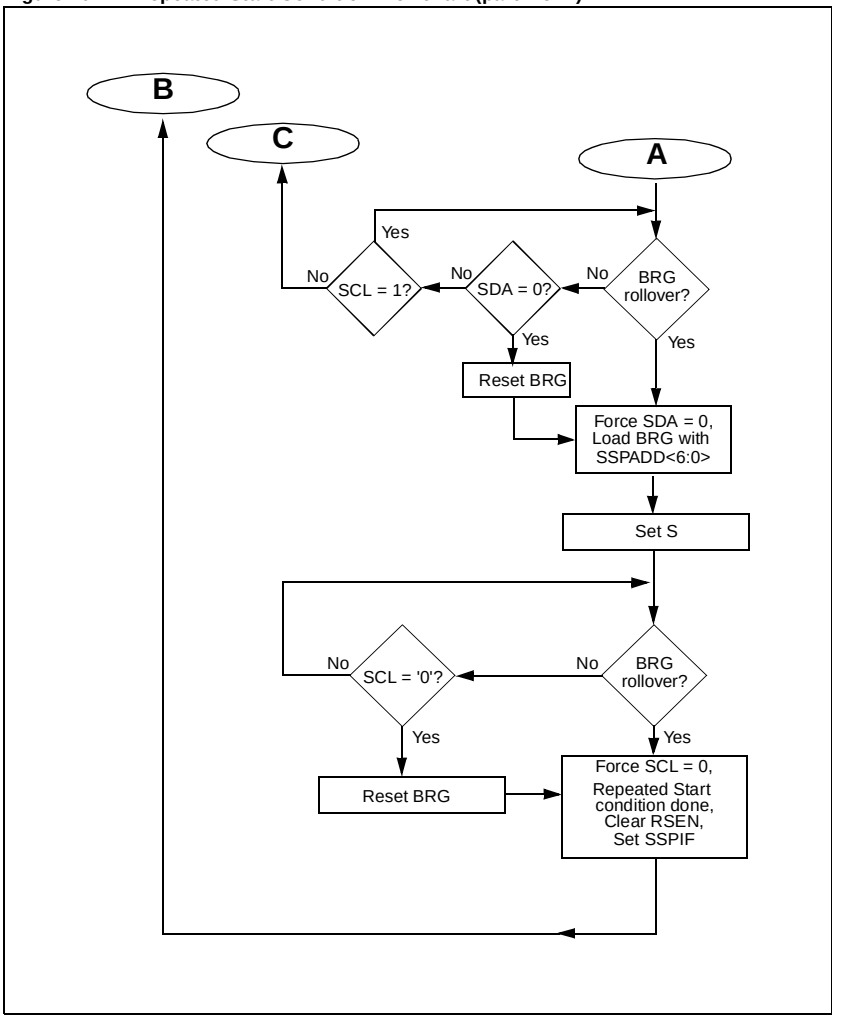


Figure 20-24: Repeated Start Condition Flowchart (part 2 of 2)



20.4.11 I²C Master Mode Transmission

Transmission of a data byte, a 7-bit address, or the other half of a 10-bit address is accomplished by simply writing a value to SSPBUF Register. This action will set the buffer full flag bit, BF, and allow the baud rate generator to begin counting and start the next transmission. Each bit of address/data will be shifted out onto the SDA pin after the falling edge of SCL is asserted (see data hold time specification parameter 106 in the **"Electrical Specifications"** section). SCL is held low for one baud rate generator roll over count (TBRG). Data should be valid before SCL is released high (see data setup time specification parameter 107 in the **"Electrical Specifications"** section). When the SCL pin is released high, it is held that way for TBRG, the data on the SDA pin must remain stable for that duration and some hold time after the next falling edge of SCL. After the eighth bit is shifted out (the falling edge of the eighth clock), the BF bit is cleared and the master releases the SDA pin. This allows the slave device being addressed to respond with an $\overline{ACK}$ bit during the ninth bit time, if an address match occurs or if data was received properly. The status of $\overline{ACK}$ is written into the ACKDT bit on the falling edge of the ninth clock. If the master receives an acknowledge, the acknowledge status bit, ACKSTAT, is cleared. If not, the bit is set. After the ninth clock, the SSPIF bit is set and the master clock (baud rate generator) is suspended until the next data byte is loaded into the SSPBUF, leaving the SCL pin low and the SDA pin unchanged (Figure 20-26).

After the write to the SSPBUF, each bit of address will be shifted out on the falling edge of SCL until all seven address bits and the R/$\overline{W}$ bit are completed. On the falling edge of the eighth clock, the master will de-assert the SDA pin allowing the slave to respond with an acknowledge. On the falling edge of the ninth clock, the master will sample the SDA pin to see if the address was recognized by a slave. The status of the ACK bit is loaded into the ACKSTAT status bit (SSPCON2 register). Following the falling edge of the ninth clock transmission of the address, the SSPIF is set, the BF flag is cleared, and the baud rate generator is turned off until another write to the SSPBUF takes place, holding SCL low and allowing SDA to float.

20.4.11.1 BF Status Flag

In transmit mode, the BF bit (SSPSTAT register) is set when the CPU writes to SSPBUF and is cleared when all 8 bits are shifted out.

20.4.11.2 WCOL Status Flag

If the user writes the SSPBUF when a transmit is already in progress (i.e. SSPSR is still shifting out a data byte), then WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

WCOL must be cleared in software.

20.4.11.3 ACKSTAT Status Flag

In transmit mode, the ACKSTAT bit (SSPCON2 register) is cleared when the slave has sent an acknowledge ($\overline{ACK}$ = 0), and is set when the slave does not acknowledge ($\overline{ACK}$ = 1). A slave sends an acknowledge when it has recognized its address (including a general call), or when the slave has properly received its data.

Figure 20-25: Master Transmit Flowchart

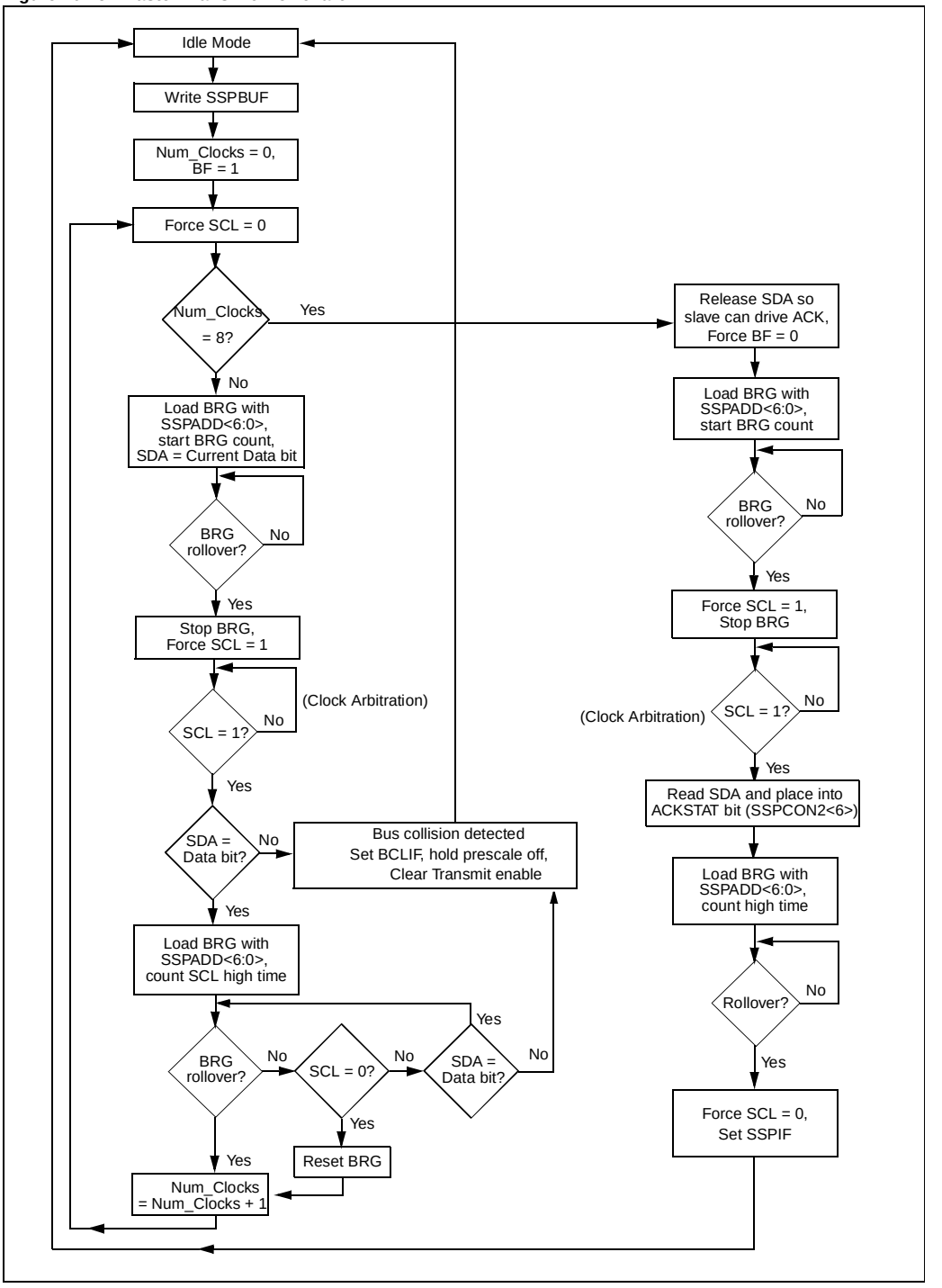
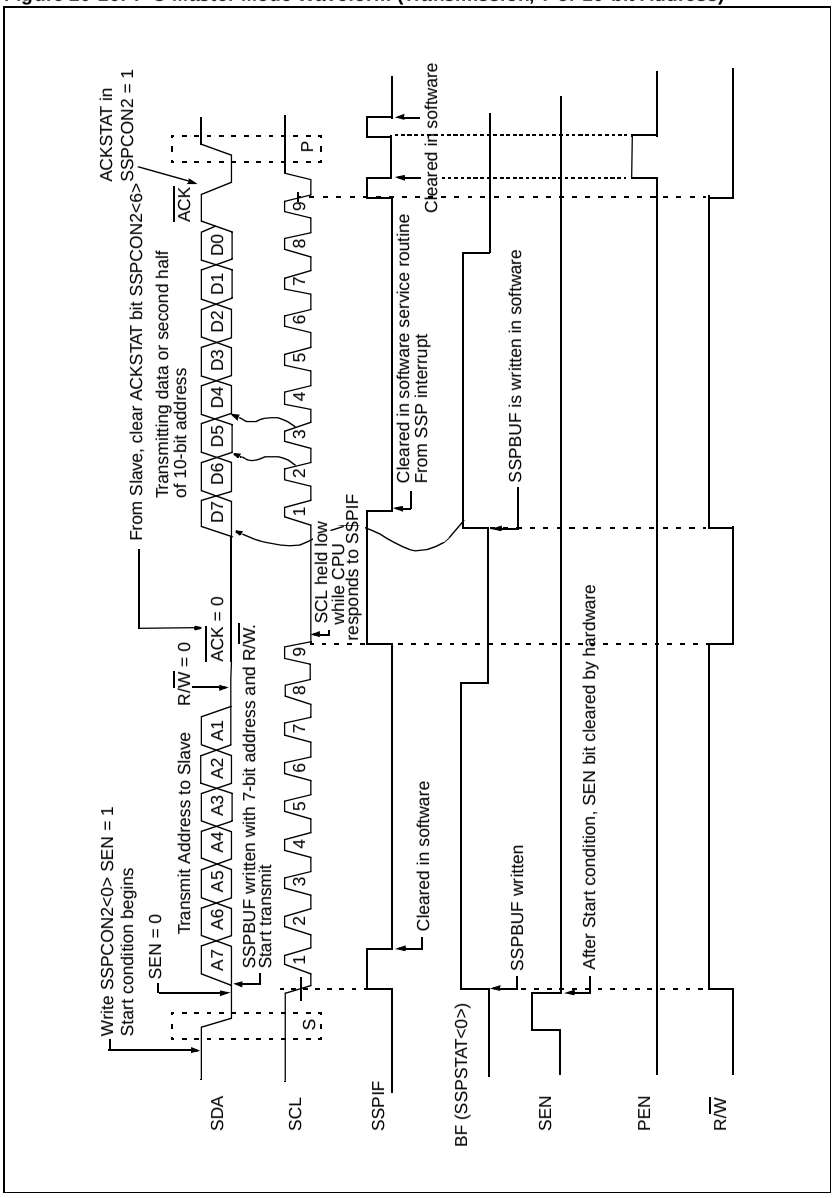


Figure 20-26: I^2C Master Mode Waveform (Transmission, 7 or 10-bit Address)



20.4.12 I²C Master Mode Reception

Master Mode reception is enabled by programming the receive enable bit, RCEN (SSPCON2 register).

Note:	The MSSP module must be in an idle state before the RCEN bit is set, or the RCEN bit will be disregarded.

The baud rate generator begins counting, and on each rollover, the state of the SCL pin changes (high to low/low to high) and data is shifted into the SSPSR. After the falling edge of the eighth clock, the receive enable flag is automatically cleared, the contents of the SSPSR are loaded into the SSPBUF, the BF flag bit is set, the SSPIF flag bit is set, and the baud rate generator is suspended from counting, holding SCL low. The MSSP is now in idle state, awaiting the next command. When the buffer is read by the CPU, the BF flag bit is automatically cleared. The user can then send an acknowledge bit at the end of reception by setting the acknowledge sequence enable bit, ACKEN (SSPCON2 register).

20.4.12.1 BF Status Flag

In receive mode, the BF bit is set when an address or data byte is loaded into SSPBUF from SSPSR. It is cleared when the SSPBUF Register is read.

20.4.12.2 SSPOV Status Flag

In receive mode, the SSPOV bit is set when 8 bits are received into the SSPSR, and the BF flag bit is already set from a previous reception.

20.4.12.3 WCOL Status Flag

If the user writes the SSPBUF when a receive is already in progress (i.e., SSPSR is still shifting in a data byte), then the WCOL bit is set and the contents of the buffer are unchanged (the write doesn't occur).

Figure 20-27: Master Receiver Flowchart

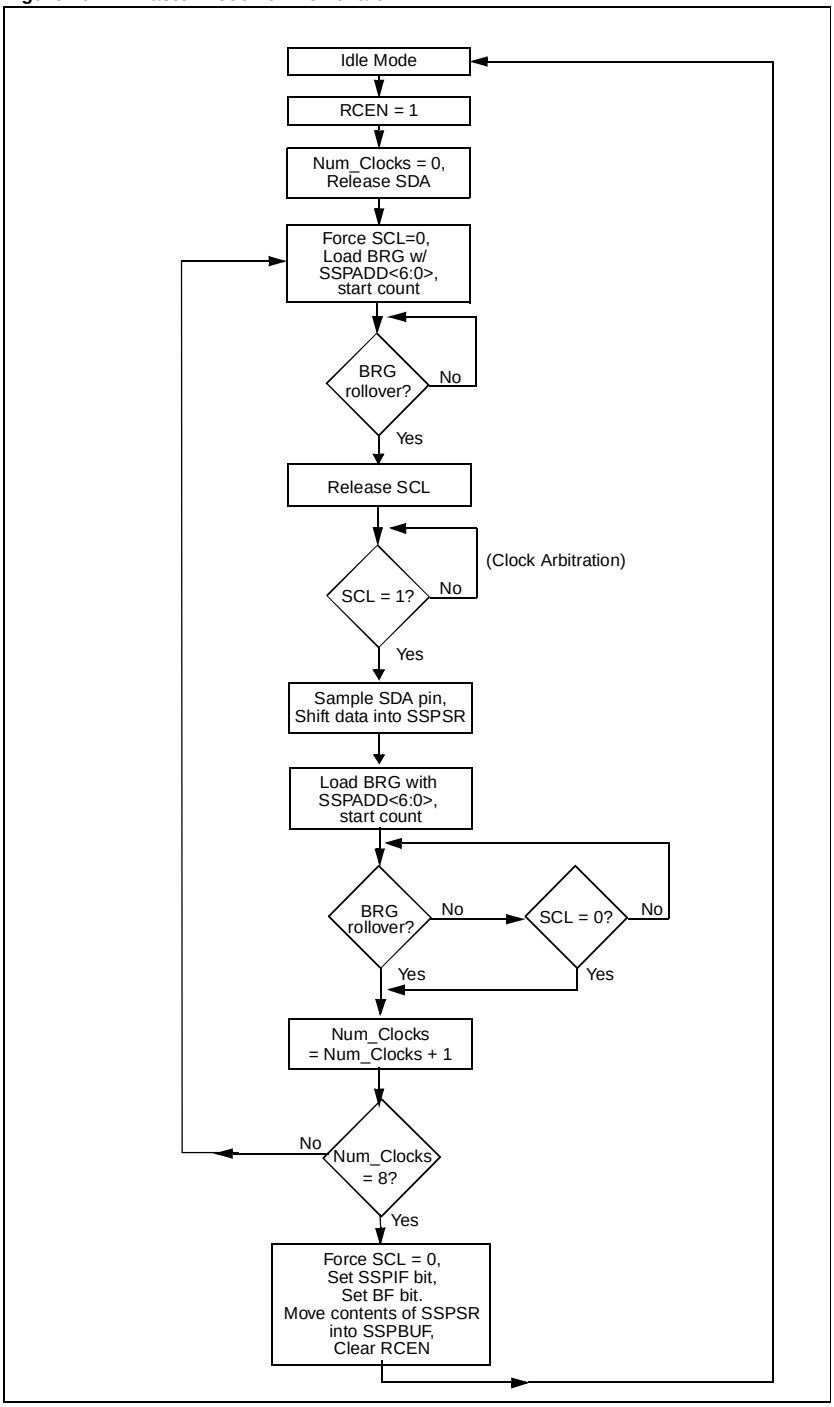
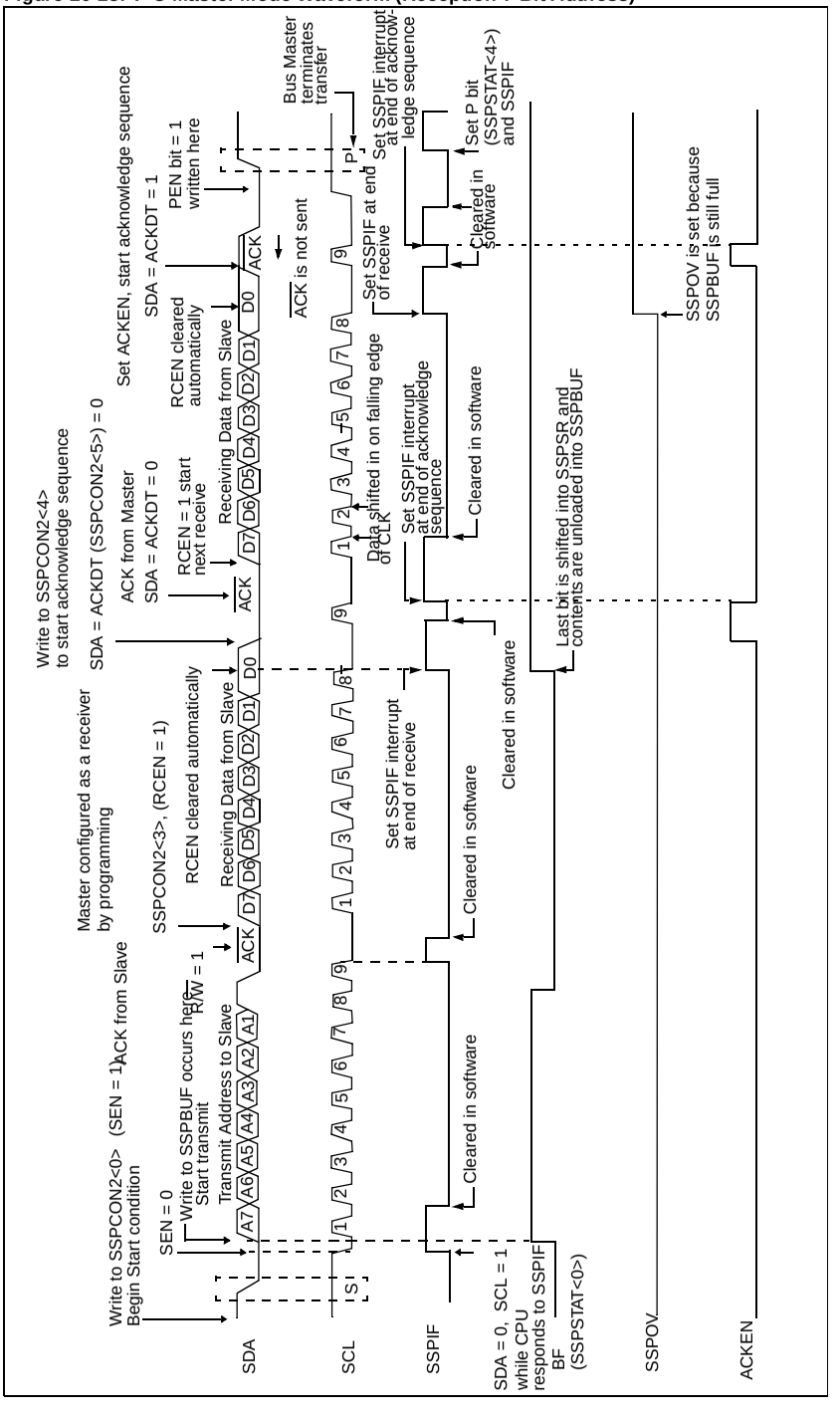


Figure 20-28: I²C Master Mode Waveform (Reception 7-Bit Address)



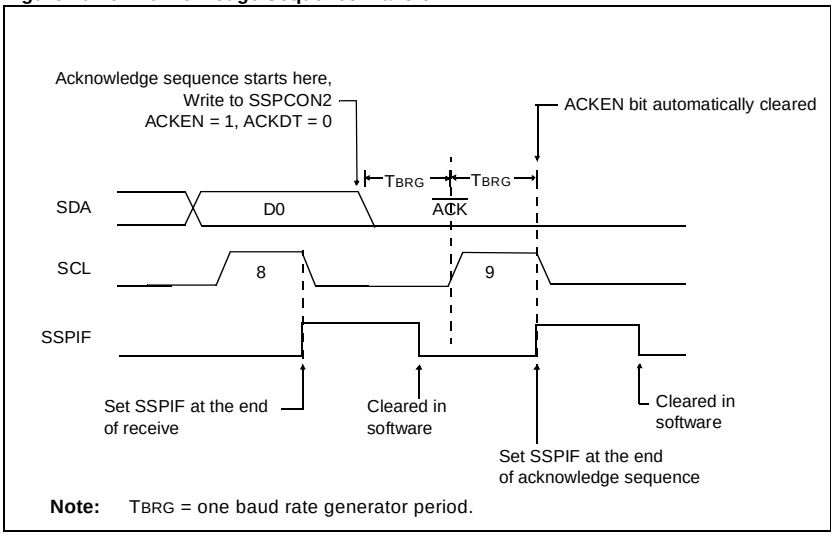
20.4.13 Acknowledge Sequence Timing

An acknowledge sequence is enabled by setting the acknowledge sequence enable bit, ACKEN (SSPCON2 register). When this bit is set, the SCL pin is pulled low and the contents of the acknowledge data bit is presented on the SDA pin. If the user wishes to generate an acknowledge, then the ACKDT bit should be cleared. If not, the user should set the ACKDT bit before starting an acknowledge sequence. The baud rate generator then counts for one rollover period (TBRG), and the SCL pin is de-asserted (pulled high). When the SCL pin is sampled high (clock arbitration), the baud rate generator counts for TBRG. The SCL pin is then pulled low. Following this, the ACKEN bit is automatically cleared, the baud rate generator is turned off, and the MSSP module then goes into idle mode (Figure 20-29).

20.4.13.1 WCOL Status Flag

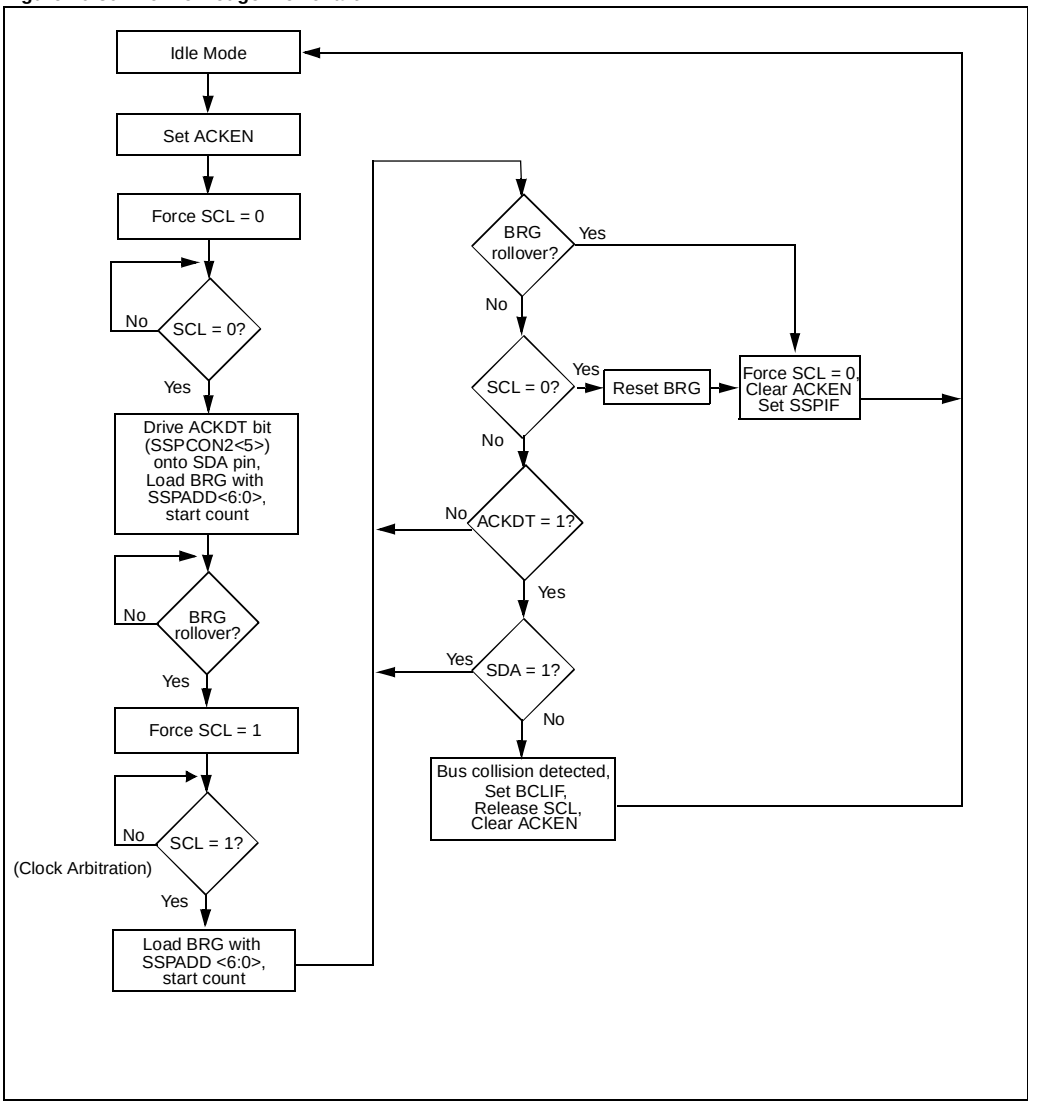
If the user writes the SSPBUF when an acknowledge sequence is in progress, then WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

Figure 20-29: Acknowledge Sequence Waveform



Note: TBRG = one baud rate generator period.

Figure 20-30: Acknowledge Flowchart



20.4.14 Stop Condition Timing

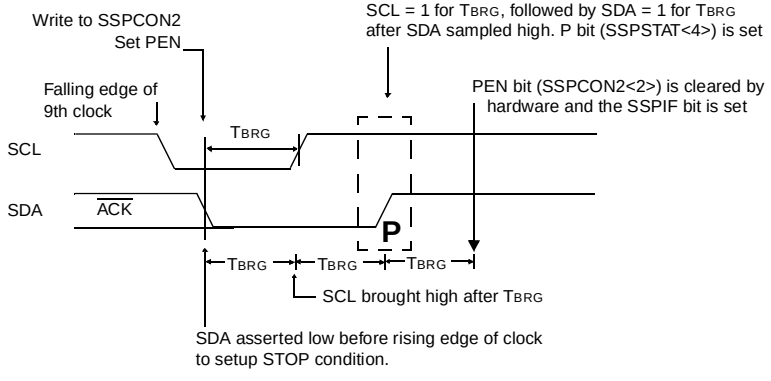
A Stop bit is asserted on the SDA pin at the end of a receive/transmit by setting the Stop sequence enable bit, PEN (SSPCON2 register). At the end of a receive/transmit, the SCL pin is held low after the falling edge of the ninth clock. When the PEN bit is set, the master will assert the SDA line low. When the SDA line is sampled low, the baud rate generator is reloaded and counts down to 0. When the baud rate generator times out, the SCL pin will be brought high, and one TBRG (baud rate generator rollover count) later, the SDA pin will be de-asserted. When the SDA pin is sampled high while the SCL pin is high, the P bit (SSPSTAT register) is set. A TBRG later, the PEN bit is cleared and the SSPIF bit is set (Figure 20-31).

Whenever the firmware decides to take control of the bus, it will first determine if the bus is busy by checking the S and P bits in the SSPSTAT Register. If the bus is busy, then the CPU can be interrupted (notified) when a Stop bit is detected (i.e., bus is free).

20.4.14.1 WCOL Status Flag

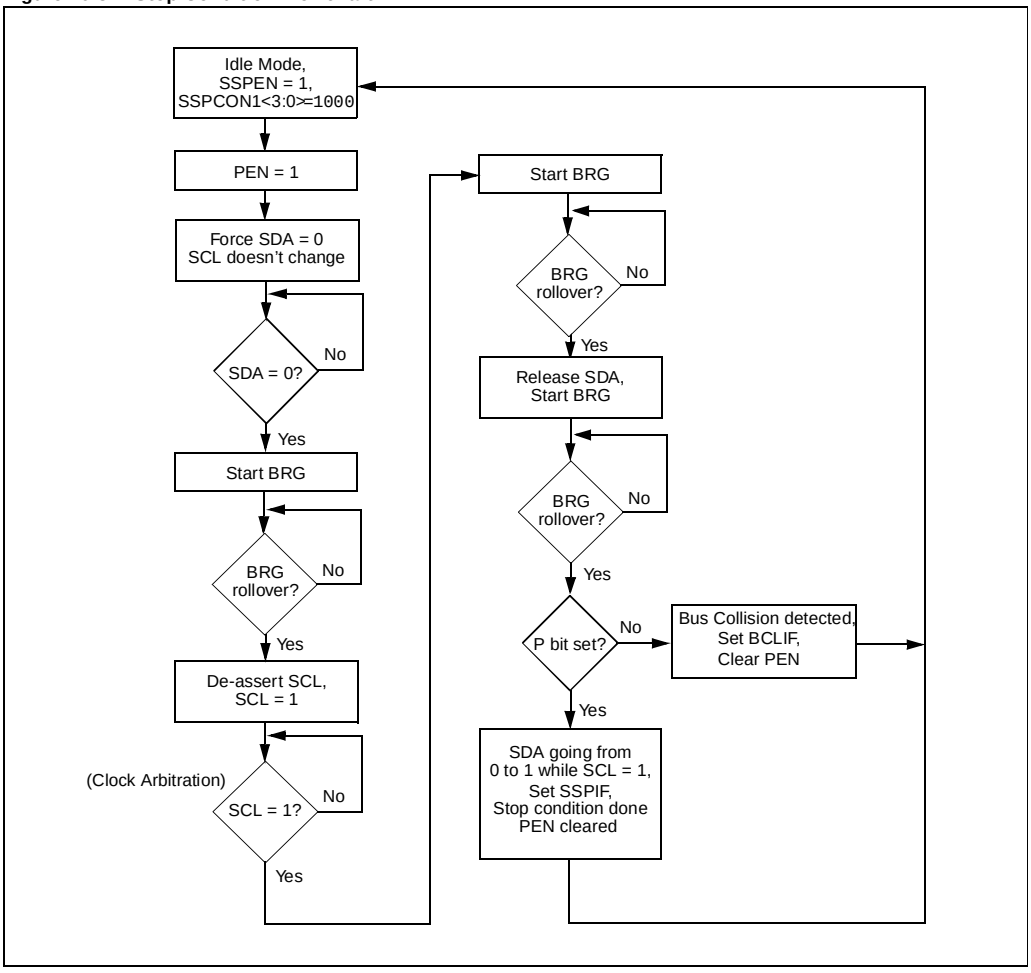
If the user writes the SSPBUF when a Stop sequence is in progress, then the WCOL bit is set and the contents of the buffer are unchanged (the write doesn't occur).

Figure 20-31: Stop Condition Receive or Transmit Mode



Note: TBRG = one baud rate generator period.

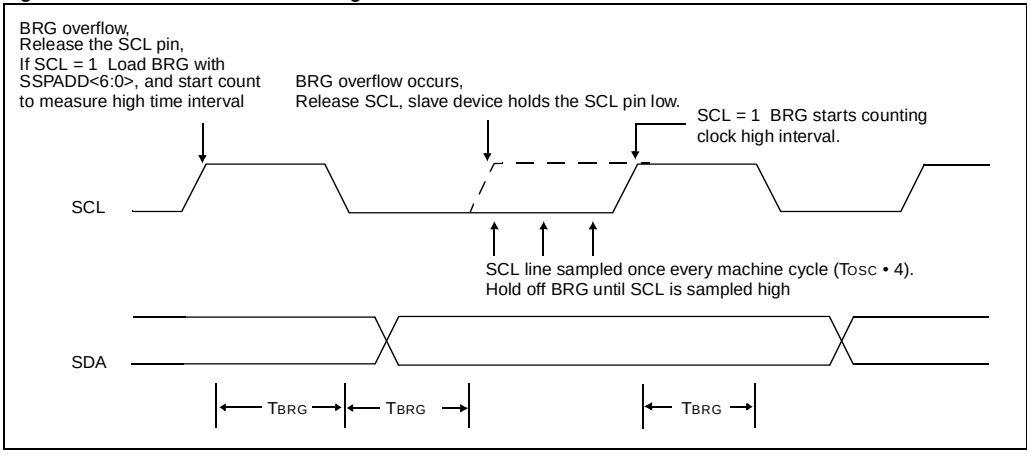
Figure 20-32: Stop Condition Flowchart



20.4.15 Clock Arbitration

Clock arbitration occurs when the master, during any receive, transmit, or Repeated Start/Stop condition de-asserts the SCL pin (SCL allowed to float high). When the SCL pin is allowed to float high, the baud rate generator (BRG) is suspended from counting until the SCL pin is actually sampled high. When the SCL pin is sampled high, the baud rate generator is reloaded with the contents of SSPADD<6:0> and begins counting. This ensures that the SCL high time will always be at least one BRG rollover count in the event that the clock is held low by an external device (Figure 20-33).

Figure 20-33: Clock Arbitration Timing in Master Transmit Mode



20.4.15.1 Sleep Operation

While in sleep mode, the I^2C module can receive addresses or data. When an address match or complete byte transfer occurs, the processor will wake-up from sleep (if the MSSP interrupt is enabled).

20.4.15.2 Effect of a Reset

A reset disables the MSSP module and terminates the current transfer.

20.4.16 Multi-Master Communication, Bus Collision, and Bus Arbitration

Multi-Master Mode support is achieved by bus arbitration. When the master outputs address/data bits onto the SDA pin, arbitration takes place when the master outputs a '1' on SDA by letting SDA float high and another master asserts a '0'. When the SCL pin floats high, data should be stable. If the expected data on SDA is a '1' and the data sampled on the SDA pin = '0', then a bus collision has taken place. The master will set the Bus Collision Interrupt Flag, BCLIF and reset the I^2C port to its idle state. (Figure 20-34).

If a transmit was in progress when the bus collision occurred, the transmission is halted, the BF flag is cleared, the SDA and SCL pins are de-asserted, and the SSPBUF can be written to. When the user services the bus collision interrupt service routine, and if the I^2C bus is free, the user can resume communication by asserting a Start condition.

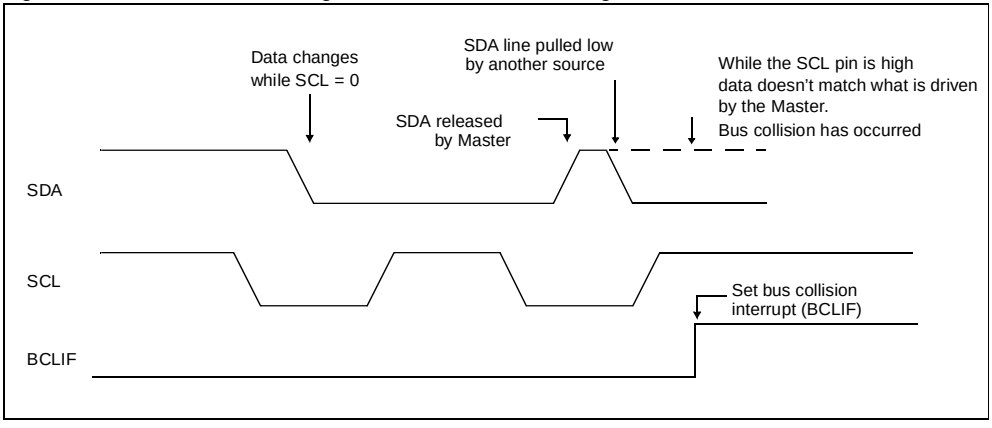
If a Start, Repeated Start, Stop, or Acknowledge condition was in progress when the bus collision occurred, the condition is aborted, the SDA and SCL lines are de-asserted, and the respective control bits in the SSPCON2 Register are cleared. When the user services the bus collision interrupt service routine, and if the I^2C bus is free, the user can resume communication by asserting a Start condition.

The Master will continue to monitor the SDA and SCL pins, and if a Stop condition occurs, the SSPIF bit will be set.

A write to the SSPBUF will start the transmission of data at the first data bit, regardless of where the transmitter left off when bus collision occurred.

In multi-Master Mode, the interrupt generation on the detection of Start and Stop conditions allows the determination of when the bus is free. Control of the I^2C bus can be taken when the P bit is set in the SSPSTAT Register, or the bus is idle and the S and P bits are cleared.

Figure 20-34: Bus Collision Timing for Transmit and Acknowledge



20.4.16.1 Bus Collision During a Start Condition

During a Start condition, a bus collision occurs if:

a) SDA or SCL pins are sampled low at the beginning of the Start condition (Figure 20-35).

b) SCL pins are sampled low before the SDA pin is asserted low (Figure 20-36).

During a Start condition both the SDA and the SCL pins are monitored.

If one of the following conditions exists:

- the SDA pin is already low
- or the SCL pin is already low,

Then, the following actions occur:

- the Start condition is aborted,
- the BCLIF bit is set,
- the MSSP module is reset to its idle state (Figure 20-35).

The Start condition begins with the SDA and SCL pins de-asserted. When the SDA pin is sampled high, the baud rate generator is loaded from SSPADD<6:0> and counts down to O. If the SCL pin is sampled low while SDA is high, a bus collision occurs, because it is assumed that another master is attempting to drive a data '1' during the Start condition.

If the SDA pin is sampled low during this count, the BRG is reset and the SDA line is asserted early (Figure 20-37). If however a '1' is sampled on the SDA pin, the SDA pin is asserted low at the end of the BRG count. The baud rate generator is then reloaded and counts down to O. During this time, if the SCL pins is sampled as '0', a bus collision does not occur. At the end of the BRG count the SCL pin is asserted low.

Note:	The reason that bus collision is not a factor during a Start condition is that no two bus masters can assert a Start condition at the exact same time. Therefore, one master will always assert SDA before the other. This condition does not cause a bus collision because the two masters must be allowed to arbitrate the first address following the Start condition, and if the address is the same, arbitration must be allowed to continue into the data portion, Repeated Start, or Stop conditions.

Figure 20-35: Bus Collision During Start Condition (SDA only)

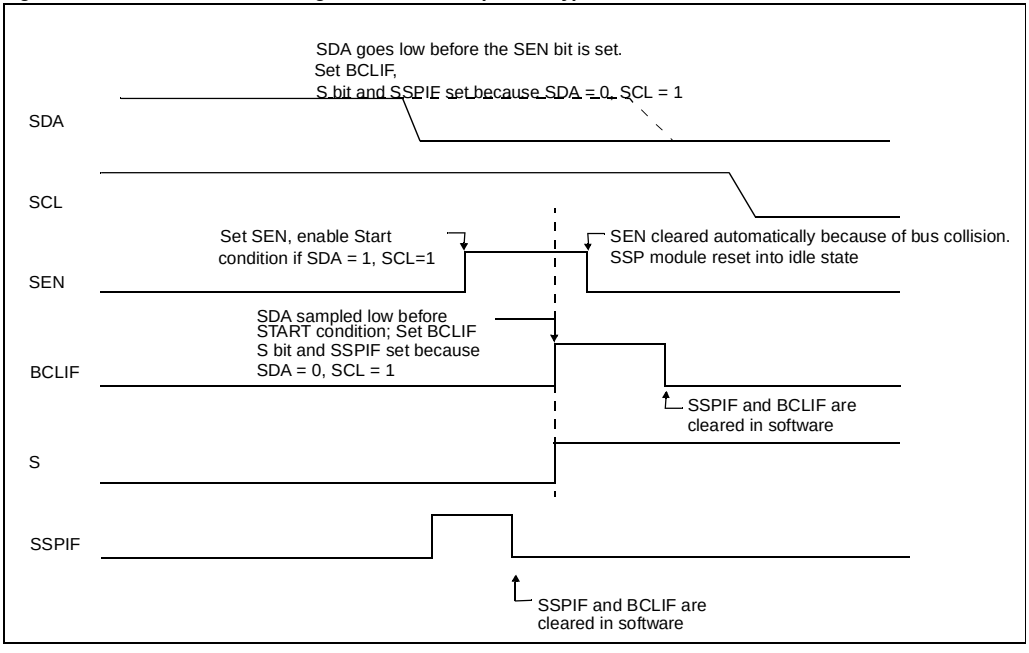


Figure 20-36: Bus Collision During Start Condition (SCL = 0)

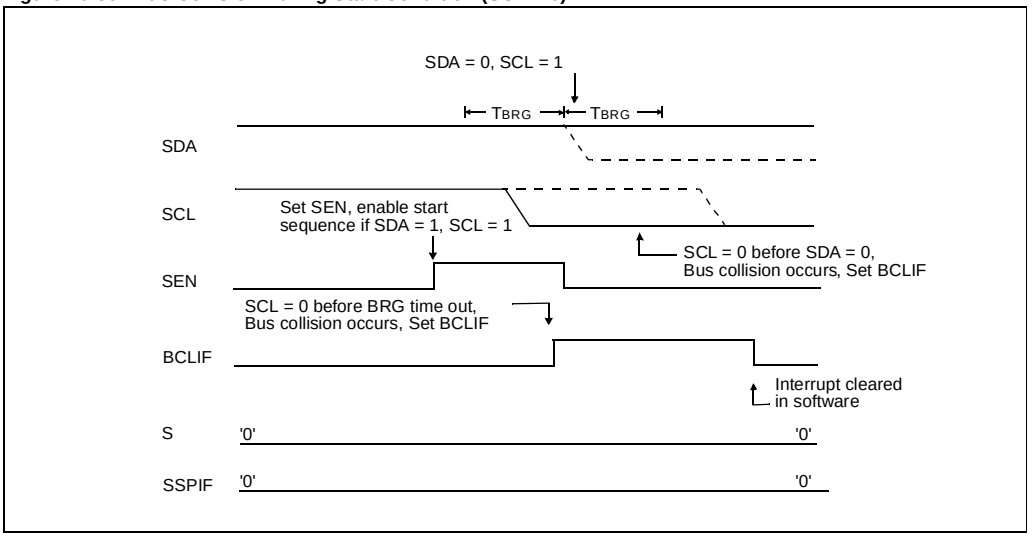
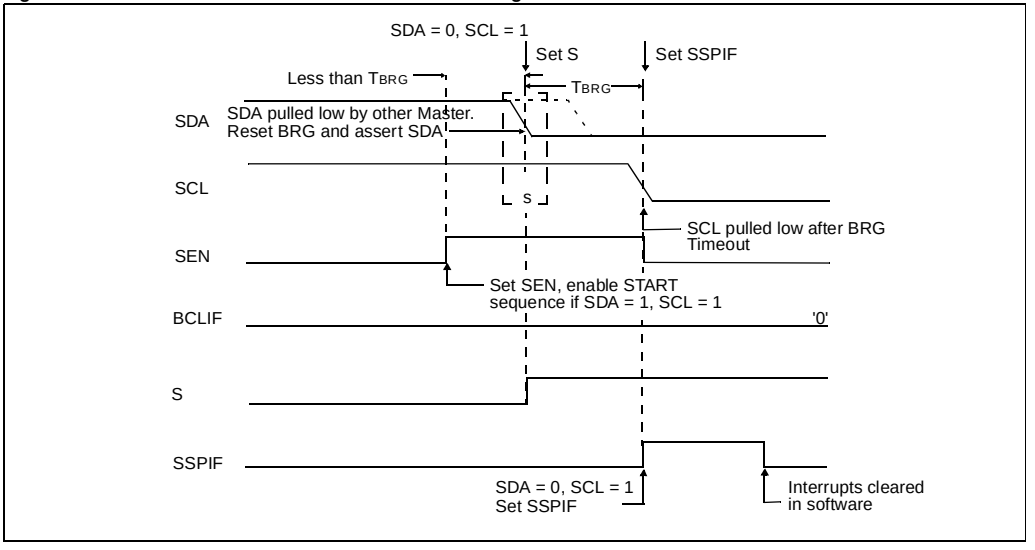


Figure 20-37: BRG Reset Due to SDA Arbitration During Start Condition



20.4.16.2 Bus Collision During a Repeated Start Condition

During a Repeated Start condition, a bus collision occurs if:

a) A low level is sampled on SDA when SCL goes from low level to high level.

b) SCL goes low before SDA is asserted low, indicating that another master is attempting to transmit a data '1'.

When the user de-asserts SDA and the pin is allowed to float high, the BRG is loaded with SSPADD<6:0> and counts down to 0. The SCL pin is then de-asserted, and when sampled high, the SDA pin is sampled.

If SDA is low, a bus collision has occurred (i.e., another master, is attempting to transmit a data '0'). If the SDA pin is sampled high, then the BRG is reloaded and begins counting. If the SDA pin goes from high to low before the BRG times out, no bus collision occurs because no two masters can assert SDA at exactly the same time, (Figure 20-38).

If the SCL pin goes from high to low before the BRG times out and the SDA pin has not already been asserted, then a bus collision occurs. In this case, another master is attempting to transmit a data '1' during the Repeated Start condition, (Figure 20-39).

If at the end of the BRG time-out, both the SCL and SDA pins are still high, the SDA pin is driven low and the BRG is reloaded and begins counting. At the end of the count, regardless of the status of the SCL pin, the SCL pin is driven low and the Repeated Start condition is complete.

Figure 20-38: Bus Collision During a Repeated Start Condition (Case 1)

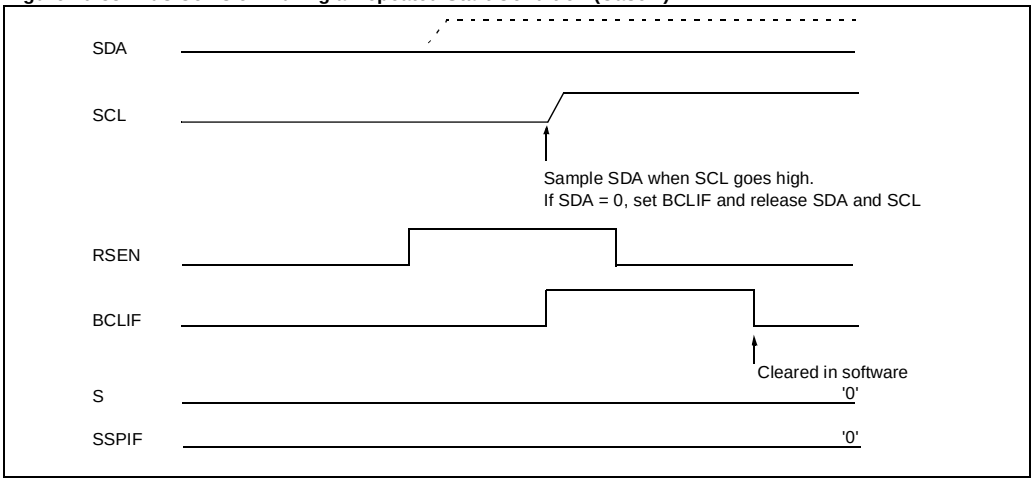
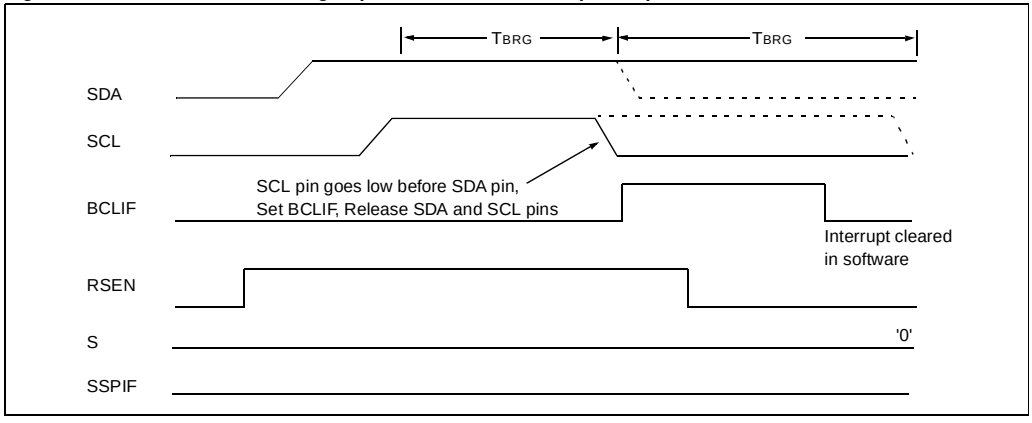


Figure 20-39: Bus Collision During Repeated Start Condition (Case 2)



20.4.16.3 Bus Collision During a Stop Condition

Bus collision occurs during a Stop condition if:

a) After the SDA pin has been de-asserted and allowed to float high, SDA is sampled low after the BRG has timed out.

b) After the SCL pin is de-asserted, SCL is sampled low before SDA goes high.

The Stop condition begins with SDA asserted low. When SDA is sampled low, the SCL pin is allow to float. When the pin is sampled high (clock arbitration), the baud rate generator is loaded with SSPADD<6:0> and counts down to 0. After the BRG times out, SDA is sampled. If SDA is sampled low, a bus collision has occurred. This is due to another master attempting to drive a data '0' (Figure 20-40). If the SCL pin is sampled low before SDA is allowed to float high, a bus collision occurs. This is another case of another master attempting to drive a data '0' (Figure 20-41).

Figure 20-40: Bus Collision During a Stop Condition (Case 1)

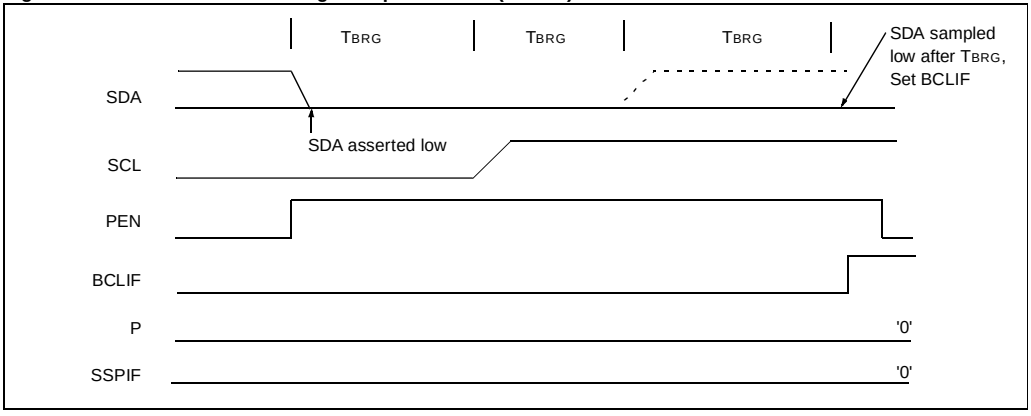
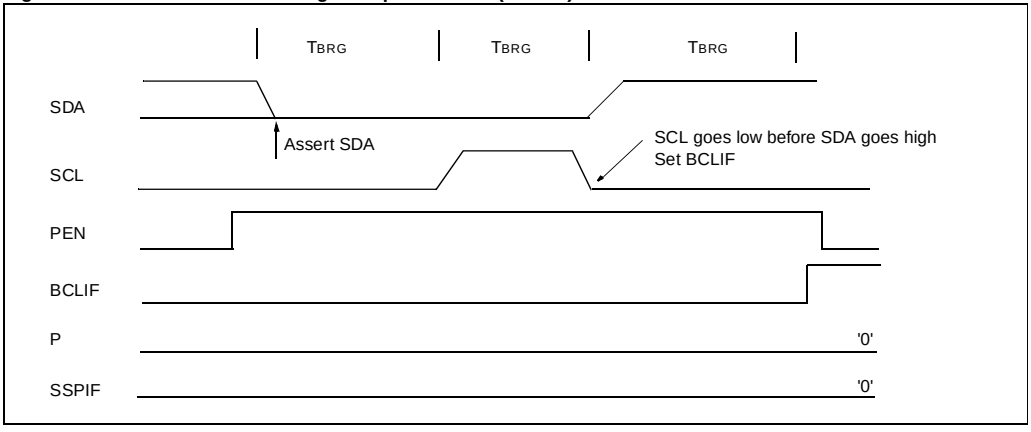


Figure 20-41: Bus Collision During a Stop Condition (Case 2)



20.4.17 Connection Considerations for I²C Bus

For standard-mode I²C bus devices, the values of resistors Rp and Rs in Figure 20-42 depend on the following parameters:

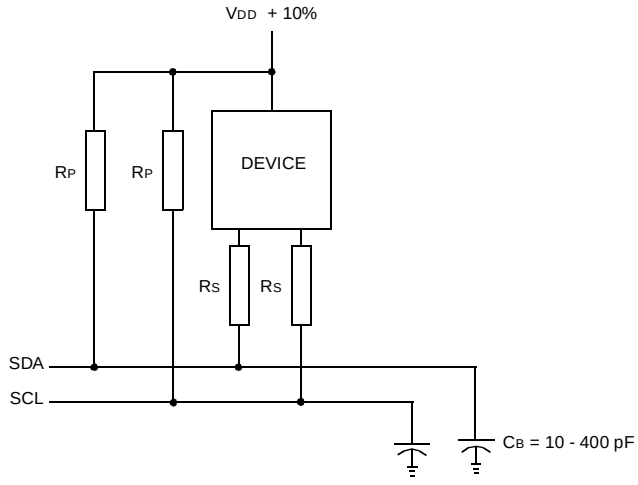
- Supply voltage
- Bus capacitance
- Number of connected devices (input current + leakage current)

The supply voltage limits the minimum value of resistor Rp due to the specified minimum sink current of 3 mA at VOLMAX = 0.4V for the specified output stages. For example, with a supply voltage of VDD = 5V±10% and VOLMAX = 0.4V at 3 mA, RPMIN = (5.5-0.4)/0.003 = 1.7 kΩ. VDD as a function of ***Rp*** is shown in Figure 20-42. The desired noise margin of 0.1VDD for the low level. This limits the maximum value of Rs. Series resistors are optional, and used to improve ESD susceptibility.

The bus capacitance is the total capacitance of wire, connections, and pins. This capacitance limits the maximum value of Rp due to the specified rise time (Figure 20-42).

The SMP bit is the slew rate control enabled bit. This bit is in the SSPSTAT Register, and controls the slew rate of the I/O pins when in I²C mode (master or slave).

Figure 20-42: Sample Device Configuration for I²C Bus



Note: I²C devices with input levels related to VDD must have one common supply line to which the pull up resistor is also connected.

20.4.18 Initialization

Example 20-2: SPI Master Mode Initialization

```
    CLRF   STATUS       ; Bank 0
    CLRF   SSPSTAT      ; SMP = 0, CKE = 0, and
                        ;   clear status bits
    BSF    SSPSTAT, CKE ; CKE = 1
    MOVLW  0x31         ; Set up SPI port, Master Mode, CLK/16,
    MOVWF  SSPCON       ;  Data xmit on falling edge
                        ;    (CKE=1 & CKP=1)
                        ;  Data sampled in middle
                        ;  (SMP=0 & Master Mode)
    BSF    PIE, SSPIE   ; Enable SSP interrupt
    BSF    INTCON, GIE  ; Enable, enabled interrupts
    MOVLW  DataByte     ; Data to be Transmitted
                        ;   Could move data from RAM location
    MOVWF  SSPBUF       ; Start Transmission
```

20.4.19 Master SSP Module / Basic SSP Module Compatibility

When changing from the SPI in the Mid-range Family Basic SSP module, the SSPSTAT Register contains two additional control bits. These bits are:

- SMP, SPI data input sample phase
- CKE, SPI Clock Edge Select

To be compatible with the SPI of the Master SSP module, these bits must be appropriately configured. If these bits are not at the states shown in Table 20-5, improper SPI communication may occur.

Table 20-5: New bit States for Compatibility

Basic SSP Module	Master SSP Module		
CKP	CKP	CKE	SMP
1	1	0	0
0	0	0	0

20.5 Design Tips

Question 1: ***Using SPI mode, I do not seem able to talk to an SPI device.***

Answer 1:

Ensure that you are using the correct SPI mode for that device. This SPI supports all 4 SPI modes so you should be able to get it to function. Check the clock polarity and the clock phase.

Question 2: ***Using I^2C mode, I write data to the SSPBUF Register, but the data did not transmit.***

Answer 2:

Ensure that you set the CKP bit to release the I^2C clock.

20.6 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is they may be written for the Baseline, the Midrange, or High-end families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the Master SSP modules are:

Title	Application Note #
Use of the SSP Module in the I ^{2}C Multi-Master Environment.	AN578
Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI Ports	AN613
Interfacing PIC16C64/74 to Microchip SPI Serial EEPROM	AN647
Interfacing a Microchip PIC16C92x to Microchip SPI Serial EEPROM	AN668

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

20.7 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Master SSP module description.

Section 21. Addressable USART

HIGHLIGHTS

This section of the manual contains the following major topics:

21.1 Introduction

The Addressable Universal Synchronous Asynchronous Receiver Transmitter (Addressable USART) module is one of the serial I/O modules available in the PIC18CXXX family (another is the MSSP module). The Addressable USART can be configured as a full duplex asynchronous system that can communicate with peripheral devices, such as CRT terminals and personal computers, or it can be configured as a half duplex synchronous system that can communicate with peripheral devices, such as A/D or D/A integrated circuits, Serial EEPROMs, etc.

The Addressable USART can be configured in the following modes:

- Asynchronous (full duplex)
- Synchronous - Master (half duplex)
- Synchronous - Slave (half duplex)

The SPEN bit (RCSTA register) and the TRIS bits, for the USART's pins, need to be set in order to configure the TX/CK and RX/DT pins for the Addressable USART.

21.2 Control Registers

Register 21-1: TXSTA: Transmit Status and Control Register

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit
When SYNC = 0 (Asynchronous mode)
Don't care

When SYNC = 1 (Synchronous mode)
1 = Master mode (Clock generated internally from BRG)
0 = Slave mode (Clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled

Note: The Receive Enable (SREN/CREN) bit overrides Transmit Enable (TXEN) bit in SYNC mode.

bit 4 **SYNC**: Addressable USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit
When SYNC = 0 (Asynchronous mode)
1 = High speed
0 = Low speed

When SYNC = 1 (Synchronous mode)
Unused in this mode

bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of transmit data.

This bit can be used as an address/data bit or a parity bit.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 21-2: RCSTA: Receive Status and Control Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (Configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit
When SYNC = 0 (Asynchronous mode)
Don't care

When SYNC = 1 (Synchronous mode) - master
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception of one byte is complete.

When SYNC = 1 (Synchronous mode) - slave
Unused in this mode

bit 4 **CREN**: Continuous Receive Enable bit
When SYNC = 0 (Asynchronous mode)
1 = Enables continuous receive
0 = Disables continuous receive

When SYNC = 1 (Synchronous mode)
1 = Enables continuous receive (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN**: Address Detect Enable bit
When SYNC = 0 (Asynchronous mode) with RX9 = 1 (9-bit receive enabled)
1 = Enables address detection, enable interrupt and loads of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received, and ninth bit can be used as parity bit

When SYNC = 0 (Asynchronous mode) with RX9 = 0 (9-bit receive disabled)

Don't care

When SYNC = 1 (Synchronous mode)

Don't care

bit 2 **FERR**: Framing Error bit
1 = Framing error (Can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit
1 = Overrun error (Can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of received data. Can be address/data bit or a parity bit.
1 = Ninth received bit was '1'
0 = Ninth received bit was '0'

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

21.3 USART Baud Rate Generator (BRG)

The BRG supports both the Asynchronous and Synchronous modes of the USART. It is a dedicated 8-bit baud rate generator. The SPBRG register controls the period of a free running 8-bit timer. In Asynchronous mode, the BRGH bit (TXSTA<2>) also controls the baud rate. In Synchronous mode, the BRGH bit is ignored. Table 21-1 shows the formula for computation of the baud rate for different USART modes that only apply in master mode (internal clock).

Given the desired baud rate and FOSC, the nearest integer value for the SPBRG register can be calculated using the formula in Table 21-1, where X equals the value in the SPBRG register (0 to 255). From this, the error in baud rate can be determined.

Table 21-1: Baud Rate Formula

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = FOSC/(64(X+1))	Baud Rate = FOSC/(16(X+1))
1	(Synchronous) Baud Rate = FOSC/(4(X+1))	NA

X = value in SPBRG (0 to 255)

Example 21-1 shows the calculation of the baud rate error for the following conditions:

FOSC = 16 MHz
Desired Baud Rate = 9600
BRGH = 0
SYNC = 0

Example 21-1:Calculating Baud Rate Error

Desired Baud Rate = FOSC / (64 (X + 1))

Solving for X:

X = ((FOSC / Desired Baud Rate) / 64) - 1
X = ((16000000 / 9600) / 64) - 1
X = [25.042] = 25

Calculated Baud Rate = 16000000 / (64 (25 + 1))
= 9615

$$\text{Error} = \frac{\text{(Calculated Baud Rate - Desired Baud Rate)}}{\text{Desired Baud Rate}}$$
= (9615 - 9600) / 9600
= 0.16%

It may be advantageous to use the high baud rate (BRGH = 1) even for slower baud clocks. This is because the $F_{OSC} / (16(X + 1))$ equation can reduce the baud rate error in some cases.

Writing a new value to the SPBRG register causes the BRG timer to be reset (or cleared). This ensures the BRG does not wait for a timer overflow before outputting the new baud rate.

21.3.1 SAMPLING

The data on the RX/DT pin is sampled three times by a majority detect circuit to determine if a high or a low level is present at the RX pin. See Section **21.4.4** for additional information.

Table 21-2: Registers Associated with Baud Rate Generator

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other resets
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 0000x	0000 000x
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented read as '0'. Shaded cells are not used by the BRG.

Table 21-3: Baud Rates for Synchronous Mode

BAUD RATE (Kbps)	FOSC = 40 MHz		SPBRG value (decimal)	33 MHz		SPBRG value (decimal)	25 MHz		SPBRG value (decimal)	20 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	NA	-	-	NA	-	-
9.6	NA	-	-	NA	-	-	NA	-	-	NA	-	-
19.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
76.8	76.92	+0.16	129	77.10	+0.39	106	77.16	+0.47	80	76.92	+0.16	64
96	96.15	+0.16	103	95.93	-0.07	85	96.15	+0.16	64	96.15	+0.16	51
300	303.03	+1.01	32	294.64	-1.79	27	297.62	-0.79	20	294.12	-1.96	16
500	500	0	19	485.30	-2.94	16	480.77	-3.85	12	500	0	9
HIGH	10000	-	0	8250	-	0	6250	-	0	5000	-	0
LOW	39.06	-	255	32.23	-	255	24.41	-	255	19.53	-	255

BAUD RATE (Kbps)	FOSC = 16 MHz		SPBRG value (decimal)	10 MHz		SPBRG value (decimal)	7.15909 MHz		SPBRG value (decimal)	5.0688 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	NA	-	-	NA	-	-
9.6	NA	-	-	NA	-	-	9.62	+0.23	185	9.60	0	131
19.2	19.23	+0.16	207	19.23	+0.16	129	19.24	+0.23	92	19.20	0	65
76.8	76.92	+0.16	51	75.76	-1.36	32	77.82	+1.32	22	74.54	-2.94	16
96	95.24	-0.79	41	96.15	+0.16	25	94.20	-1.88	18	97.48	+1.54	12
300	307.70	+2.56	12	312.50	+4.17	7	298.35	-0.57	5	316.80	+5.60	3
500	500	0	7	500	0	4	447.44	-10.51	3	422.40	-15.52	2
HIGH	4000	-	0	2500	-	0	1789.80	-	0	1267.20	-	0
LOW	15.63	-	255	9.77	-	255	6.99	-	255	4.95	-	255

BAUD RATE (Kbps)	FOSC = 4 MHz		SPBRG value (decimal)	3.579545 MHz		SPBRG value (decimal)	1 MHz		SPBRG value (decimal)	32.768 kHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	0.30	+1.14	26
1.2	NA	-	-	NA	-	-	1.20	+0.16	207	1.17	-2.48	6
2.4	NA	-	-	NA	-	-	2.40	+0.16	103	2.73	+13.78	2
9.6	9.62	+0.16	103	9.62	+0.23	92	9.62	+0.16	25	8.20	-14.67	0
19.2	19.23	+0.16	51	19.04	-0.83	46	19.23	+0.16	12	NA	-	-
76.8	76.92	+0.16	12	74.57	-2.90	11	83.33	+8.51	2	NA	-	-
96	1000	+4.17	9	99.43	+3.57	8	83.33	-13.19	2	NA	-	-
300	333.33	+11.11	2	298.30	-0.57	2	250	-16.67	0	NA	-	-
500	500	0	1	447.44	-10.51	1	NA	-	-	NA	-	-
HIGH	1000	-	0	894.89	-	0	250	-	0	8.20	-	0
LOW	3.91	-	255	3.50	-	255	0.98	-	255	0.03	-	255

Table 21-4: Baud Rates for Asynchronous Mode (BRGH = 0)

BAUD RATE (Kbps)	FOSC = 40 MHz		SPBRG value (decimal)	33 MHz		SPBRG value (decimal)	25 MHz		SPBRG value (decimal)	20 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	2.40	-0.07	214	2.40	-0.15	162	2.40	+0.16	129
9.6	9.62	+0.16	64	9.55	-0.54	53	9.53	-0.76	40	9.47	-1.36	32
19.2	18.94	-1.36	32	19.10	-0.54	26	19.53	+1.73	19	19.53	+1.73	15
76.8	78.13	+1.73	7	73.66	-4.09	6	78.13	+1.73	4	78.13	+1.73	3
96	89.29	-6.99	6	103.13	+7.42	4	97.66	+1.73	3	104.17	+8.51	2
300	312.50	+4.17	1	257.81	-14.06	1	NA	-	-	312.50	+4.17	0
500	625	+25.00	0	NA	-	-	NA	-	-	NA	-	-
HIGH	625	-	0	515.63	-	0	390.63	-	0	312.50	-	0
LOW	2.44	-	255	2.01	-	255	1.53	-	255	1.22	-	255

BAUD RATE (Kbps)	FOSC = 16 MHz		SPBRG value (decimal)	10 MHz		SPBRG value (decimal)	7.15909 MHz		SPBRG value (decimal)	5.0688 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	1.20	+0.16	207	1.20	+0.16	129	1.20	+0.23	92	1.20	0	65
2.4	2.40	+0.16	103	2.40	+0.16	64	2.38	-0.83	46	2.40	0	32
9.6	9.62	+0.16	25	9.77	+1.73	15	9.32	-2.90	11	9.90	+3.13	7
19.2	19.23	+0.16	12	19.53	+1.73	7	18.64	-2.90	5	19.80	+3.13	3
76.8	83.33	+8.51	2	78.13	+1.73	1	111.86	+45.65	0	79.20	+3.13	0
96	83.33	-13.19	2	78.13	-18.62	1	NA	-	-	NA	-	-
300	250	-16.67	0	156.25	-47.92	0	NA	-	-	NA	-	-
500	NA	-	-	NA	-	-	NA	-	-	NA	-	-
HIGH	250	-	0	156.25	-	0	111.86	-	0	79.20	-	0
LOW	0.98	-	255	0.61	-	255	0.44	-	255	0.31	-	255

BAUD RATE (Kbps)	FOSC = 4 MHz		SPBRG value (decimal)	3.579545 MHz		SPBRG value (decimal)	1 MHz		SPBRG value (decimal)	32.768 kHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	0.30	-0.16	207	0.30	+0.23	185	0.30	+0.16	51	0.26	-14.67	1
1.2	1.20	+1.67	51	1.19	-0.83	46	1.20	+0.16	12	NA	-	-
2.4	2.40	+1.67	25	2.43	+1.32	22	2.23	-6.99	6	NA	-	-
9.6	8.93	-6.99	6	9.32	-2.90	5	7.81	-18.62	1	NA	-	-
19.2	20.83	+8.51	2	18.64	-2.90	2	15.63	-18.62	0	NA	-	-
76.8	62.50	-18.62	0	55.93	-27.17	0	NA	-	-	NA	-	-
96	NA	-	-	NA	-	-	NA	-	-	NA	-	-
300	NA	-	-	NA	-	-	NA	-	-	NA	-	-
500	NA	-	-	NA	-	-	NA	-	-	NA	-	-
HIGH	62.50	-	0	55.93	-	0	15.63	-	0	0.51	-	0
LOW	0.24	-	255	0.22	-	255	0.06	-	255	0.002	-	255

Table 21-5: Baud Rates for Asynchronous Mode (BRGH = 1)

BAUD RATE (Kbps)	FOSC = 40 MHz		SPBRG value (decimal)	33 MHz		SPBRG value (decimal)	25 MHz		SPBRG value (decimal)	20 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	NA	-	-	NA	-	-
9.6	NA	-	-	9.60	-0.07	214	9.59	-0.15	162	9.62	+0.16	129
19.2	19.23	+0.16	129	19.28	+0.39	106	19.30	+0.47	80	19.23	+0.16	64
76.8	75.76	-1.36	32	76.39	-0.54	26	78.13	+1.73	19	78.13	+1.73	15
96	96.15	+0.16	25	98.21	+2.31	20	97.66	+1.73	15	96.15	+0.16	12
300	312.50	+4.17	7	294.64	-1.79	6	312.50	+4.17	4	312.50	+4.17	3
500	500	0	4	515.63	+3.13	3	520.83	+4.17	2	416.67	-16.67	2
HIGH	2500	-	0	2062.50	-	0	1562.50	-	0	1250	-	0
LOW	9.77	-	255	8,06	-	255	6.10	-	255	4.88	-	255

BAUD RATE (Kbps)	FOSC = 16 MHz		SPBRG value (decimal)	10 MHz		SPBRG value (decimal)	7.15909 MHz		SPBRG value (decimal)	5.0688 MHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	NA	-	-	NA	-	-
1.2	NA	-	-	NA	-	-	NA	-	-	NA	-	-
2.4	NA	-	-	NA	-	-	2.41	+0.23	185	2.40	0	131
9.6	9.62	+0.16	103	9.62	+0.16	64	9.52	-0.83	46	9.60	0	32
19.2	19.23	+0.16	51	18.94	-1.36	32	19.45	+1.32	22	18.64	-2.94	16
76.8	76.92	+0.16	12	78.13	+1.73	7	74.57	-2.90	5	79.20	+3.13	3
96	100	+4.17	9	89.29	-6.99	6	89.49	-6.78	4	105.60	+10.00	2
300	333.33	+11.11	2	312.50	+4.17	1	447.44	+49.15	0	316.80	+5.60	0
500	500	0	1	625	+25.00	0	447.44	-10.51	0	NA	-	-
HIGH	1000	-	0	625	-	0	447.44	-	0	316.80	-	0
LOW	3.91	-	255	2.44	-	255	1.75	-	255	1.24	-	255

BAUD RATE (Kbps)	FOSC = 4 MHz		SPBRG value (decimal)	3.579545 MHz		SPBRG value (decimal)	1 MHz		SPBRG value (decimal)	32.768 kHz		SPBRG value (decimal)
	KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR		KBAUD	% ERROR	
0.3	NA	-	-	NA	-	-	0.30	+0.16	207	0.29	-2.48	6
1.2	1.20	+0.16	207	1.20	+0.23	185	1.20	+0.16	51	1.02	-14.67	1
2.4	2.40	+0.16	103	2.41	+0.23	92	2.40	+0.16	25	2.05	-14.67	0
9.6	9.62	+0.16	25	9.73	+1.32	22	8.93	-6.99	6	NA	-	-
19.2	19.23	+0.16	12	18.64	-2.90	11	20.83	+8.51	2	NA	-	-
76.8	NA	-	-	74.57	-2.90	2	62.50	-18.62	0	NA	-	-
96	NA	-	-	111.86	+16.52	1	NA	-	-	NA	-	-
300	NA	-	-	223.72	-25.43	0	NA	-	-	NA	-	-
500	NA	-	-	NA	-	-	NA	-	-	NA	-	-
HIGH	250	-	0	55.93	-	0	62.50	-	0	2.05	-	0
LOW	0.98	-	255	0.22	-	255	0.24	-	255	0.008	-	255

21.4 USART Asynchronous Mode

In this mode, the USART uses standard nonreturn-to-zero (NRZ) format (one start bit, eight or nine data bits and one stop bit). The most common data format is 8 bits. An on-chip dedicated 8-bit baud rate generator can be used to derive standard baud rate frequencies from the oscillator. The USART transmits and receives the LSb first. The USART's transmitter and receiver are functionally independent, but use the same data format and baud rate. The baud rate generator produces a clock either x16 or x64 of the bit shift rate, depending on the BRGH bit (TXSTA register). Parity is not supported by the hardware, but can be implemented in software (stored as the ninth data bit). Asynchronous mode is stopped during SLEEP.

Asynchronous mode is selected by clearing the SYNC bit (TXSTA register).

The USART Asynchronous module consists of the following important elements:

- Baud Rate Generator
- Sampling Circuit
- Asynchronous Transmitter
- Asynchronous Receiver

21.4.1 USART Asynchronous Transmitter

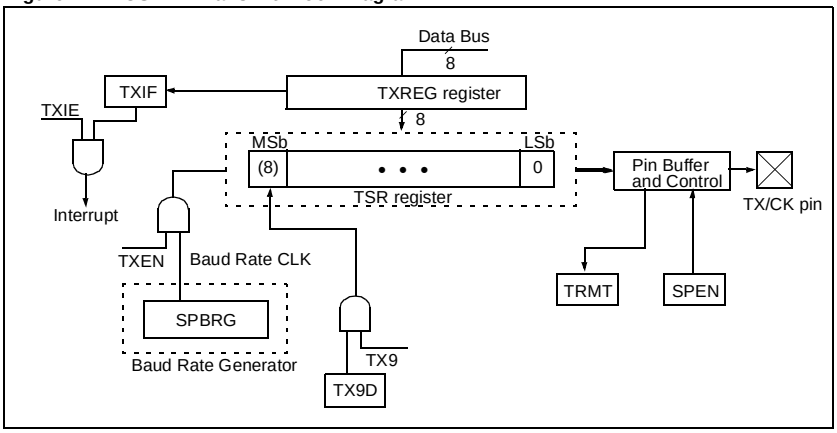
The USART transmitter block diagram is shown in Figure 21-1. The heart of the transmitter is the Transmit Shift Register (TSR). The shift register obtains its data from the transmit buffer, TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the STOP bit has been transmitted from the previous load. As soon as the STOP bit is transmitted, the TSR is loaded with new data from the TXREG register (if available). Once the TXREG register transfers the data to the TSR register (occurs in one TCY), the TXREG register is empty and the TXIF flag bit is set. This interrupt can be enabled/disabled by setting/clearing the TXIE enable bit. The TXIF flag bit will be set, regardless of the state of the TXIE enable bit and cannot be cleared in software. It will reset only when new data is loaded into the TXREG register. While the TXIF flag bit indicated the status of the TXREG register, the TRMT bit (TXSTA register) shows the status of the TSR register. The TRMT status bit is a read only bit, which is set when the TSR register is empty. No interrupt logic is tied to this bit, so the user has to poll this bit in order to determine if the TSR register is empty.

Note 1:	The TSR register is not mapped in data memory, so it is not available to the user.
2:	When the TXEN bit is set, the TXIF flag bit will also be set since the transmit buffer is not yet full (can move transmit data to the TXREG register).

Transmission is enabled by setting the TXEN enable bit (TXSTA register). The actual transmission will not occur until the TXREG register has been loaded with data and the Baud Rate Generator (BRG) has produced a shift clock (Figure 21-1). The transmission can also be started by first loading the TXREG register and then setting the TXEN enable bit. Normally when transmission is first started, the TSR register is empty, so a transfer to the TXREG register will result in an immediate transfer to TSR, resulting in an empty TXREG. A back-to-back transfer is thus possible (Figure 21-3). Clearing the TXEN enable bit during a transmission will cause the transmission to be aborted and will reset the transmitter. As a result, the TX/CK pin will revert to hi-impedance.

In order to select 9-bit transmission the TX9 bit (TXSTA register) should be set and the ninth bit should be written to the TX9D bit (TXSTA register). The ninth bit must be written before writing the 8-bit data to the TXREG register. This is because a data write to the TXREG register can result in an immediate transfer of the data to the TSR register (if the TSR is empty). In such a case, an incorrect ninth data bit may be loaded in the TSR register.

Figure 21-1: USART Transmit Block Diagram



Steps to follow when setting up an Asynchronous Transmission:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is desired, set the BRGH bit. (Subsection **21.3 "USART Baud Rate Generator (BRG)"**).
2. Enable the asynchronous serial port by clearing the SYNC bit and setting the SPEN bit.
3. If interrupts are desired, then set the TXIE, GIE/GIEH and PEIE/GIEL bits. Specify the interrupt priority if required.
4. If 9-bit transmission is desired, then set the TX9 bit (can be used as address/data bit).
5. Enable the transmission by setting the TXEN bit, which will also set the TXIF bit.
6. If 9-bit transmission is selected, the ninth bit should be loaded in the TX9D bit.
7. Load data to the TXREG register (starts transmission).

Figure 21-2: Asynchronous Transmission (8- or 9-bit Data)

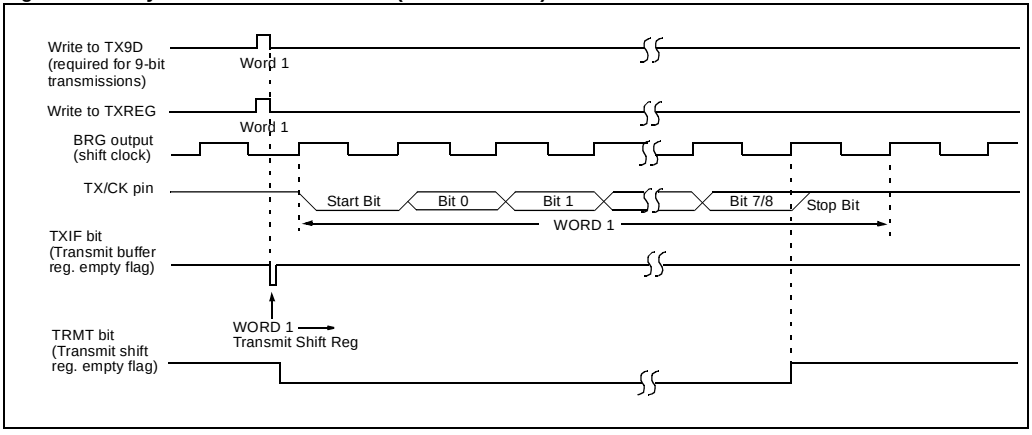
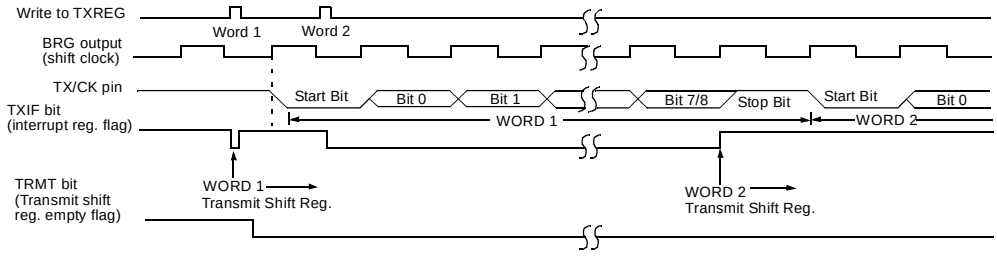


Figure 21-3: Asynchronous Transmission (Back to Back)



Note: This timing diagram shows two consecutive transmissions.

Table 21-6: Registers Associated with Asynchronous Transmission

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
INTCON	GIE/GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIRx	TXIF (1)								0	0
PIEx	TXIE (1)								0	0
IPRx	TXIP (1)								0	0
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
TXREG	TX7	TX6	TX5	TX4	TX3	TX2	TX1	TX0	0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'.
Shaded cells are not used for Asynchronous Transmission.

Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18C2X2 devices. Always maintain these bits clear.

2: The position of this bit is device dependent.

21.4.2 USART Asynchronous Receiver

The receiver block diagram is shown in Figure 21-4. The data is received on the RX/DT pin and drives the data recovery block. The data recovery block is actually a high speed shifter operating at x16 times the baud rate, whereas the main receive serial shifter operates at the bit rate or at FOSC. This mode would typically be used in RS-232 systems.

The USART module has a special provision for multi-processor communication. When the RX9 bit is set in the RCSTA register, 9-bits are received and the ninth bit is placed in the RX9D status bit of the RSTA register. The port can be programmed such that when the stop bit is received, the serial port interrupt will only be activated if the RX9D bit is set. This feature is enabled by setting the ADDEN bit in the RCSTA register and can be used in a multi-processor system in the following manner.

To transmit a block of data in a multi-processor system, the master processor must first send an address byte that identifies the target slave. An address byte is identified by the RX9D bit being a '1' (instead of a '0' for a data byte). If the ADDEN bit is set in the slave's RCSTA register, all data bytes will be ignored. However, if the ninth received bit is equal to a '1', indicating that the received byte is an address, the slave will be interrupted and the contents of the Receive Shift Register (RSR) will be transferred into the receive buffer. This allows the slave to be interrupted only by addresses, so that the slave can examine the received byte to see if it is addressed. The addressed slave will then clear its ADDEN bit and prepare to receive data bytes from the master.

When the ADDEN bit is set, all data bytes are ignored. Following the STOP bit, the data will not be loaded into the receive buffer and no interrupt will occur. If another byte is shifted into the RSR register, the previous data byte will be lost.

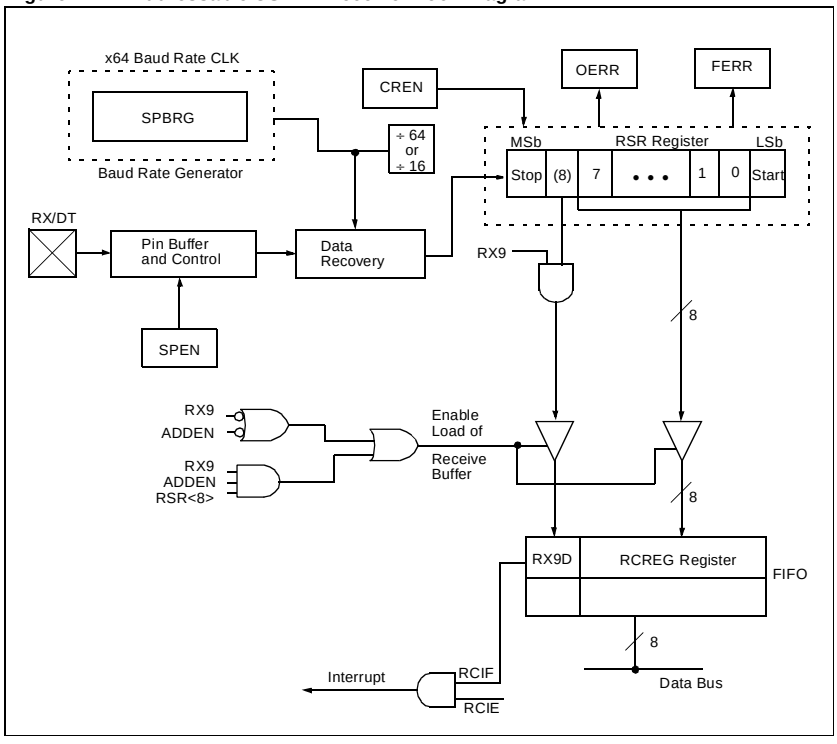
The ADDEN bit will only take effect when the receiver is configured in 9-bit mode.

Once Asynchronous mode is selected, reception is enabled by setting the CREN bit.

The heart of the receiver is the Receive (serial) Shift Register (RSR). After sampling the RX/TX pin for the STOP bit, the received data in the RSR is transferred to the RCREG register (if it is empty). If the transfer is complete, the RCIF flag bit is set. The actual interrupt can be enabled/disabled by setting/clearing the RCIE enable bit. The RCIF flag bit is a read only bit that is cleared by the hardware. It is cleared when the RCREG register has been read and is empty. The RCREG is a double-buffered register (i.e., it is a two-deep FIFO). It is possible for two bytes of data to be received and transferred to the RCREG FIFO and a third byte begin shifting to the RSR register. On the detection of the STOP bit of the third byte, if the RCREG register is still full, overrun error bit, OERR, will be set. The word in the RSR will be lost. The RCREG register can be read twice to retrieve the two bytes in the FIFO. The OERR bit has to be cleared in software. This is done by resetting the receive logic (the CREN bit is cleared and then set). If the OERR bit is set, transfers from the RSR register to the RCREG register are inhibited, so it is essential to clear the OERR bit if it is set. Framing error bit, FERR, is set if a stop bit is detected as a low level. The FERR bit and the 9th receive bit are buffered the same way as the receive data. Reading the RCREG will load the RX9D and FERR bits with new values. Therefore, it is essential for the user to read the RCSTA register before reading the next RCREG register, in order not to lose the old (previous) information in the FERR and RX9D bits.

Figure 21-4 shows a block diagram for the receive of the Addressable USART.

Figure 21-4: Addressable USART Receive Block Diagram

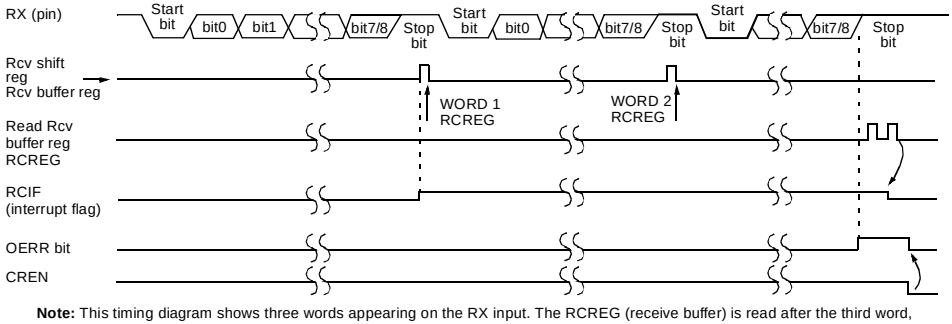


21.4.2.1 Asynchronous Receptions (no Address Detect)

Steps to follow when setting up an Asynchronous Reception:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is desired, set bit BRGH. (Subsection **21.3 "USART Baud Rate Generator (BRG)"**).
2. Enable the asynchronous serial port by clearing the SYNC bit and setting the SPEN bit.
3. If interrupts are desired, then set the RCIE bit and configure the RCIP, GIE/GIEH and PEIE/GIEL bits, appropriately.
4. If 9-bit reception is desired, then set the RX9 bit.
5. Enable the reception by setting the CREN bit.
6. The RCIF flag bit will be set when reception is complete. An interrupt will be generated depending on the configuration of the RCIE, RCIP, GIE/GIEH and PEIE/GIEL bits.
7. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
8. Read the 8-bit received data by reading the RCREG register.
9. If any error occurred, clear the error by clearing the CREN bit.

Figure 21-5: Asynchronous Reception (8- or 9-bit Data)



Note: This timing diagram shows three words appearing on the RX input. The RCREG (receive buffer) is read after the third word, causing the OERR (overrun) bit to be set.

21.4.3 Setting up 9-bit mode with Address Detect

Address detect mode allows an Addressable USART node to ignore all data on the bus until a new address byte is present. This reduces the interrupt overhead since not every byte will generate an interrupt (only bytes that are directed to that node).

21.4.3.1 Transmit

Steps to follow when setting up an Asynchronous Transmission:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is desired, set the BRGH bit. (Subsection **21.3 "USART Baud Rate Generator (BRG)"**).
2. Enable the asynchronous serial port by clearing the SYNC bit and setting the SPEN bit.
3. If interrupts are desired, then set the TXIE, TXIP, GIE/GIEH and PEIE/GIEL bits.
4. If 9-bit transmission is desired, then set the TX9 bit (can be used as address/data bit).
5. Enable the transmission by setting the TXEN bit, which will also set the TXIF bit.
6. If 9-bit transmission is selected, set the TX9D bit for address, clear the TX9D bit for data, set the TX9D bit for address and clear the TX9D bit for data.
7. Load data to the TXREG register (starts transmission).

21.4.3.2 Receive

Steps to follow when setting up an Asynchronous Reception with Address Detect enabled:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is desired, set bit BRGH.
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, then set the RCIE bit and configure the RCIP, GIE/GIEH and PEIE/GIEL bits, appropriately.
4. Set bit RX9 to enable 9-bit reception.
5. Set ADDEN to enable address detect.
6. Enable the reception by setting enable bit CREN.
7. The RCIF flag bit will be set when reception is complete. An interrupt will be generated depending on the configuration of the RCIE, RCIP, GIE/GIEH and PEIE/GIEL bits.
8. Read the RCSTA register to get the ninth bit and determine if any error occurred during reception.
9. Read the 8-bit received data by reading the RCREG register, to determine if the device is being addressed.
10. If any error occurred, clear the error by clearing enable bit CREN.
11. If the device has been addressed, clear the ADDEN bit to allow data bytes and address bytes to be read into the receive buffer, and interrupt the CPU.

Figure 21-6: USART Receive Block Diagram

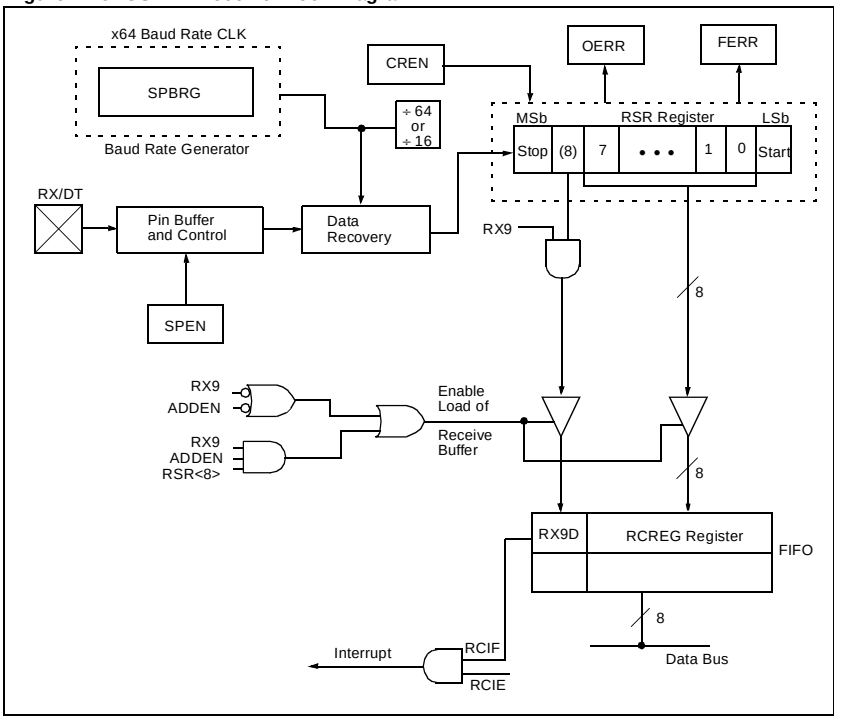
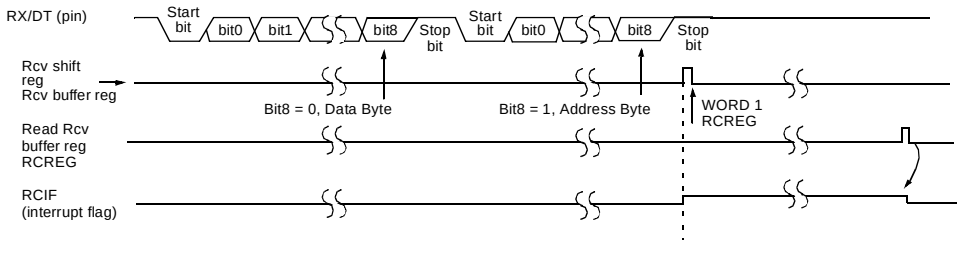
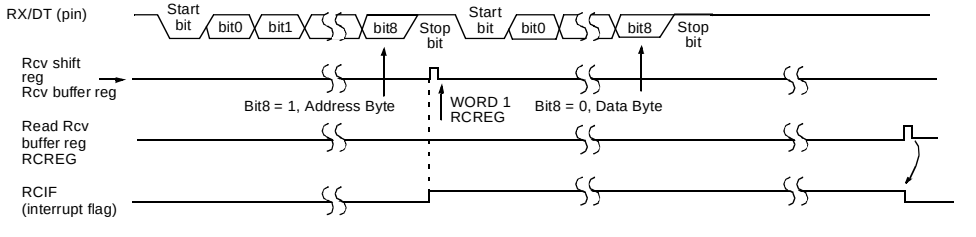


Figure 21-7: Asynchronous Reception with Address Detect



Note: This timing diagram shows a data byte followed by an address byte. The data byte is not read into the RCREG (receive buffer) because ADDEN = 0.

Figure 21-8: Asynchronous Reception with Address Byte First



Note: This timing diagram shows an address byte followed by an data byte. The data byte is not read into the RCREG (receive buffer) because ADDEN was not updated and still = 0.

21.4.4 Sampling

The data on the RX/DT pin is sampled three times by a majority detect circuit to determine if a high or a low level is present at the RX pin. Figure 21-9 shows the waveform for the sampling circuit. The sampling operates the same regardless of the state of the BRGH bit, only the source of the x16 clock is different.

Figure 21-9: RX Pin Sampling Scheme, BRGH = 0 or BRGH = 1

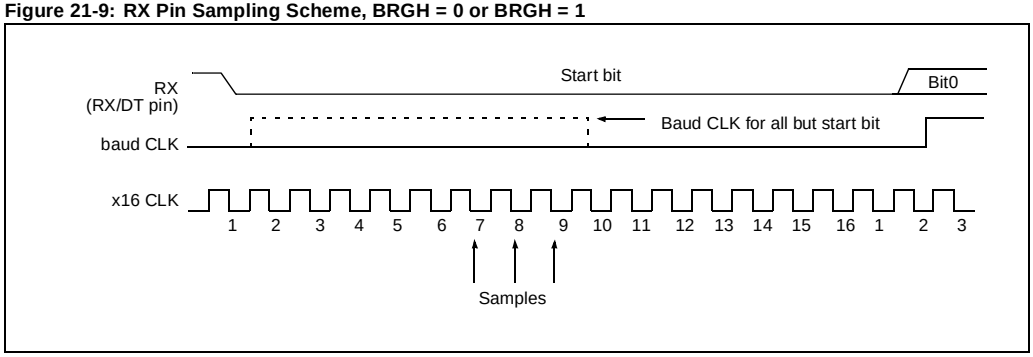


Table 21-7: Registers Associated with Asynchronous Reception

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIRx	RCIF (1)								0	0
PIEx	RCIE (1)								0	0
IPRx	RCIP (1)								0	0
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
RCREG	RX7	RX6	RX5	RX4	RX3	RX2	RX1	RX0	0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'.
Shaded cells are not used for Asynchronous Reception.

Note 1: The position of this bit is device dependent.

21.5 USART Synchronous Master Mode

In Synchronous Master mode, the data is transmitted in a half-duplex manner, (i.e., transmission and reception do not occur at the same time). When transmitting data, the reception is inhibited and vice versa. Synchronous mode is entered by setting the SYNC bit. In addition, the SPEN enable bit is set in order to configure the TX/CK and RX/DT I/O pins to CK (clock) and DT (data) lines respectively. The Master mode indicates that the processor transmits the master clock on the CK line. The Master mode is entered by setting the CSRC bit.

21.5.1 USART Synchronous Master Transmission

The USART transmitter block diagram is shown in Figure 21-1. The heart of the transmitter is the Transmit Shift Register (TSR). The shift register obtains its data from the read/write transmit buffer register TXREG. The TXREG register is loaded with data in software. The TSR register is not loaded until the last bit has been transmitted from the previous load. As soon as the last bit is transmitted, the TSR is loaded with new data from the TXREG (if available). Once the TXREG register transfers the data to the TSR register (occurs in one Tcycle), the TXREG is empty and the TXIF interrupt flag bit is set. The interrupt can be enabled/disabled by setting/clearing the TXIE enable bit. The TXIF flag bit will be set regardless of the state of the TXIE enable bit and cannot be cleared in software. It will reset only when new data is loaded into the TXREG register. While the TXIF flag bit indicates the status of the TXREG register, the TRMT bit shows the status of the TSR register. The TRMT bit is a read only bit that is set when the TSR is empty. No interrupt logic is tied to this bit, so the user has to poll this bit in order to determine if the TSR register is empty. The TSR is not mapped in data memory, so it is not available to the user.

Transmission is enabled by setting the TXEN bit. The actual transmission will not occur until the TXREG register has been loaded with data. The first data bit will be shifted out on the next available rising edge of the clock on the CK line. Data out is stable at the falling edge of the synchronous clock (Figure 21-10). The transmission can also be started by first loading the TXREG register and then setting the TXEN bit. This is advantageous when slow baud rates are selected, since the BRG is kept in RESET when the TXEN, CREN and SREN bits are clear. Setting the TXEN bit will start the BRG, creating a shift clock immediately. Normally, when transmission is first started, the TSR register is empty, so a transfer to the TXREG register will result in an immediate transfer to TSR, resulting in an empty TXREG. Back-to-back transfers are possible.

Clearing the TXEN bit during a transmission will cause the transmission to be aborted and will reset the transmitter. The DT and CK pins will revert to hi-impedance. If either of the CREN or SREN bits are set during a transmission, the transmission is aborted and the DT pin reverts to a hi-impedance state (for a reception). The CK pin will remain an output if the CSRC bit is set (internal clock). The transmitter logic is not reset although it is disconnected from the pins. In order to reset the transmitter, the user has to clear the TXEN bit. If the SREN bit is set (to interrupt an on-going transmission and receive a single word), then after the single word is received, the SREN bit will be cleared and the serial port will revert back to transmitting, since the TXEN bit is still set. The DT line will immediately switch from hi-impedance receive mode to transmit and start driving. To avoid this, the TXEN bit should be cleared.

In order to select 9-bit transmission, the TX9 bit should be set and the ninth bit should be written to the TX9D bit. The ninth bit must be written before writing the 8-bit data to the TXREG register. This is because a data write to the TXREG can result in an immediate transfer of the data to the TSR register (if the TSR is empty). If the TSR was empty and the TXREG was written before writing the “new” value to the TX9D bit, the “present” value of the TX9D bit is loaded.

Steps to follow when setting up a Synchronous Master Transmission:

1. Initialize the SPBRG register for the appropriate baud rate. (Subsection **21.3 "USART Baud Rate Generator (BRG)"**).
2. Enable the synchronous master serial port by setting the SYNC, SPEN and CSRC bits.
3. If interrupts are desired, then set the TXIE bit and configure the RCIP, GIE/GIEH and PEIE/GIEL bits, appropriately.
4. If 9-bit transmission is desired, then set the TX9 bit.
5. Enable the transmission by setting the TXEN bit.
6. If 9-bit transmission is selected, the ninth bit should be loaded in the TX9D bit.
7. Start transmission by loading data to the TXREG register.

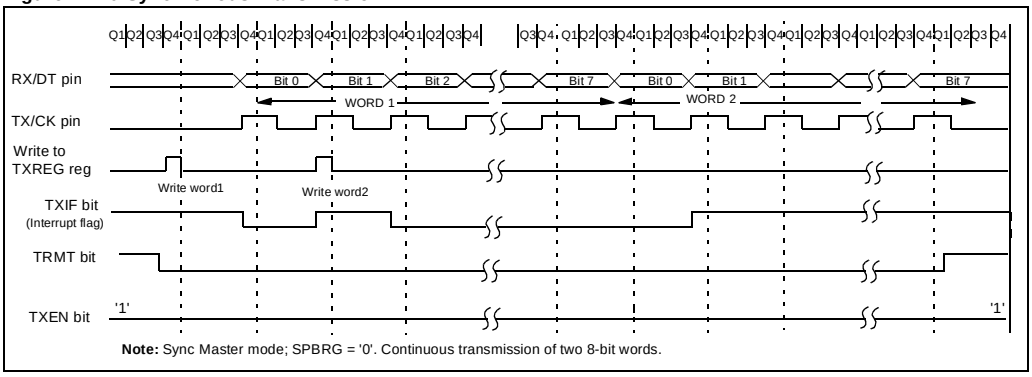
Table 21-8: Registers Associated with Synchronous Master Transmission

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
INTCON	GIE/GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIRx	TXIF (1)								0	0
PIEx	TXIE (1)								0	0
IPRx	TXIP (1)								0	0
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
TXREG	TX7	TX6	TX5	TX4	TX3	TX2	TX1	TX0	0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, — = unimplemented, read as '0'.
Shaded cells are not used for Synchronous Master Transmission.

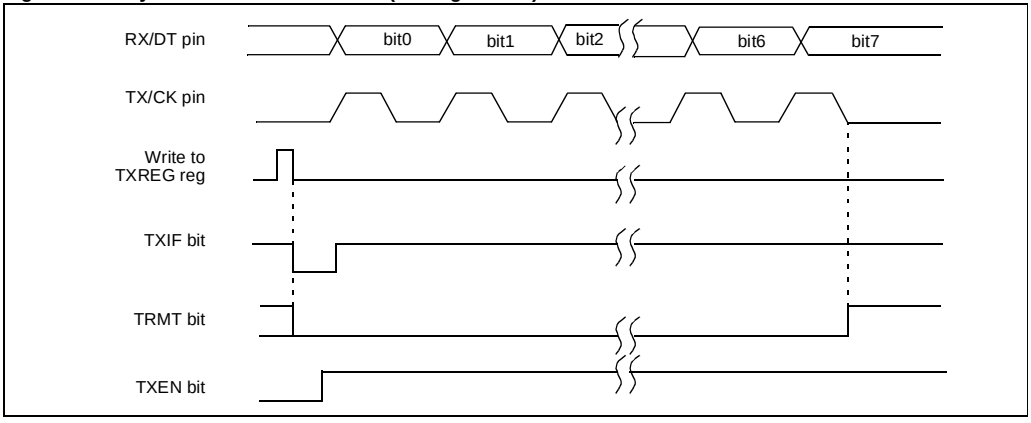
Note 1: The position of this bit is device dependent.

Figure 21-10:Synchronous Transmission



Note: Sync Master mode; SPBRG = '0'. Continuous transmission of two 8-bit words.

Figure 21-11:Synchronous Transmission (Through TXEN)



21.5.1.1 USART Synchronous Master Reception

Once Synchronous mode is selected, reception is enabled by setting either the SREN or CREN bits. Data is sampled on the RX/DT pin on the falling edge of the clock. If the SREN bit is set, then only a single word is received. If the CREN bit is set, the reception is continuous until the CREN bit is cleared. If both bits are set, then the CREN bit takes precedence. After clocking the last serial data bit, the received data in the Receive Shift Register (RSR) is transferred to the RCREG register (if it is empty). When the transfer is complete, the RCIF interrupt flag bit is set. The actual interrupt can be enabled/disabled by setting/clearing the RCIE enable bit. The RCIF flag bit is a read only bit that is cleared by the hardware. In this case, it is cleared when the RCREG register has been read and is empty. The RCREG is a double buffered register (i.e., it is a two-deep FIFO). It is possible for two bytes of data to be received and transferred to the RCREG FIFO and a third byte to begin shifting into the RSR register. On the clocking of the last bit of the third byte, if the RCREG register is still full, then the overrun error bit, OERR, is set and the word in the RSR is lost. The RCREG register can be read twice to retrieve the two bytes in the FIFO. The OERR bit has to be cleared in software (by clearing the CREN bit). If the OERR bit is set, transfers from the RSR to the RCREG are inhibited, so it is essential to clear the OERR bit if it is set. The 9th receive bit is buffered the same way as the receive data. Reading the RCREG register will load the RX9D bit with a new value; therefore, it is essential for the user to read the RCSTA register before reading RCREG in order to not lose the old (previous) information in the RX9D bit.

Steps to follow when setting up a Synchronous Master Reception:

1. Initialize the SPBRG register for the appropriate baud rate. (Subsection **21.3 "USART Baud Rate Generator (BRG)"**)
2. Enable the synchronous master serial port by setting the SYNC, SPEN and CSRC bits.
3. Ensure that the CREN and SREN bits are clear.
4. If interrupts are desired, then set the RCIE bit and configure the RCIP, GIE/GIEH and PEIE/GIEL bits, appropriately.
5. If 9-bit reception is desired, then set the RX9 bit.
6. If a single reception is required, set the SREN bit. For continuous reception, set the CREN bit.
7. The RCIF bit will be set when reception is complete and an interrupt will be generated if the RCIE bit is set.
8. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
9. Read the 8-bit received data by reading the RCREG register.
10. If any error occurred, clear the error by clearing the CREN bit.

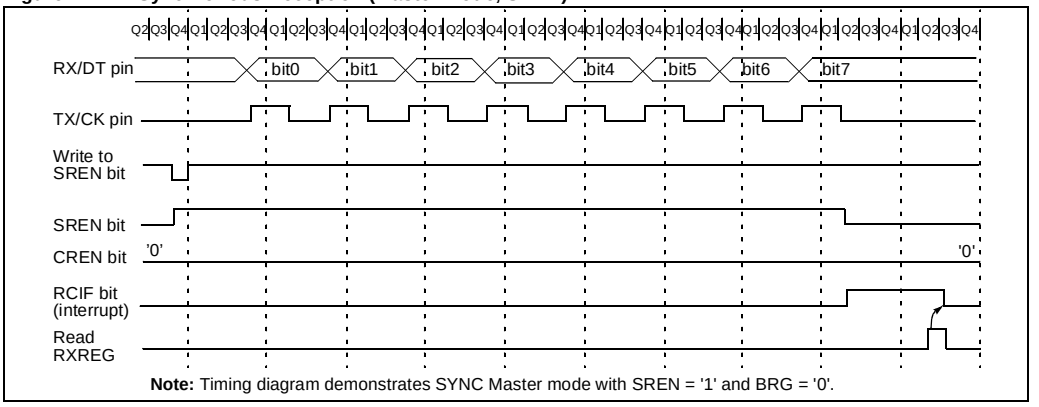
Table 21-9: Registers Associated with Synchronous Master Reception

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIRx	RCIF (1)								0	0
PIEx	RCIE (1)								0	0
IPRx	RCIP (1)								0	0
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
RCREG	RX7	RX6	RX5	RX4	RX3	RX2	RX1	RX0	0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented read as '0'.
Shaded cells are not used for Synchronous Master Reception.

Note 1: The position of this bit is device dependent.

Figure 21-12: Synchronous Reception (Master Mode, SREN)



Note: Timing diagram demonstrates SYNC Master mode with SREN = '1' and BRG = '0'.

21.6 USART Synchronous Slave Mode

Synchronous slave mode differs from the Master mode in the fact that the shift clock is supplied externally at the TX/CK pin (instead of being supplied internally in Master mode). This allows the device to transfer or receive data while in SLEEP mode. Slave mode is entered by clearing the CSRC bit (TXSTA<7>).

21.6.1 USART Synchronous Slave Transmit

The operation of the Synchronous Master and Slave modes are identical, except in the case of the SLEEP mode.

If two words are written to the TXREG and then the SLEEP instruction is executed, the following will occur:

a) The first word will immediately transfer to the TSR register and transmit.
b) The second word will remain in TXREG register.
c) The TXIF flag bit will not be set.
d) When the first word has been shifted out of TSR, the TXREG register will transfer the second word to the TSR and the TXIF flag bit will now be set.
e) If the TXIE enable bit is set, the interrupt will wake the chip from SLEEP and if the global interrupt is enabled, the program will branch to the interrupt vector.

Steps to follow when setting up a Synchronous Slave Transmission:

1. Enable the synchronous slave serial port by setting the SYNC and SPEN bits and clearing the CSRC bit.
2. Clear the CREN and SREN bits.
3. If interrupts are desired, then set the TXIE enable bit and configure the RCIP, GIE/GIEH and PEIE/GIEL bits, appropriately.
4. If 9-bit transmission is desired, then set the TX9 bit.
5. Enable the transmission by setting the TXEN enable bit.
6. If 9-bit transmission is selected, the ninth bit should be loaded into the TX9D bit.
7. Start transmission by loading data to the TXREG register.

Table 21-10: Registers Associated with Synchronous Slave Transmission

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIRx	TXIF (1)								0	0
PIEx	TXIE (1)								0	0
IPRx	TXIP (1)								0	0
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
TXREG	TX7	TX6	TX5	TX4	TX3	TX2	TX1	TX0	0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented read as '0'.
Shaded cells are not used for Synchronous Slave Transmission.
Note 1: The position of this bit is device dependent.

21.6.2 USART Synchronous Slave Reception

The operation of the Synchronous Master and Slave modes is identical, except in the case of the SLEEP mode. Also, bit SREN is a "don't care" in Slave mode.

If receive is enabled, by setting the CREN bit prior to the SLEEP instruction, then a word may be received during SLEEP. On completely receiving the word, the RSR register will transfer the data to the RCREG register and if the RCIE enable bit bit is set, the interrupt generated will wake the chip from SLEEP. If the global interrupt is enabled, the program will branch to the appropriate interrupt vector.

Steps to follow when setting up a Synchronous Slave Reception:

1. Enable the synchronous master serial port by setting the SYNC and SPEN bits and clearing the CSRC bit.
2. If interrupts are desired, then set the RCIE enable bit and configure the RCIP, GIE/GIEH and PEIE/GIEL bits, appropriately.
3. If 9-bit reception is desired, then set the RX9 bit.
4. To enable reception, set the CREN enable bit.
5. The RCIF bit will be set when reception is complete and an interrupt will be generated, if the RCIE bit is set.
6. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
7. Read the 8-bit received data by reading the RCREG register.
8. If any error occurred, clear the error by clearing the CREN bit.

Table 21-11: Registers Associated with Synchronous Slave Reception

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIRx	RCIF (1)								0	0
PIEx	RCIE (1)								0	0
IPRx	RCIP (1)								0	0
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
RCREG	RX7	RX6	RX5	RX4	RX3	RX2	RX1	RX0	0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented read as '0'.
Shaded cells are not used for Synchronous Slave Reception.

Note 1: The position of this bit is device dependent.

21.7 Initialization

Example 21-2 is an initialization routine for Asynchronous Transmitter/Receiver mode. Example 21-3 is for the Synchronous mode. In both examples, the data is 8 bits, and the value to load into the SPBRG register is dependent on the desired baud rate and the device frequency. Example 21-4 is an initialization of the Addressable USART in 9-bit address detect mode.

Example 21-2: Asynchronous Transmitter/Receiver

```
  MOVLW  baudrate       ; Set Baudrate
  MOVWF  SPBRG
  MOVLW  0x20           ; 8-bit transmit, transmitter enabled,
  MOVWF  TXSTA          ; asynchronous mode, low speed mode
  CLRF   PIR1           ; Clear all iterrupt flags
                        ;   including AUSART TX & RX
  BSF    PIE1,TXIE      ; Enable transmit interrupts
  BSF    PIE1,RCIE      ; Enable receive interrupts
  MOVLW  0x90           ; 8-bit receive, receiver enabled,
  MOVWF  RCSTA          ; serial port enabled
```

Example 21-3: Synchronous Transmitter/Receiver

```
MOVLW  baudrate      ; Set Baudrate
MOVWF  SPBRG
MOVLW  0xB0          ; Synchronous Master,8-bit transmit,
MOVWF  TXSTA         ; transmitter enabled, low speed mode
CLRF   PIR1          ; Clear all iterrupt flags
                     ;   including AUSART TX & RX
BSF    PIE1,TXIE     ; Enable transmit interrupts
BSF    PIE1,RCIE     ; Enable receive interrupts
MOVLW  0x90          ; 8-bit receive, receiver enabled,
MOVWF  RCSTA         ; continuous receive, serial port enabled
```

Example 21-4: Asynchronous 9-bit Transmitter/Receiver (Address Detect Enabled)

```
  MOVLW  baudrate       ; Set Baudrate
  MOVWF  SPBRG
  MOVLW  0x60           ; 9-bit transmit, transmitter enabled,
  MOVWF  TXSTA          ; asynchronous mode, low speed mode
  CLRF   PIR1           ; Clear all iterrupt flags
                        ;   including AUSART TX & RX
  BSF    PIE1,TXIE      ; Enable transmit interrupts
  BSF    PIE1,RCIE      ; Enable receive interrupts
  MOVLW  0xD8           ; 9-bit, Address Detect Enable
  MOVWF  RCSTA          ; serial port enabled
```

21.8 Design Tips

Question 1: ***Using the Asynchronous mode I am getting a lot of transmission errors.***

Answer 1:

The most common reasons are

1. You have incorrectly calculated the value to load in to the SPBRG register.
2. The sum of the baud errors for the transmitter and receiver is too high.

Question 2: ***The PICmicro device is not receiving the data transmitted even though there are good levels on the Addressable USART pins.***

Answer 2:

Ensure that the Address Detect Enable bit is at the desired setting.

21.9 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is, they may be written for the Base-Line, the Mid-Range or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to this section are:

Title	Application Note #
Serial Port Utilities	AN547
Servo Control of a DC Brush Motor	AN532
Brush-DC Servomotor Implementation using PIC17C756A	AN718

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

21.10 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Addressable USART module description.

Section 22. CAN

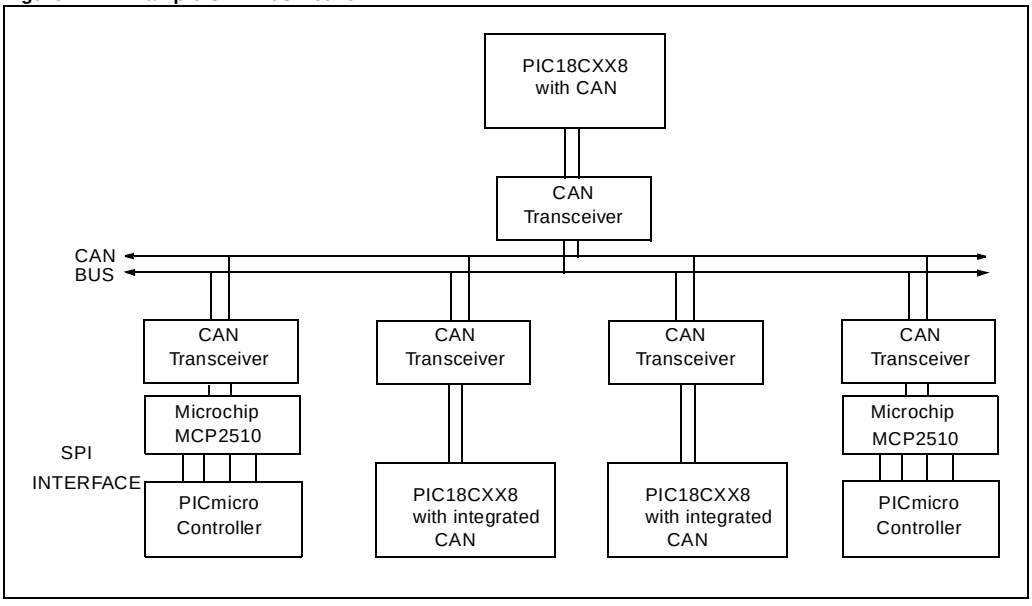
HIGHLIGHTS

This section of the manual contains the following major topics:

22.1 Introduction

The Controller Area Network (CAN) module is a serial interface useful for communicating with other peripherals or microcontroller devices. This interface/protocol was designed to allow communications within noisy environments. Figure 22-1 shows an example CAN Bus network.

Figure 22-1: Example CAN Bus Network



22.2 Control Registers for the CAN Module

There are many registers associated with the CAN module. Descriptions of these registers are grouped into sections. These sections are:

- Control and Status Registers
- Transmit Buffer Registers
- Receive Buffer Registers
- Baud Rate Control Registers
- Interrupt Status and Control Registers

22.2.1 CAN Control and Status Registers

This section shows the CAN Control and Status registers.

Register 22-1: CANCON: CAN Control Register

R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
REQOP2	REQOP1	REQOP0	ABAT	WIN2	WIN1	WIN0	—
bit 7							bit 0

bit 7 - 5 **REQOP2:REQOP0:** Request CAN Operation mode bits

1xx = Request Configuration mode
011 = Request Listen Only mode
010 = Request Loopback mode
001 = Request Disable mode
000 = Request Normal mode

bit 4 **ABAT:** Abort All Pending Transmissions bit

1 = Abort All Pending Transmissions (in all transmit buffers)
0 = Transmissions proceeding as normal, or all Transmissions aborted

Note: This bit will automatically be cleared when all transmissions are aborted.

bit 3 - 1 **WIN2:WIN0:** Window Address bits

This selects which of the CAN buffers to switch into the access bank area. This allows access to the buffer registers from any data memory bank. After a frame has caused an interrupt, the ICODE2:ICODE0 bits can be copied to the WIN2:WIN0 bits to select the correct buffer.

111 = Receive Buffer 0
110 = Receive Buffer 0
101 = Receive Buffer 1
100 = Transmit Buffer 0
011 = Transmit Buffer 1
010 = Transmit Buffer 2
001 = Receive Buffer 0
000 = Receive Buffer 0

bit 0 **Unimplemented:** Read as '0'

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-2: CANSTAT: CAN Status Register

R-1	R-0	R-0	U-0	R-0	R-0	R-0	U-0
OPMODE2	OPMODE1	OPMODE0	—	ICODE2	ICODE1	ICODE0	—
bit 7							bit 0

bit 7-5 **OPMODE2:OPMODE0:** Operation Mode Status bits

111 = Reserved
110 = Reserved
101 = Reserved
100 = Configuration mode
011 = Listen Only mode
010 = Loopback mode
001 = Disable mode
000 = Normal mode

Note: Before the device goes into SLEEP mode, select Disable Mode.

bit 4 **Unimplemented:** Read as '0'

bit 3-1 **ICODE2:ICODE0:** Interrupt Code bits

When an interrupt occurs, a prioritized coded Interrupt value will be present in the ICODE2:ICODE0 bits. These codes indicate the source of the interrupt. The ICODE2:ICODE0 bits can be copied to the WIN2:WIN0 bits to select the correct buffer to map into the Access Bank area.

111 = Wake-up on Interrupt
110 = RXB0 Interrupt
101 = RXB1 Interrupt
100 = TXB0 Interrupt
011 = TXB1 Interrupt
010 = TXB2 Interrupt
001 = Error Interrupt
000 = No Interrupt

bit 0 **Unimplemented:** Read as '0'

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-3: COMSTAT: Communication Status Register

R/C-0	R/C-0	R-0	R-0	R-0	R-0	R-0	R-0
RX0OVFL	RX1OVFL	TXBO	TXBP	RXBP	TXWARN	RXWARN	EWARN
bit 7							bit 0

bit 7 **RX0OVFL:** Receive Buffer 0 Overflow bit

1 = Receive Buffer 0 Overflowed
0 = Receive Buffer 0 has not overflowed.

bit 6 **RX1OVFL:** Receive Buffer 1 Overflow bit

1 = Receive Buffer 1 Overflowed
0 = Receive Buffer 1 has not overflowed

bit 5 **TXB0:** Transmitter Bus Off bit

1 = Transmit Error Counter > 255
0 = Transmit Error Counter ≤ 255

bit 4 **TXBP:** Transmitter Bus Passive bit

1 = Transmission Error Counter >127
0 = Transmission Error Counter ≤127

bit 3 **RXBP:** Receiver Bus Passive bit

1 = Receive Error Counter > 127
0 = Receive Error Counter ≤127

bit 2 **TXWARN:** Transmitter Warning bit

1 = Transmit Error Counter > 95
0 = Transmit Error Counter ≤ 95

bit 1 **RXWARN:** Receiver Warning bit

1 = Receive Error Counter > 95
0 = Receive Error Counter ≤ 95

bit 0 **EWARN:** Error Warning bit

This bit is a flag of the RXWARN and TXWARN bits

1 = The RXWARN or the TXWARN bits are set
0 = Neither the RXWARN or the TXWARN bits are set

Legend			
R = Readable bit	C = Clearable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

22.2.2 CAN Transmit Buffer Registers

This section describes the CAN Transmit Buffer Register and the associated Transmit Buffer Control Registers.

Register 22-4: TXB0CON: Transmit Buffer 0 Control Register
TXB1CON: Transmit Buffer 1 Control Register
TXB2CON: Transmit Buffer 2 Control Register

U-0	R-0	R-0	R-0	R/W-0	U-0	R/W-0	R/W-0
—	TXABT	TXLARB	TXERR	TXREQ	—	TXPRI1	TXPRI0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6 **TXABT:** Transmission Aborted Status bit

1 = Message was aborted
0 = Message completed transmission successfully

bit 5 **TXLARB:** Transmission Lost Arbitration Status bit

1 = Message lost arbitration while being sent
0 = Message did not lose arbitration while being sent

bit 4 **TXERR:** Transmission Error detected Status bit

1 = A bus error occurred while the message was being sent
0 = A bus error did not occur while the message was being sent

bit 3 **TXREQ:** Transmit Request Status bit

1 = Requests sending a message. Clears the TXABT, TXLARB, and TXERR bits.
0 = Automatically cleared when the message is successfully sent

Note: Clearing this bit in software, while the bit is set will request a message abort.

bit 2 **Unimplemented:** Read as '0'

bit 1:0 **TXPRI1:TXPRI0:** Transmit Priority bits

11 = Priority Level 3 (Highest Priority)
10 = Priority Level 2
01 = Priority Level 1
00 = Priority Level 0 (Lowest Priority)

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-5: TXB0SIDH: Transmit Buffer 0 Standard Identifier High Byte Register
TXB1SIDH: Transmit Buffer 1 Standard Identifier High Byte Register
TXB2SIDH: Transmit Buffer 2 Standard Identifier High Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

bit 7-0 **SID10:SID3:** Standard Identifier bits

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-6: TXB0SIDL: Transmit Buffer 0 Standard Identifier Low Byte Register
TXB1SIDL: Transmit Buffer 1 Standard Identifier Low Byte Register
TXB2SIDL: Transmit Buffer 2 Standard Identifier Low Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID2	SID1	SID0	—	EXIDEN	—	EID17	EID16
bit 7							bit 0

bit 7-5 **SID2:SID0:** Standard Identifier bits

bit 4 **Unimplemented:** Read as '0'

bit 3 **EXIDEN:** Extended Identifier Enable bit

1 = Message will transmit Extended ID
0 = Message will transmit Standard ID

bit 2 **Unimplemented:** Read as '0'

bit 1-0 **EID17:EID16:** Extended Identifier bits

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-7: TXB0EIDH: Transmit Buffer 0 Extended Identifier High Byte Register
TXB1EIDH: Transmit Buffer 1 Extended Identifier High Byte Register
TXB2EIDH: Transmit Buffer 2 Extended Identifier High Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8
bit 7							bit 0

bit 7-0 **EID15:EID8:** Extended Identifier bits EID15 to EID8

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-8: TXB0EIDL: Transmit Buffer 0 Extended Identifier Low Byte Register
TXB1EIDL: Transmit Buffer 1 Extended Identifier Low Byte Register
TXB2EIDL: Transmit Buffer 2 Extended Identifier Low Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0
bit 7							bit 0

bit 7-0 **EID7:EID0:** Extended Identifier bits EID7 to EID0

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-9: TXB0Dm: Transmit Buffer 0 Data Field Byte m Register
TXB1Dm: Transmit Buffer 1 Data Field Byte m Register
TXB2Dm: Transmit Buffer 2 Data Field Byte m Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
TXBnDm7	TXBnDm6	TXBnDm5	TXBnDm4	TXBnDm3	TXBnDm2	TXBnDm1	TXBnDm0
bit 7							bit 0

bit 1-0 **TXBnDm7:TXBnDm0:** Transmit Buffer n Data Field Byte m bits (where 0 < n < 3 and 0 < m < 8)

Each Transmit Buffer has an array of registers. For example Transmit buffer 0 has 7 registers: TXB0D1 to TXB0D7.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-10: TXB0DLC: Transmit Buffer 0 Data Length Code Register
TXB1DLC: Transmit Buffer 1 Data Length Code Register
TXB2DLC: Transmit Buffer 2 Data Length Code Register

U-0	R/W-x	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x
—	TXRTR	—	—	DLC3	DLC2	DLC1	DLC0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6 **TXRTR:** Transmission Frame Remote Transmission Request bit

1 = Transmitted Message will have TXRTR bit set
0 = Transmitted Message will have TXRTR bit cleared.

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **DLC3:DLC0:** Data Length Code bits

1111 = Reserved
1110 = Reserved
1101 = Reserved
1100 = Reserved
1011 = Reserved
1010 = Reserved
1001 = Reserved
1000 = Data Length = 8 bytes
0111 = Data Length = 7 bytes
0110 = Data Length = 6 bytes
0101 = Data Length = 5 bytes
0100 = Data Length = 4 bytes
0011 = Data Length = 3 bytes
0010 = Data Length = 2 bytes
0001 = Data Length = 1 bytes
0000 = Data Length = 0 bytes

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-11: TXERRCNT: Transmit Error Count Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0
bit 7							bit 0

bit 7-0 **TEC7:TEC0:** Transmit Error Counter bits

This register contains a value which is derived from the rate at which errors occur. When the error count overflows, the bus off state occurs. When the bus has an occurrence of 11 consecutive recessive bits, the counter value is cleared.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

22.2.3 CAN Receive Buffer Registers

This section shows the Receive buffer registers with their associated control registers.

Register 22-12: RXB0CON: Receive Buffer 0 Control Register

R/C-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R-0	R/W-0
RXFUL	RXM1	RXM0	—	RXRTRRO	RX0DBEN	JTOFF	FILHIT0
bit 7							bit 0

bit 7 **RXFUL:** Receive Full status bit

1 = Receive buffer contains a valid received message
0 = Receive buffer is open to receive a new message

Note: This bit is set by the CAN module and should be cleared by software after the buffer is read.

bit 6-5 **RXM1:RXM0:** Receive Buffer Mode bits

11 = Receive all messages (including those with errors)
10 = Receive only valid messages with extended identifier
01 = Receive only valid messages with standard identifier
00 = Receive all valid messages

bit 4 **Unimplemented:** Read as '0'

bit 3 **RXRTRRO:** Receive Remote Transfer Request Read Only bit

1 = Remote Transfer Request
0 = No Remote Transfer Request

bit 2 **RX0DBEN:** Receive Buffer 0 Double Buffer Enable bit

1 = Receive Buffer 0 overflow will write to Receive Buffer 1
0 = No Receive Buffer 0 overflow to Receive Buffer 1

bit 1 **JTOFF:** Jump Table offset bit

1 = Allows Jump Table offset between 6 and 7
0 = Allows Jump Table offset between 1 and 0

bit 0 **FILHIT0:** Filter Hit bit

This bit indicates which acceptance filter enabled the message reception into receive buffer 0.

1 = Acceptance Filter 1 (RXF1)
0 = Acceptance Filter 0 (RXF0)

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
C = Clearable bit			
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-13: RXB1CON: Receive Buffer 1 Control Register

R/C-0	R/W-0	R/W-0	U-0	R-0	R-0	R-0	R-0
RXFUL	RXM1	RXM0	—	RXRTRRO	FILHIT2	FILHIT1	FILHIT0
bit 7							bit 0

bit 7 **RXFUL:** Receive Full status bit

1 = Receive buffer contains a valid received message
0 = Receive buffer is open to receive a new message

Note: This bit is set by the CAN module and should be cleared by software after the buffer is read.

bit 6-5 **RXM1:RXM0:** Receive Buffer Mode bits

11 = Receive all messages (including those with errors)
10 = Receive only valid messages with extended identifier
01 = Receive only valid messages with standard identifier
00 = Receive all valid messages

bit 4 **Unimplemented:** Read as '0'

bit 3 **RXRTRRO:** Receive Remote Transfer Request bit (read only)

1 = Remote Transfer Request
0 = No Remote Transfer Request

bit 2-0 **FILHIT2:FILHIT0:** Filter Hit bits

These bits indicate which acceptance filter enabled the last message reception into Receive Buffer 1

111 = Reserved
110 = Reserved
101 = Acceptance Filter 5 (RXF5)
100 = Acceptance Filter 4 (RXF4)
011 = Acceptance Filter 3 (RXF3)
010 = Acceptance Filter 2 (RXF2)
001 = Acceptance Filter 1 (RXF1) only possible when RX0DBEN bit is set
000 = Acceptance Filter 0 (RXF0) only possible when RX0DBEN bit is set

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
C = Clearable bit			
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-14: RXB0SIDH: Receive Buffer 0 Standard Identifier High Byte Register
RXB1SIDH: Receive Buffer 1 Standard Identifier High Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

bit 7-0 **SID10:SID3:** Standard Identifier bits

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-15: RXB0SIDL: Receive Buffer 0 Standard Identifier Low Byte Register
RXB1SIDL: Receive Buffer 1 Standard Identifier Low Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	U-0	R/W-x	R/W-x
SID2	SID1	SID0	SRR	EXID	—	EID17	EID16
bit 7							bit 0

bit 7-5 **SID2:SID0:** Standard Identifier bits SID2 to SID1

bit 4 **SRR:** Substitute Remove Request bit (only when EXID = '1')

1 = Remote Transfer Request Occurred
0 = No Remote Transfer Request Occurred

bit 3 **EXID:** Extended Identifier bit

1 = Received message is an Extended Data Frame
0 = Received message is a Standard Data Frame

bit 2 **Unimplemented:** Read as '0'

bit 1-0 **EID17:EID16:** Extended Identifier bits EID17 to EID16

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-16: RXB0EIDH: Receive Buffer 0 Extended Identifier High Byte Register
RXB1EIDH: Receive Buffer 1 Extended Identifier High Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8
bit 7							bit 0

bit 7-0 **EID15:EID8:** Extended Identifier bits EID15 to EID8

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-17: RXB0EIDL: Transmit Buffer 0 Extended Identifier Low Byte Register
RXB1EIDL: Transmit Buffer 1 Extended Identifier Low Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0
bit 7							bit 0

bit 7-0 **EID7:EID0:** Extended Identifier bits EID7 to EID0

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-18: RXB0DLC: Receive Buffer 0 Data Length Code Register
RXB1DLC: Receive Buffer 1 Data Length Code Register

U-x	R/W-x	R-x	R-x	R-x	R-x	R-x	R-x
—	RXRTR	RB1	RB0	DLC3	DLC2	DLC1	DLC0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6 **RXRTR:** Receiver Remote Transmission Request bit

1 = Remote Transfer Request
0 = No Remote Transfer Request

bit 5 **RB1:** Reserved bit 1

Reserved by CAN Specification and read as '0'

bit 4 **RB0:** Reserved bit 0

Reserved by CAN Specification and read as '0'

bit 3-0 **DLC3:DLC0:** Data Length Code bits

1111 = Invalid
1110 = Invalid
1101 = Invalid
1100 = Invalid
1011 = Invalid
1010 = Invalid
1001 = Invalid
1000 = Data Length = 8 bytes
0111 = Data Length = 7 bytes
0110 = Data Length = 6 bytes
0101 = Data Length = 5 bytes
0100 = Data Length = 4 bytes
0011 = Data Length = 3 bytes
0010 = Data Length = 2 bytes
0001 = Data Length = 1 bytes
0000 = Data Length = 0 bytes

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-19: RXB0Dm: Receive Buffer 0 Data Field Byte m Register
RXB1Dm: Receive Buffer 1 Data Field Byte m Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
RXBnDm7	RXBnDm6	RXBnDm5	RXBnDm4	RXBnDm3	RXBnDm2	RXBnDm1	RXBnDm0
bit 7							bit 0

bit 7-0 **RXBnDm7:RXBnDm0:** Receive Buffer n Data Field Byte m bits (where 0 < n < 1 and 0 < m < 7)

Each Receive Buffer has an array of registers. For example Receive buffer 0 has 7 registers: RXB0D1 to RXB0D7.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-20: RXERRCNT: Receive Error Count Register

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
REC7	REC6	REC5	REC4	REC3	REC2	REC1	REC0
bit 7							bit 0

bit 7-0 **REC7:REC0:** Receive Error Counter bits

This register contains the number of errors that occurred for the Reception of this buffers message.

When the error count overflows, the bus off state occurs. When the bus has 256 occurrences of 11 consecutive recessive bits, the counter value is cleared.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

22.2.4 Message Acceptance Filters

This subsection describes the Message Acceptance filters.

Register 22-21: RXF0SIDH: Receive Acceptance Filter 0 Std. Identifier Filter High Byte
RXF1SIDH: Receive Acceptance Filter 1 Std. Identifier Filter High Byte
RXF2SIDH: Receive Acceptance Filter 2 Std. Identifier Filter High Byte
RXF3SIDH: Receive Acceptance Filter 3 Std. Identifier Filter High Byte
RXF4SIDH: Receive Acceptance Filter 4 Std. Identifier Filter High Byte
RXF5SIDH: Receive Acceptance Filter 5 Std. Identifier Filter High Byte

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

bit 7-0 **SID10:SID3:** Standard Identifier Filter bits SID10 to SID3

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-22: RXF0SIDL: Receive Acceptance Filter 0 Std. Identifier Filter Low Byte
RXF1SIDL: Receive Acceptance Filter 1 Std. Identifier Filter Low Byte
RXF2SIDL: Receive Acceptance Filter 2 Std. Identifier Filter Low Byte
RXF3SIDL: Receive Acceptance Filter 3 Std. Identifier Filter Low Byte
RXF4SIDL: Receive Acceptance Filter 4 Std. Identifier Filter Low Byte
RXF5SIDL: Receive Acceptance Filter 5 Std. Identifier Filter Low Byte

R/W-x	R/W-x	R/W-x	U-0	R/W-x	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	EXIDEN	—	EID17	EID16
bit 7							bit 0

bit 7-5 **SID2:SID0:** Standard Identifier Filter bits SID2 to SID0

bit 4 **Unimplemented:** Read as '0'

bit 3 **EXIDEN:** Extended Identifier Filter Enable bit

1 = Message will transmit Extended ID
0 = Message will not transmit Extended ID.

bit 2 **Unimplemented:** Read as '0'

bit 1-0 **EID17:EID16:** Extended Identifier Filter bits EID17 to EID16

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-23: RXF0EIDH: Receive Acceptance Filter 0 Extended Identifier High Byte
RXF1EIDH: Receive Acceptance Filter 1 Extended Identifier High Byte
RXF2EIDH: Receive Acceptance Filter 2 Extended Identifier High Byte
RXF3EIDH: Receive Acceptance Filter 3 Extended Identifier High Byte
RXF4EIDH: Receive Acceptance Filter 4 Extended Identifier High Byte
RXF5EIDH: Receive Acceptance Filter 5 Extended Identifier High Byte

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8
bit 7							bit 0

bit 7-0 **EID15:EID8:** Extended Identifier Filter bits EID15 to EID8

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR reset	’1’ = bit is set	’0’ = bit is cleared	x = bit is unknown

Register 22-24: RXB0EIDL: Receive Buffer 0 Extended Identifier Low Byte Register
RXB1EIDL: Receive Buffer 1 Extended Identifier Low Byte Register
RXB2EIDL: Receive Buffer 2 Extended Identifier Low Byte Register
RXB3EIDL: Receive Buffer 3 Extended Identifier Low Byte Register
RXB4EIDL: Receive Buffer 4 Extended Identifier Low Byte Register
RXB5EIDL: Receive Buffer 5 Extended Identifier Low Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0
bit 7							bit 0

bit 7-0 **EID7:EID0:** Extended Identifier Filterbits EID7 to EID0

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR reset	’1’ = bit is set	’0’ = bit is cleared	x = bit is unknown

Register 22-25: RXM0SIDH: Receive Acceptance Mask 0 Std. Identifier Mask High Byte Register
RXM1SIDH: Receive Acceptance Mask 1 Std. Identifier Mask High Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7							bit 0

bit 7-0 **SID10:SID3:** Standard Identifier Mask bits SID10 to SID3

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-26: RXM0SIDL: Receive Acceptance Mask 0 Std. Identifier Mask Low Byte Register
RXM1SIDL: Receive Acceptance Mask 1 Std. Identifier Mask Low Byte Register

R/W-x	R/W-x	R/W-x	U-0	U-0	U-0	R/W-x	R/W-x
SID2	SID1	SID0	—	—	—	EID17	EID16
bit 7							bit 0

bit 7-5 **SID2:SID0:** Standard Identifier Mask bits SID2 to SID0

bit 4-2 **Unimplemented:** Read as '0'

bit 1-0 **EID17:EID16:** Extended Identifier Mask bits EID17 to EID16

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-27: RXM0EIDH: Receive Acceptance Mask 0 Extended Identifier Mask High Byte Register
RXM1EIDH: Receive Acceptance Mask 1 Extended Identifier Mask High Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID15	EID14	EID13	EID12	EID11	EID10	EID9	EID8
bit 7							bit 0

bit 1-0 **EID15:EID8:** Extended Identifier Mask bits EID15 to EID8

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-28: RXM0EIDL: Receive Acceptance Mask 0 Extended Identifier Mask Low Byte Register
RXM1EIDL: Receive Acceptance Mask 1 Extended Identifier Mask Low Byte Register

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID7	EID6	EID5	EID4	EID3	EID2	EID1	EID0
bit 7							bit 0

bit 1-0 **EID7:EID0:** Extended Identifier Mask bits EID7 to EID0

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

22.2.5 CAN Baud Rate Registers

This subsection describes the CAN baud rate registers.

Register 22-29: BRGCON1: Baud Rate Control Register 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
bit 7							bit 0

bit 7-6 **SJW1:SJW0:** Synchronized Jump Width bits

11 = Synchronization Jump Width Time = 4 x TQ
10 = Synchronization Jump Width Time = 3 x TQ
01 = Synchronization Jump Width Time = 2 x TQ
00 = Synchronization Jump Width Time = 1 x TQ

bit 5-0 **BRP5:BRP0:** Baud Rate Prescaler bits

11111 = TQ = (2 x 64)/FOSC
11110 = TQ = (2 x 63)/FOSC
:
:
00001 = TQ = (2 x 2)/FOSC
00000 = TQ = (2 x 1)/FOSC

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Note: This register is only accessible in configuration mode (see Section **22.7.1**).

Register 22-30: BRGCON2: Baud Rate Control Register 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SEG2PHTS	SAM	SEG1PH2	SEG1PH1	SEG1PH0	PRSEG2	PRSEG1	PRSEG0
bit 7							bit 0

bit 7 **SEG2PHTS:** Phase Segment 2 Time Select bit

1 = Freely programmable
0 = Maximum of PHEG1 or Information Processing Time (IPT), whichever is greater

bit 6 **SAM:** Sample of the CAN bus line bit

1 = Bus line is sampled three times at the sample point
0 = Bus line is sampled once at the sample point

bit 5-3 **SEG1PH2:SEG1PH0:** Phase segment 1 bits

111 = Phase segment 1 Time = 8 x TQ
110 = Phase segment 1 Time= 7 x TQ
101 = Phase segment 1 Time = 6 x TQ
100 = Phase segment 1 Time = 5 x TQ
011 = Phase segment 1 Time = 4 x TQ
010 = Phase segment 1 Time = 3 x TQ
001 = Phase segment 1 Time = 2 x TQ
000 = Phase segment 1 Time = 1 x TQ

bit 2-0 **PRSEG2:PRSEG0:** Propagation Time Select bits

111 = Propagation Time = 8 x TQ
110 = Propagation Time = 7 x TQ
101 = Propagation Time = 6 x TQ
100 = Propagation Time = 5 x TQ
011 = Propagation Time = 4 x TQ
010 = Propagation Time = 3 x TQ
001 = Propagation Time = 2 x TQ
000 = Propagation Time = 1 x TQ

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Note: This register is only accessible in configuration mode (see Section **22.7.1**).

Register 22-31: BRGCON3: Baud Rate Control Register 3

U-0	R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
—	WAKFIL	—	—	—	SEG2PH2	SEG2PH1	SEG2PH0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6 **WAKFIL:** Selects CAN Bus Line Filter for Wake-up bit

1 = Use CAN bus line filter for wake-up
0 = CAN bus line filter is not used for wake-up

bit 5-3 **Unimplemented:** Read as '0'

bit 2-0 **SEG2PH2:SEG2PH0:** Phase Segment 2 Time Select bits

111 = Phase Segment 2 Time = 8 x TQ
110 = Phase Segment 2 Time = 7 x TQ
101 = Phase Segment 2 Time = 6 x TQ
100 = Phase Segment 2 Time = 5 x TQ
011 = Phase Segment 2 Time = 4 x TQ
010 = Phase Segment 2 Time = 3 x TQ
001 = Phase Segment 2 Time = 2 x TQ
000 = Phase Segment 2 Time = 1 x TQ

Note: Ignored if SEG2PHTS bit is clear.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

22.2.6 CAN Module I/O Control Register

This subsection describes the CAN Module I/O Control register.

Register 22-32: CIOCON: CAN I/O Control Register

R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0	U-0	U-0
TX1SRC	TX1EN	ENDRHI	CANCAP	—	—	—	—
bit 7							bit 0

bit 7 **TX1SRC**: TX1 Pin Data Source

1 = TX1 pin will output the CAN clock
0 = TX1 pin will output $\overline{TXD}$

bit 6 **TX1EN:** TX1 Pin Enable

1 = TX1 pin will output $\overline{TXD}$ or CAN clock
0 = TX1 pin will have digital I/O function

bit 5 **ENDRHI:** Enable Drive High

1 = TX0, TX1 pins will drive Vdd when recessive
0 = TX0, TX1 pins will tri-state when recessive

bit 4 **CANCAP:** CAN Message Receive Capture Enable

1 = Enable CAN capture
0 = Disable CAN capture

bit 3-0 **Unimplemented:** Read as '0'

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

22.2.7 CAN Interrupt Registers

This subsection documents the CAN Registers which are associated to Interrupts.

Register 22-33: PIR3: Peripheral Interrupt Flag Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRXIF	WAKIF	ERRIF	TXB2IF	TXB1IF	TXB0IF	RXB1IF	RXB0IF
bit 7							bit 0

bit 7 **IRXIF:** CAN Invalid Received message Interrupt Flag bit

1 = An invalid message has occurred on the CAN bus
0 = No invalid message on CAN bus

bit 6 **WAKIF:** CAN Bus Activity Wake-up Interrupt Flag bit

1 = Activity on CAN bus has occurred
0 = No activity on CAN bus

bit 5 **ERRIF:** CAN bus Error Interrupt Flag bit

1 = An error has occurred in the CAN module (multiple sources)
0 = No CAN module errors

bit 4 **TXB2IF:** CAN Transmit Buffer 2 Interrupt Flag bit

1 = Transmit Buffer 2 has completed transmission of a message, and may be re-loaded
0 = Transmit Buffer 2 has not completed transmission of a message

bit 3 **TXB1IF:** CAN Transmit Buffer 1 Interrupt Flag bit

1 = Transmit Buffer 1 has completed transmission of a message, and may be re-loaded
0 = Transmit Buffer 1 has not completed transmission of a message

bit 2 **TXB0IF:** CAN Transmit Buffer 0 Interrupt Flag bit

1 = Transmit Buffer 0 has completed transmission of a message, and may be re-loaded
0 = Transmit Buffer 0 has not completed transmission of a message

bit 1 **RXB1IF:** CAN Receive Buffer 1 Interrupt Flag bit

1 = Receive Buffer 1 has received a new message
0 = Receive Buffer 1 has not received a new message

bit 0 **RXB0IF:** CAN Receive Buffer 0 Interrupt Flag bit

1 = Receive Buffer 0 has received a new message
0 = Receive Buffer 0 has not received a new message

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR reset	’1’ = bit is set	’0’ = bit is cleared	x = bit is unknown

Register 22-34: PIE3: Peripheral Interrupt Enable Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRXIE	WAKIE	ERRIE	TXB2IE	TXB1IE	TXB0IE	RXB1IE	RXB0IE
bit 7							bit 0

bit 7 **IRXIE:** CAN Invalid Received Message Interrupt Enable bit

1 = Enable invalid message received interrupt
0 = Disable invalid message received interrupt

bit 6 **WAKIE:** CAN Bus Activity Wake-up Interrupt Enable bit

1 = Enable Bus Activity Wake-up Interrupt
0 = Disable Bus Activity Wake-up Interrupt

bit 5 **ERRIE:** CAN Bus Error Interrupt Enable bit

1 = Enable CAN bus Error Interrupt
0 = Disable CAN bus Error Interrupt

bit 4 **TXB2IE:** CAN Transmit Buffer 2 Interrupt Enable bit

1 = Enable Transmit Buffer 2 Interrupt
0 = Disable Transmit Buffer 2 Interrupt

bit 3 **TXB1IE:** CAN Transmit Buffer 1 Interrupt Enable bit

1 = Enable Transmit Buffer 1 Interrupt
0 = Disable Transmit Buffer 1 Interrupt

bit 2 **TXB0IE:** CAN Transmit Buffer 0 Interrupt Enable bit

1 = Enable Transmit Buffer 0 Interrupt
0 = Disable Transmit Buffer 0 Interrupt

bit 1 **RXB1IE:** CAN Receive Buffer 1 Interrupt Enable bit

1 = Enable Receive Buffer 1 Interrupt
0 = Disable Receive Buffer 1 Interrupt

bit 0 **RXB0IE:** CAN Receive Buffer 0 Interrupt Enable bit

1 = Enable Receive Buffer 0 Interrupt
0 = Disable Receive Buffer 0 Interrupt

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 22-35: IPR3: Peripheral Interrupt Priority Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRXIP	WAKIP	ERRIP	TXB2IP	TXB1IP	TXB0IP	RXB1IP	RXB0IP
bit 7							bit 0

bit 7 **IRXIP:** CAN Invalid Received Message Interrupt Priority bit

1 = High Priority
0 = Low Priority

bit 6 **WAKIP:** CAN Bus Activity Wake-up Interrupt Priority bit

1 = High Priority
0 = Low Priority

bit 5 **ERRIP:** CAN Bus Error Interrupt Priority bit

1 = High Priority
0 = Low Priority

bit 4 **TXB2IP:** CAN Transmit Buffer 2 Interrupt Priority bit

1 = High Priority
0 = Low Priority

bit 3 **TXB1IP:** CAN Transmit Buffer 1 Interrupt Priority bit

1 = High Priority
0 = Low Priority

bit 2 **TXB0IP:** CAN Transmit Buffer 0 Interrupt Priority bit

1 = High Priority
0 = Low Priority

bit 1 **RXB1IP:** CAN Receive Buffer 1 Interrupt Priority bit

1 = High Priority
0 = Low Priority

bit 0 **RXB0IP:** CAN Receive Buffer 0 Interrupt Priority bit

1 = High Priority
0 = Low Priority

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Table 22-1: CAN Controller Register Map

Address	Register Name	Address	Register Name	Address	Register Name	Address	Register Name
F7Fh	—	F5Fh	—	F3Fh	—	F1Fh	RXM1EIDL
F7Eh	—	F5Eh	CANSTATRO1	F3Eh	CANSTATRO3	F1Eh	RXM1EIDH
F7Dh	—	F5Dh	RXB1D7	F3Dh	TXB1D7	F1Dh	RXM1SIDL
F7Ch	—	F5Ch	RXB1D6	F3Ch	TXB1D6	F1Ch	RXM1SIDH
F7Bh	—	F5Bh	RXB1D5	F3Bh	TXB1D5	F1Bh	RXM0EIDL
F7Ah	—	F5Ah	RXB1D4	F3Ah	TXB1D4	F1Ah	RXM0EIDH
F79h	—	F59h	RXB1D3	F39h	TXB1D3	F19h	RXM0SIDL
F78h	—	F58h	RXB1D2	F38h	TXB1D2	F18h	RXM0SIDH
F77h	—	F57h	RXB1D1	F37h	TXB1D1	F17h	RXF5EIDL
F76h	TXERRCNT	F56h	RXB1D0	F36h	TXB1D0	F16h	RXF5EIDH
F75h	RXERRCNT	F55h	RXB1DLC	F35h	TXB1DLC	F15h	RXF5SIDL
F74h	COMSTAT	F54h	RXB1EIDH	F34h	TXB1EIDH	F14h	RXF5SIDH
F73h	CIOCON	F53h	RXB1EIDL	F33h	TXB1EIDL	F13h	RXF4EIDL
F72h	BRGCON3	F52h	RXB1SIDL	F32h	TXB1SIDL	F12h	RXF4EIDH
F71h	BRGCON2	F51h	RXB1SIDH	F31h	TXB1SIDH	F11h	RXF4SIDL
F70h	BRGCON1	F50h	RXB1CON	F30h	TXB1CON	F10h	RXF4SIDH
F6Fh	CANCON	F4Fh	—	F2Fh	—	F0Fh	RXF3EIDL
F6Eh	CANSTAT	F4Eh	CANSTATRO2	F2Eh	CANSTATRO4	F0Eh	RXF3EIDH
F6Dh	RXB0D7	F4Dh	TXB0D7	F2Dh	TXB2D7	F0Dh	RXF3SIDL
F6Ch	RXB0D6	F4Ch	TXB0D6	F2Ch	TXB2D6	F0Ch	RXF3SIDH
F6Bh	RXB0D5	F4Bh	TXB0D5	F2Bh	TXB2D5	F0Bh	RXF2EIDL
F6Ah	RXB0D4	F4Ah	TXB0D4	F2Ah	TXB2D4	F0Ah	RXF2EIDH
F69h	RXB0D3	F49h	TXB0D3	F29h	TXB2D3	F09h	RXF2SIDL
F68h	RXB0D2	F48h	TXB0D2	F28h	TXB2D2	F08h	RXF2SIDH
F67h	RXB0D1	F47h	TXB0D1	F27h	TXB2D1	F07h	RXF1EIDL
F66h	RXB0D0	F46h	TXB0D0	F26h	TXB2D0	F06h	RXF1EIDH
F65h	RXB0DLC	F45h	TXB0DLC	F25h	TXB2DLC	F05h	RXF1SIDL
F64h	RXB0EIDL	F44h	TXB0EIDL	F24h	TXB2EIDL	F04h	RXF1SIDH
F63h	RXB0EIDH	F43h	TXB0EIDH	F23h	TXB2EIDH	F03h	RXF0EIDH
F62h	RXB0SIDL	F42h	TXB0SIDL	F22h	TXB2SIDL	F02h	RXF0EIDL
F61h	RXB0SIDH	F41h	TXB0SIDH	F21h	TXB2SIDH	F01h	RXF0SIDL
F60h	RXB0CON	F40h	TXB0CON	F20h	TXB2CON	F00h	RXF0SIDH

Note: The shaded addresses indicate the registers that are in the access RAM.

22.3 CAN Overview

The Controller Area Network (CAN) is a serial communications protocol which efficiently supports distributed real-time control with a very high level of robustness. The Protocol is fully defined by Robert Bosch GmbH, in the CAN Specification V2.0B from 1991.

Its domain of application ranges from high speed networks to low cost multiplex wiring. In automotive electronics; (engine control units, sensors, anti-skid-systems, etc.) are connected using CAN with bit rates up to 1 Mbit/sec. The CAN Network allows a cost effective replacement of the wiring harnesses in the automobile. The robustness of the bus in noisy environments and the ability to detect and recover from fault conditions makes the bus suitable for industrial control applications such as DeviceNet, SDS and other fieldbus protocols.

CAN is an asynchronous serial bus system with one logical bus line. It has an open, linear bus structure with equal bus nodes. A CAN bus consists of two or more nodes. The number of nodes on the bus may be changed dynamically without disturbing the communication of other nodes. This allows easy connection and disconnection of bus nodes (e.g. for addition of system function, error recovery or bus monitoring).

The bus logic corresponds to a "wired-AND" mechanism, "Recessive" bits (mostly, but not necessarily equivalent to the logic level “1”) are overwritten by "Dominant" bits (mostly logic level "0"). As long as no bus node is sending a dominant bit, the bus line is in the recessive state, but a dominant bit from any bus node generates the dominant bus state. Therefore, for the CAN bus line, a medium must be chosen that is able to transmit the two possible bit states (dominant and recessive). One of the most common and cheapest ways is to use a twisted wire pair. The bus lines are then called "CANH" and "CANL", and may be connected directly to the nodes or via a connector. There's no standard defined by CAN regarding the connector to be used. The twisted wire pair is terminated by terminating resistors at each end of the bus line. The maximum bus speed is 1 Mbit, which can be achieved with a bus length of up to 40 meters. For bus lengths longer than 40 meters the bus speed must be reduced (a 1000 m bus can be realized with a 40 Kbit bus speed). For a bus length above 1000 meters special drivers should be used. At least 20 nodes may be connected without additional equipment. Due to the differential nature of transmission, CAN is insensitive to EMI because both bus lines are affected in the same way which leaves the differential signal unaffected. The bus lines can also be shielded to reduce the electromagnetic emission of the bus itself, especially at high baud rates.

The binary data is coded corresponding to the NRZ code (Non-Return-to-Zero; low level = dominant state; high level = recessive state). To ensure clock synchronization of all bus nodes, bit-stuffing is used. This means that during the transmission of a message a maximum of five consecutive bits may have the same polarity. Whenever five consecutive bits of the same polarity have been transmitted, the transmitter will insert one additional bit of the opposite polarity into the bit stream before transmitting further bits. The receiver also checks the number of bits with the same polarity and removes the stuff bits from the bit stream (destuffing).

In the CAN protocol it is not bus nodes that are addressed, but the address information is contained in the messages that are transmitted. This is done via an identifier (part of each message) which identifies the message content (e.g. engine speed, oil temperature etc.,). The identifier additionally indicates the priority of the message. The lower the binary value of the identifier, the higher the priority of the message.

For bus arbitration, Carrier Sense Multiple Access/Collision Detection (CSMA/CD) with Non-Destructive Arbitration (NDA) is used. If bus node A wants to transmit a message across the network, it first checks that the bus is in the idle state ("Carrier Sense") i.e., no node is currently transmitting. If this is the case (and no other node wishes to start a transmission at the same moment) node A becomes the bus master and sends its message. All other nodes switch to receive mode during the first transmitted bit (Start Of Frame bit). After correct reception of the message (which is acknowledged by each node) each bus node checks the message identifier and stores the message, if required. Otherwise, the message is discarded.

If two or more bus nodes start their transmission at the same time ("Multiple Access"), collision of the messages is avoided by bitwise arbitration ("Collision Detection/Non-Destructive Arbitration" together with the "Wired-AND" mechanism, "dominant" bits override "recessive" bits). Each node sends the bits of its message identifier (MSb first) and monitors the bus level. A node that sends a recessive identifier bit but reads back a dominant one loses bus arbitration and switches to receive mode. This condition occurs when the message identifier of a competing node has a

lower binary value (dominant state = logic 0) and therefore, the competing node is sending a message with a higher priority. In this way, the bus node with the highest priority message wins arbitration without losing time by having to repeat the message. All other nodes automatically try to repeat their transmission once the bus returns to the idle state. It is not permitted for different nodes to send messages with the same identifier as arbitration could fail leading to collisions and errors later in the message.

The original CAN specifications (Versions 1.0, 1.2 and 2.0A) defined the message identifier as having a length of 11 bits giving a possible 2048 message identifiers. The specification has since been updated (to version 2.0B) to remove this limitation. CAN specification Version 2.0B allows message identifier lengths of 11 and/or 29 bits to be used (an identifier length of 29 bits allows over 536 Million message identifiers). Version 2.0B CAN is also referred to as "Extended CAN"; and Versions 1.0, 1.2 and 2.0A) are referred to as "Standard CAN".

22.3.1 Standard CAN vs. Extended CAN

Those Data Frames and Remote Frames, which only contain the 11 bit identifier, are called Standard Frames according to CAN specification V2.0A. With these frames, 2048 different messages can be identified (identifiers 0-2047). However, the 16 messages with the lowest priority (2032-2047) are reserved. Extended Frames according to CAN specification V2.0B have a 29 bit identifier. As already mentioned, this 29 bit identifier is made up of the 11 bit identifier ("Standard ID") and the 18 bit Extended identifier ("Extended ID").

CAN modules specified by CAN V2.0A are only able to transmit and receive Standard Frames according to the Standard CAN protocol. Messages using the 29 bit identifier cause errors. If a device is specified by CAN V2.0B, there is one more distinction. Modules named "Part B Passive" can only transmit and receive Standard Frames but tolerate Extended Frames without generating Error Frames. "Part B Active" devices are able to transmit and receive both Standard and Extended Frames.

22.3.2 Basic CAN vs. Full CAN

There is one more CAN characteristic concerning the interface between the CAN module and the host CPU, dividing CAN chips into "Basic CAN" and "Full CAN" devices. This distinction is not related to Standard vs. Extended CAN, which makes it possible to use both Basic and Full CAN devices in the same network.

In the Basic CAN devices, only basic functions of the protocol are implemented in hardware, e.g. the generation and the check of the bit stream. The decision, if a received message has to be stored or not (acceptance filtering) and the whole message management has to be done by software, i.e., by the host CPU. In addition, the CAN chip typically provides only one transmit buffer and one or two receive buffers. So the host CPU load is quite high using Basic CAN modules, and these devices can only be used at low baud rates and low bus loads with only a few different messages. The advantages of Basic CAN are the small chip size leading to low costs of these devices.

Full CAN devices implement the whole bus protocol in hardware including the acceptance filtering and the message management. They contain several so called message objects which handle the identifier, the data, the direction (receive or transmit) and the information Standard CAN/Extended CAN. During the initialization of the device, the host CPU defines which messages are to be sent and which are to be received. The host CPU is informed by interrupt if the identifier of a received message matches with one of the programmed (receive-) message objects. In this way. the CPU load is reduced. Using Full CAN devices, high baud rates and high bus loads with many messages can be handled. These chips are more expensive than the Basic CAN devices, though.

Many Full CAN chips provide a "Basic-CAN Feature". One of the messages objects can be programmed in so that every message is stored there that does not match with one of the other message objects. This can be very helpful in a number of applications.

22.3.3 ISO Model

The ISO/OSI Reference Model is used to define the layers of protocol of a communication system as shown in Figure 22-2. At the highest end, the applications need to communicate between each other. At the lowest end, some physical medium is used to provide electrical signaling.

The higher levels of the protocol are run by software. Within the CAN bus specification, there is no definition of the type of message or the contents or meaning of the messages transferred. These definitions are made in systems such as Volcano, the Volvo automotive CAN specification; J1939, the U.S. heavy truck multiplex wiring specification; and Allen-Bradly DeviceNet and Honeywell SDS, examples of industrial protocols.

The CAN bus module definition encompasses two levels of the overall protocol.

- The Data Link Layer
  - The Logical Link Control (LLC) sub layer
  - The Medium Access Control (MAC) sub layer
- The Physical Layer
  - The Physical Signaling (PLS) sub layer

The LLC sub layer is concerned with Message Filtering, Overload Notification and Error Recovery Management. The scope of the LLC sub layer is:

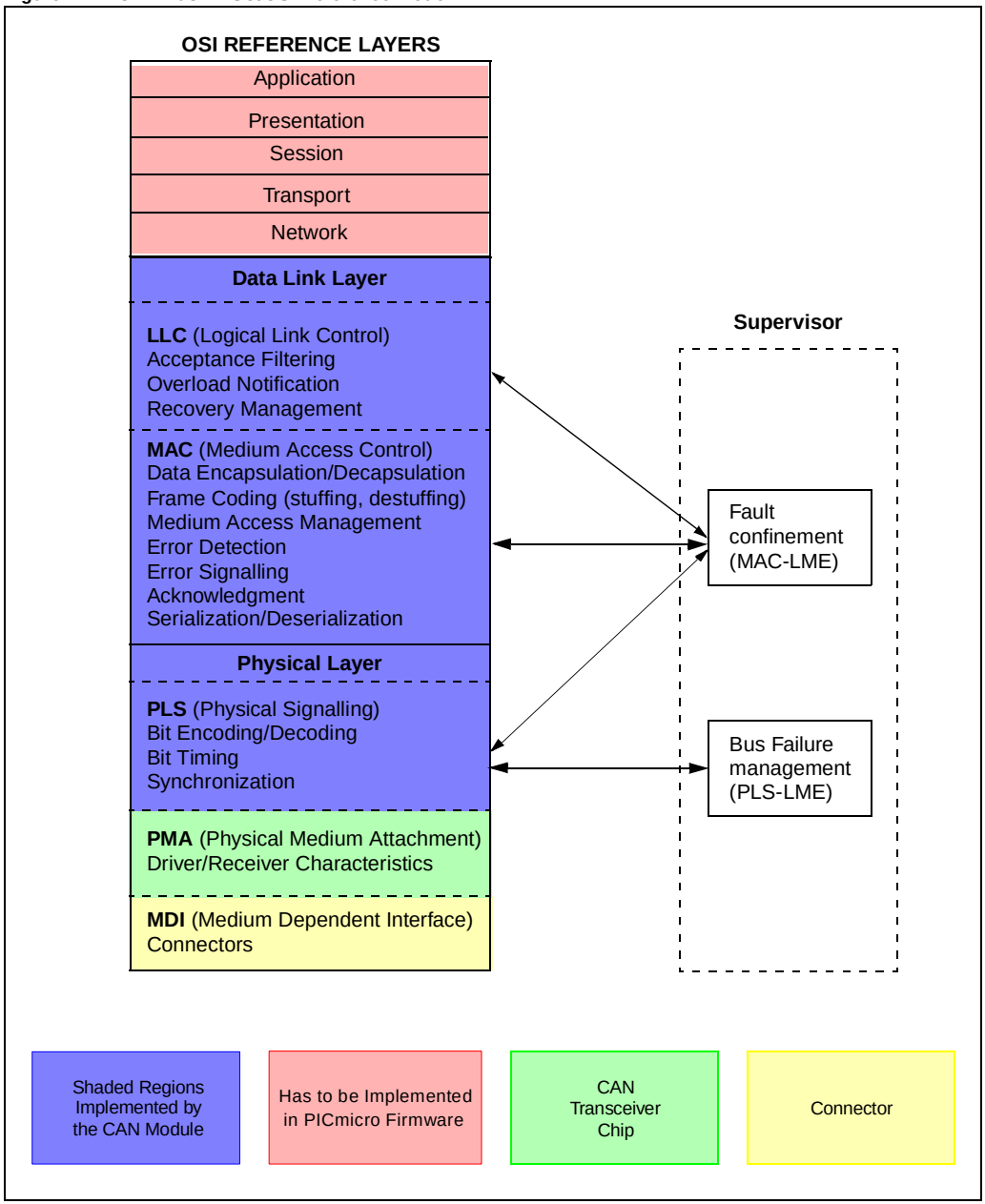
- To provide services for data transfer and for remote data request,
- To decide which messages received by the LLC sub layer are actually to be accepted,
- To provide means for error recovery management and overload notifications.

The MAC sub layer represents the kernel of the CAN protocol. The MAC sub layer defines the transfer protocol, i.e., controlling the Framing, Performing Arbitration, Error Checking, Error Signalling and Fault Confinement. It presents messages received from the LLC sub layer and accepts messages to be transmitted to the LLC sub layer. Within the MAC sub layer it is decided whether the bus is free for starting a new transmission or whether a reception is just starting. The MAC sub layer is supervised by a management entity called Fault Confinement which is self-checking mechanism for distinguishing short disturbances from permanent failures. Also, some general features of the bit timing are regarded as part of the MAC sub layer.

The physical layer defines the actual transfer of the bits between the different nodes with respect to all electrical properties. The PLS sub layer defines how signals are actually transmitted and therefore deals with the description of Bit Timing, Bit Encoding, and Synchronization.

The lower levels of the protocol are implemented in driver/receiver chips and the actual interface such as twisted pair wiring or optical fiber etc. Within one network, the physical layer has to be the same for all nodes. The Driver/Receiver Characteristics of the Physical Layer are not defined so as to allow transmission medium and signal level implementations to be optimized for their application. The most common example is defined in ISO11898 Road Vehicles multiplex wiring specification.

Figure 22-2: CAN Bus in ISO/OSI Reference Model



22.4 CAN Bus Features

The CAN module is a communication controller implementing the CAN 2.0A/B protocol as defined in the BOSCH specification. The module will support CAN 1.2, CAN 2.0A, CAN 2.0B Passive, and CAN 2.0B Active versions of the protocol. The module implementation is a Full CAN system.

The module features are as follows:

- Implementation of the CAN protocol CAN 1.2, CAN 2.0A and CAN 2.0B
- Standard and extended data frames
- Data length from 0 - 8 bytes
- Programmable bit rate up to 1 Mbit/sec
- Support for remote frames
- Double buffered receiver with two prioritized received message storage buffers
- 6 full (standard/extended identifier) acceptance filters, 2 associated with the high priority receive buffer, and 4 associated with the low priority receive buffer
- 2 full acceptance filter masks, one each associated with the high and low priority receive buffers
- Three transmit buffers with application specified prioritization and abort capability
- Programmable wake-up functionality with integrated low-pass filter
- Programmable loop-back mode and programmable state clocking supports self-test operation
- Signaling via interrupt capabilities for all CAN receiver and transmitter error states
- Programmable clock source
- Programmable link to timer module for time-stamping and network synchronization
- Low power SLEEP mode

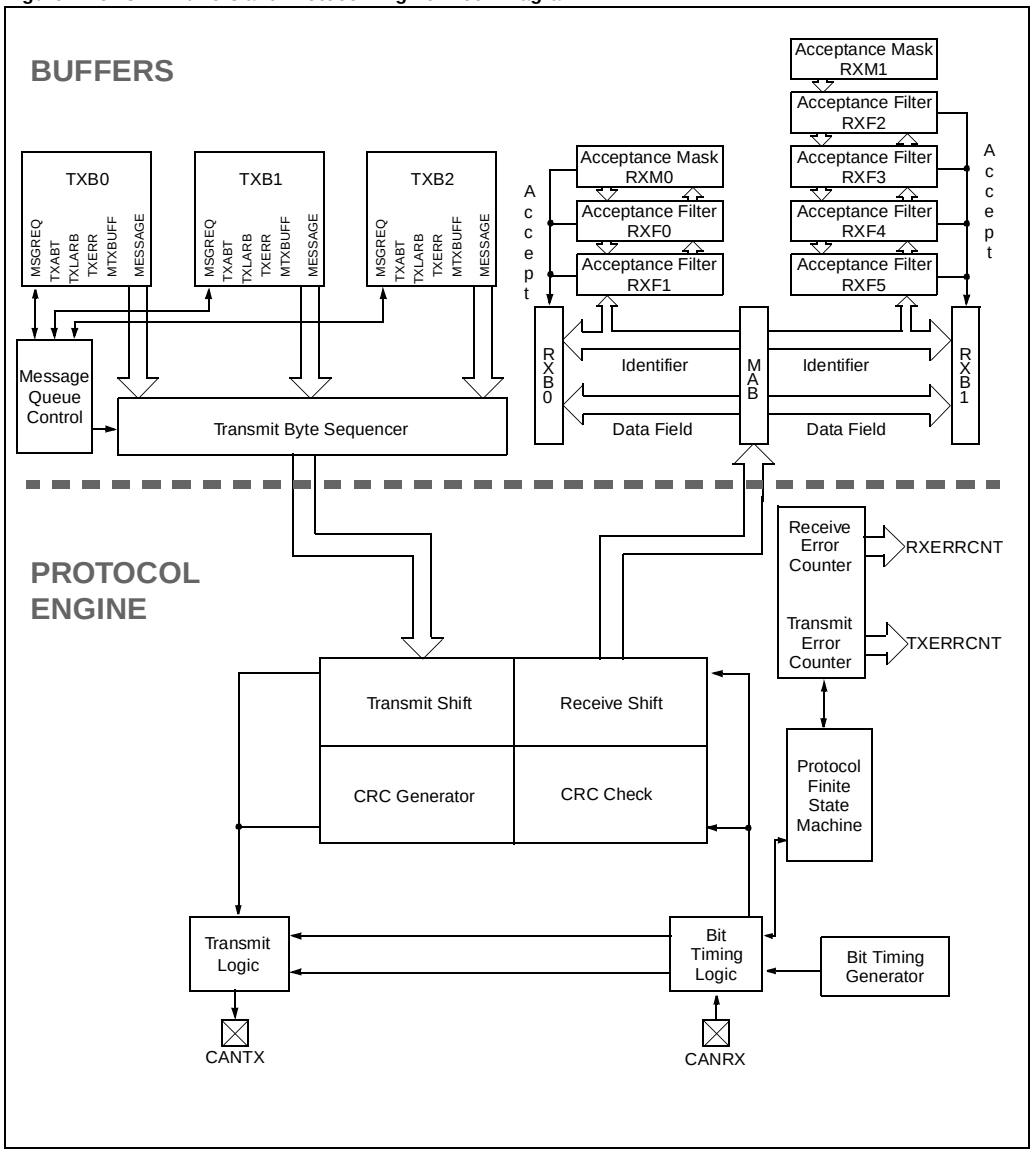
22.5 CAN Module Implementation

This subsection will discuss the implementation of the CAN module and the supported frame formats.

22.5.1 Overview of the Module

The CAN bus module consists of a Protocol Engine and message buffering and control. The Protocol Engine can best be understood by defining the types of data frames to be transmitted and received by the module. These blocks are shown in Figure 22-3.

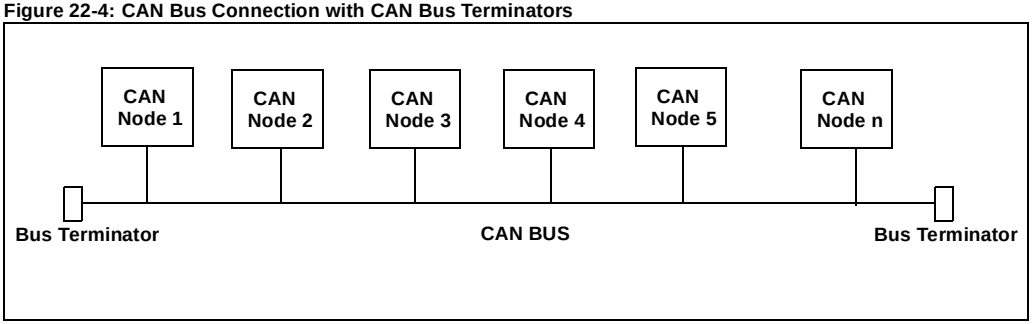
Figure 22-3: CAN Buffers and Protocol Engine Block Diagram



22.5.1.1 Typical Connection

Figure 22-4 shows a typical connection between multiple CAN nodes with CAN bus terminators.

Figure 22-4: CAN Bus Connection with CAN Bus Terminators



22.5.2 CAN Protocol Engine

The CAN protocol engine combines several functional blocks, shown in Figure 22-5. These units and the functions they provide are described below.

The heart of the engine is the Protocol Finite State Machine (FSM). This state machine sequences through the messages on a bit by bit basis, changing states of the machine as various fields of various frame types are transmitted or received. The framing messages in Section **22.6** show the states associated with each bit. The FSM is a sequencer controlling the sequential data stream between the TX/RX Shift Register, the CRC Register, and the bus line. The FSM also controls the Error Management Logic (EML) and the parallel data stream between the TX/RX Shift Register and the buffers such that the processes of reception arbitration, transmission, and error signaling are performed according to the CAN protocol. Note that the automatic retransmission of messages on the bus line is handled by the FSM.

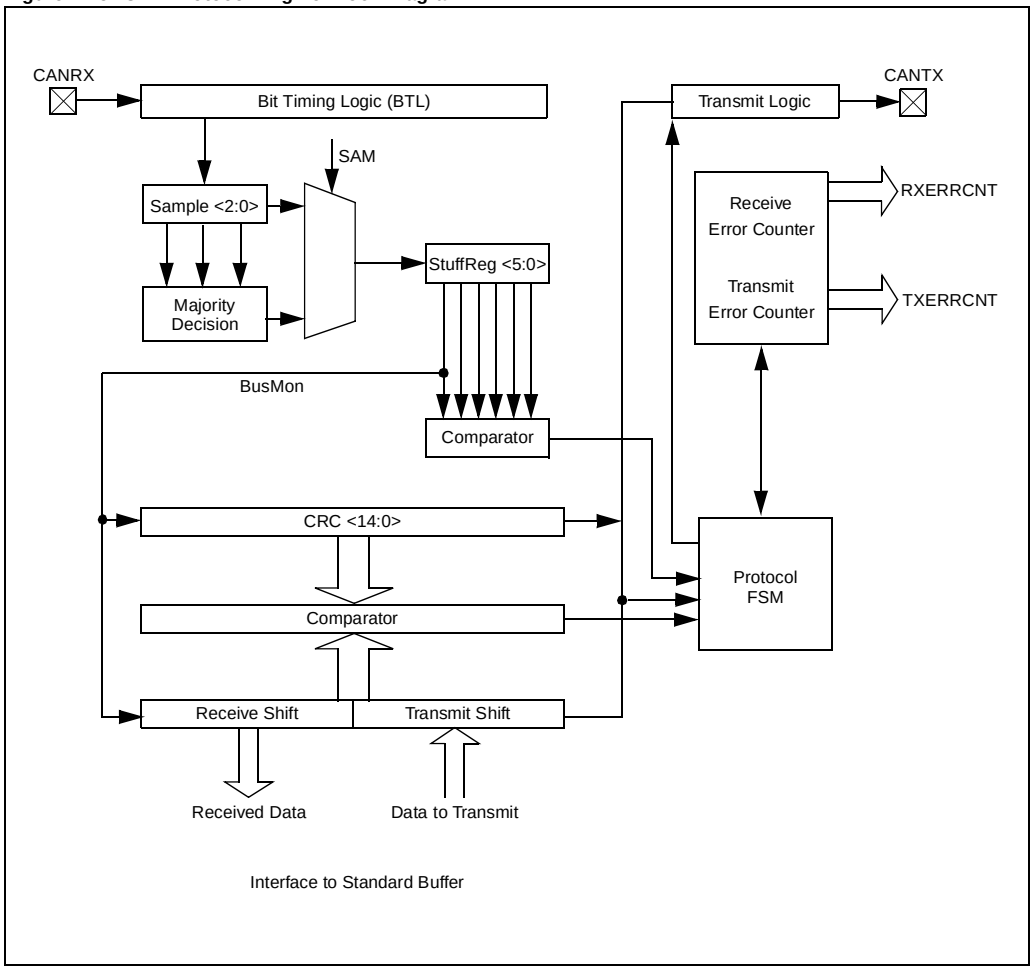
The data interface to the engine consists of byte wide transmit and receive data. Rather than assembling and shifting an entire frame, the frames are broken into bytes. A receive or transmit address from the Protocol FSM signifies which byte of the frame is current. For transmission, the appropriate byte from the transmit buffer is selected and presented to the engine, which then uses an 8 bit shift register to serialize the data. For reception, an 8 bit shift register assembles a byte which is then loaded into the appropriate byte in the message assembly buffer.

The Cyclic Redundancy Check Register generates the Cyclic Redundancy Check (CRC) code to be transmitted over the data bytes and checks the CRC code of incoming messages.

The Error Management Logic (EML) is responsible for the fault confinement of the CAN device. Its counters, the Receive Error Counter and the Transmit Error Counter, are incremented and decremented by commands from the Bit Stream Processor. According to the values of the error counters, the CAN controller is set into the states error active, error passive or bus off.

The Bit Timing Logic (BTL) monitors the bus line input and handles the bus line related bit timing according to the CAN protocol. The BTL synchronizes on a recessive to dominant busline transition at Start of Frame (hard synchronization) and on any further recessive to dominant bus line transition, if the CAN controller itself does not transmit a dominant bit (resynchronization). The BTL also provides programmable time segments to compensate for the propagation delay time and for phase shifts and in defining the position of the Sample Point in the bit time. The programming of the BTL depends on the baud rate and on external physical delay times.

Figure 22-5: CAN Protocol Engine Block Diagram



22.5.3 CAN Module Functionality

The CAN protocol engine handles all functions for receiving and transmitting messages on the CAN bus. Messages are transmitted by first loading the appropriate data registers. Status and errors can be checked by reading the appropriate registers. Any message detected on the CAN bus is checked for errors and then matched against filters to see if it should be received and stored in one of the 2 receive registers.

The CAN Module supports the following Frame types:

- Standard Data Frame
- Extended Data Frame
- Remote Frame
- Error Frame
- Overload Frame
- Interframe Space

Section **22.6** describes the Frames and their formats.

22.6 Frame Types

This chapter describes the CAN Frame types supported by the CAN module.

22.6.1 Standard Data Frame

A Standard Data Frame is generated by a node when the node wishes to transmit data. The Standard CAN Data Frame is shown in Figure 22-6. In common with all other frames, the frame begins with a Start Of Frame bit (SOF - dominant state) for hard synchronization of all nodes.

The SOF is followed by the Arbitration Field consisting of 12 bits, the 11 bit Identifier (reflecting the contents and priority of the message) and the RTR bit (Remote Transmission Request bit). The RTR bit is used to distinguish a data Frame (RTR - dominant) from a Remote Frame.

The next field is the Control Field, consisting of 6 bits. The first bit of this field is called the Identifier Extension (IDE) bit and is at dominant state to specify that the frame is a Standard Frame. The following bit is reserved, RB0, and defined as a dominant bit. The remaining 4 bits of the Control Field are the Data Length Code (DLC) and specify the number of bytes of data contained in the message.

The data being sent follows in the Data Field which is of the length defined by the DLC above (0 - 8 bytes).

The Cyclic Redundancy Field (CRC) follows and is used to detect possible transmission errors. The CRC Field consists of a 15 bit CRC sequence, completed by the End of Frame (EOF) field, which consists of seven recessive bits (no bit-stuffing).

The final field is the Acknowledge Field. During the ACK Slot bit the transmitting node sends out a recessive bit. Any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit (regardless of whether the node is configured to accept that specific message or not). The recessive Acknowledge Delimiter completes the Acknowledge Slot and may not be overwritten by a dominant bit.

22.6.1.1 Extended Data Frame

In the Extended CAN Data Frame, shown in Figure 22-7, the Start of Frame bit (SOF) is followed by the Arbitration Field consisting of 38 bits. The first 11 bits are the 11 most significant bits of the 29 bit identifier ("Base-ID"). These 11 bits are followed by the Substitute Remote Request bit (SRR), which is transmitted as recessive. The SRR is followed by the IDE bit which is recessive to denote that the frame is an Extended CAN frame. It should be noted from this, that if arbitration remains unresolved after transmission of the first 11 bits of the identifier, and one of the nodes involved in arbitration is sending a Standard CAN frame (11 bit identifier), then the Standard CAN frame will win arbitration due to the assertion of a dominant IDE bit. Also, the SRR bit in an Extended CAN frame must be recessive to allow the assertion of a dominant RTR bit by a node that is sending a Standard CAN Remote Frame. The SRR and IDE bits are followed by the remaining 18 bits of the identifier ("ID-Extension") and the Remote Transmission Request bit.

To enable standard and extended frames to be sent across a shared network, it is necessary to split the 29 bit extended message Identifier into 11 bit (most significant) and 18 bit (least significant) sections. This split ensures that the Identifier Extension bit (IDE) can remain at the same bit position in both standard and extended frames.

The next field is the Control Field, consisting of 6 bits. The first 2 bits of this field are reserved and are at dominant state. The remaining 4 bits of the Control Field are the Data Length Code (DLC) and specify the number of data bytes.

The remaining portion of the frame (Data Field, CRC Field, Acknowledge Field, End Of Frame and Intermission) is constructed in the same way as for a Standard Data Frame.

22.6.1.2 Remote Frame

Normally data transmission is performed on an autonomous basis with the data source node (e.g. a sensor sending out a Data Frame). It is possible, however, for a destination node to request the data from the source. For this purpose, the destination node sends a "Remote Frame" with an identifier that matches the identifier of the required Data Frame. The appropriate data source node will then send a Data Frame as a response to this Remote request.

There are 2 differences between a Remote Frame and a Data Frame, shown in Figure 22-8. First, the RTR bit is at the recessive state and second there is no Data Field. In the very unlikely event of a Data Frame and a Remote Frame with the same identifier being transmitted at the same time, the Data Frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the Remote Frame receives the desired data immediately.

22.6.1.3 The Error Frame

An Error Frame is generated by any node that detects a bus error. An error frame, shown in Figure 22-9, consists of 2 fields, an Error Flag field followed by an Error Delimiter field. The Error Delimiter consists of 8 recessive bits and allows the bus nodes to restart bus communications cleanly after an error. There are two forms of Error Flag fields. The form of the Error Flag field depends on the error status of the node that detects the error.

If an error-active node detects a bus error then the node interrupts transmission of the current message by generating an active error flag. The active error flag is composed of six consecutive dominant bits. This bit sequence actively violates the bit-stuffing rule. All other stations recognize the resulting bit-stuffing error and in turn generate Error Frames themselves, called Error Echo Flags. The Error Flag field therefore consists of between six and twelve consecutive dominant bits (generated by one or more nodes). The Error Delimiter field completes the Error Frame. After completion of the Error Frame, bus activity retains to normal and the interrupted node attempts to resend the aborted message.

If an error passive node detects a bus error then the node transmits an error passive flag followed, again, by the Error Delimiter field. The error passive flag consists of six consecutive recessive bits. From this it follows that, unless the bus error is detected by the bus master node or other error active receiver, that is actually transmitting, the transmission of an Error Frame by an error passive node will not affect any other node on the network. If the bus master node generates an error passive flag then this may cause other nodes to generate error frames due to the resulting bit-stuffing violation. After transmission of an Error Frame, an error passive node must wait for 6 consecutive recessive bits on the bus before attempting to rejoin bus communications.

22.6.1.4 The Overload Frame

An Overload Frame, shown in Figure 22-10, has the same format as an Active Error Frame. An Overload Frame, however can only be generated during Interframe Space. This way, an Overload Frame can be differentiated from an Error Frame (an Error Frame is sent during the transmission of a message). The Overload Frame consists of 2 fields, an Overload Flag followed by an Overload Delimiter. The Overload Flag consists of six dominant bits followed by Overload Flags generated by other nodes (as for active error flag, again giving a maximum of twelve dominant bits). The Overload Delimiter consists of eight recessive bits. An Overload Frame can be generated by a node as a result of 2 conditions. First, the node detects a dominant bit during Interframe Space which is an illegal condition. Second, due to internal conditions, the node is not yet able to start reception of the next message. A node may generate a maximum of 2 sequential Overload Frames to delay the start of the next message.

22.6.1.5 The Interframe Space

Interframe Space separates a proceeding frame (of whatever type) from a following Data or Remote Frame. Interframe space is composed of at least 3 recessive bits, called the intermission. This is provided to allow nodes time for internal processing before the start of the next message frame. After the intermission, the bus line remains in the recessive state (Bus idle) until the next transmission starts.

Figure 22-6: Standard Data Frame

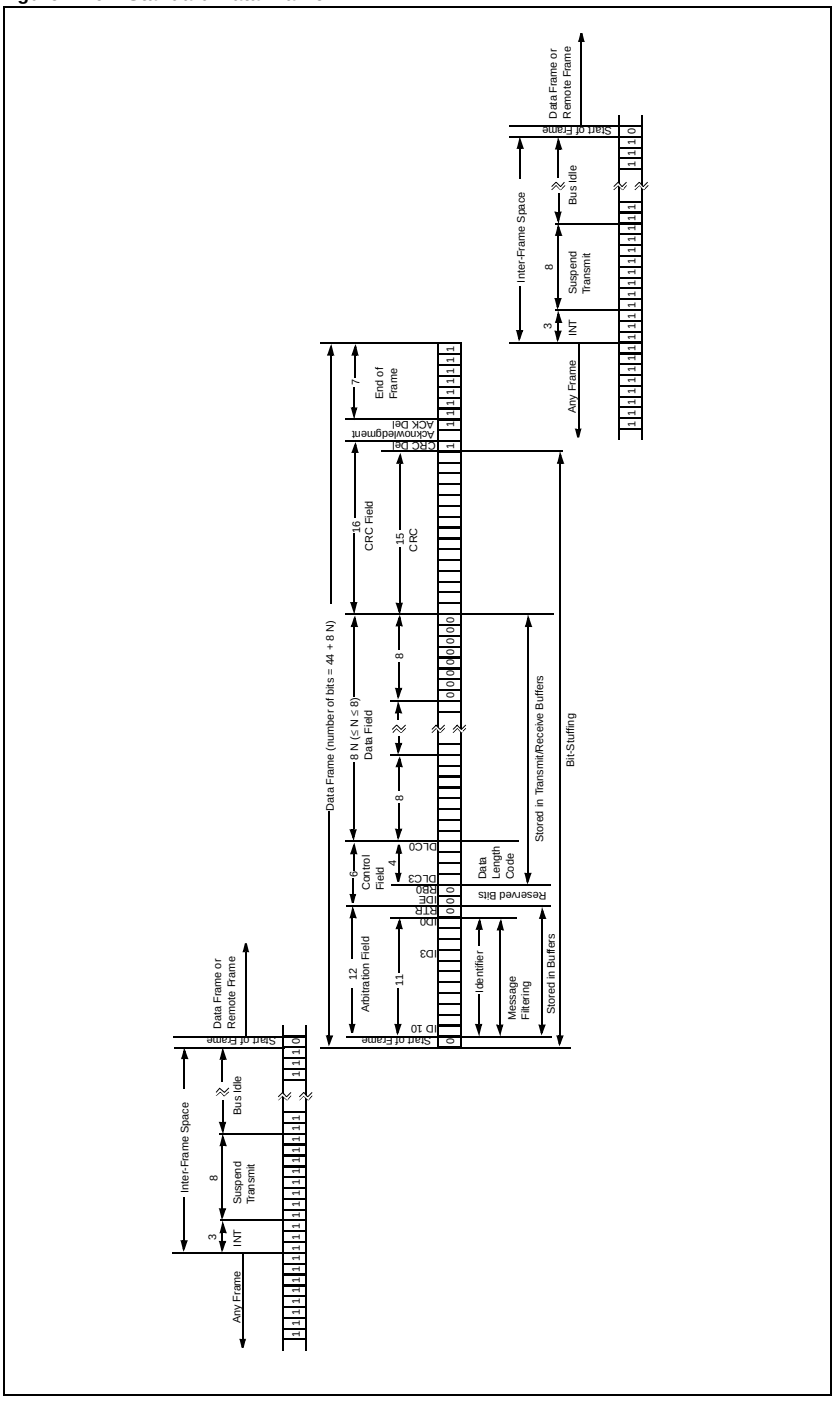


Figure 22-7: Extended Data Format

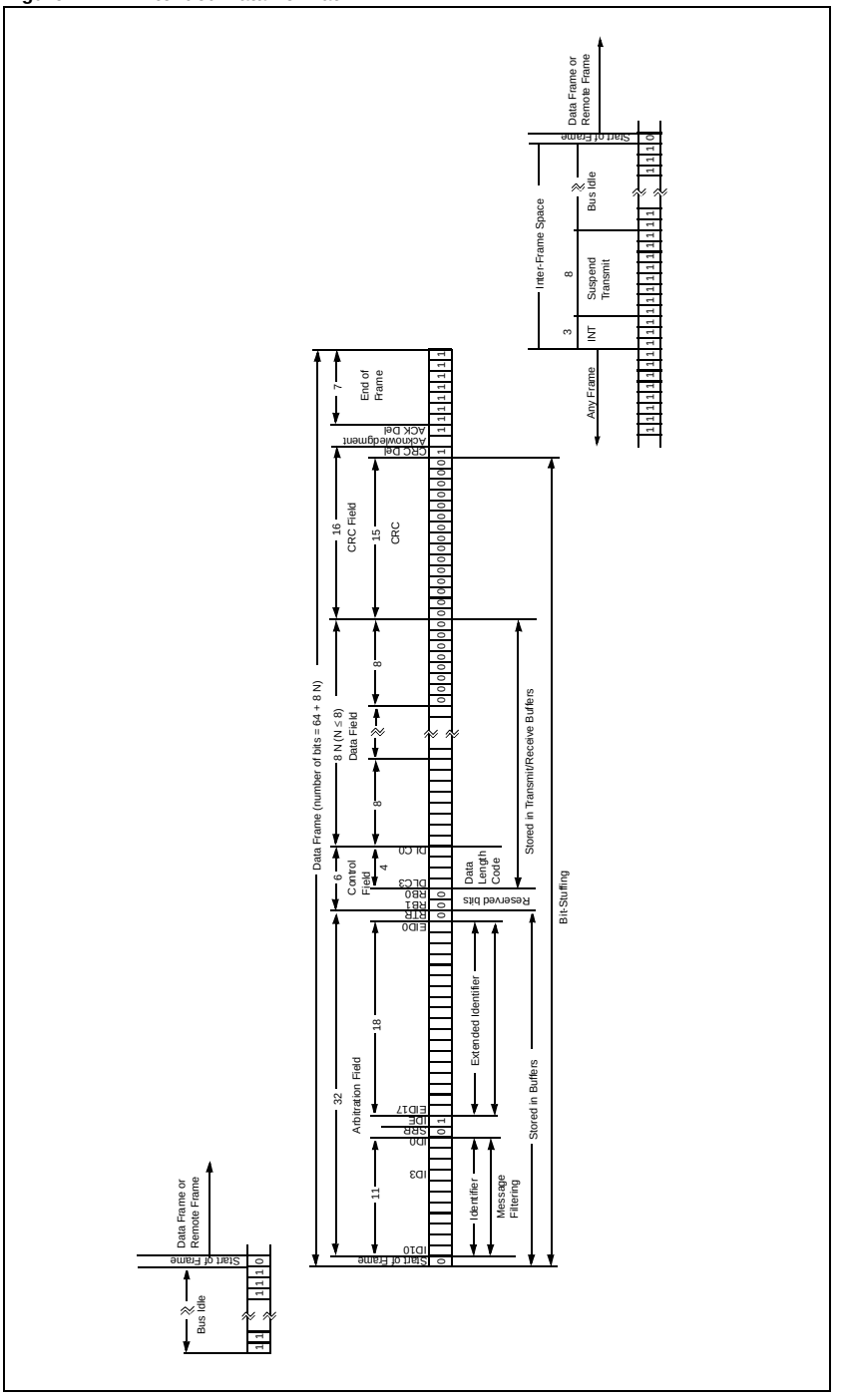


Figure 22-8: Remote Data Frame

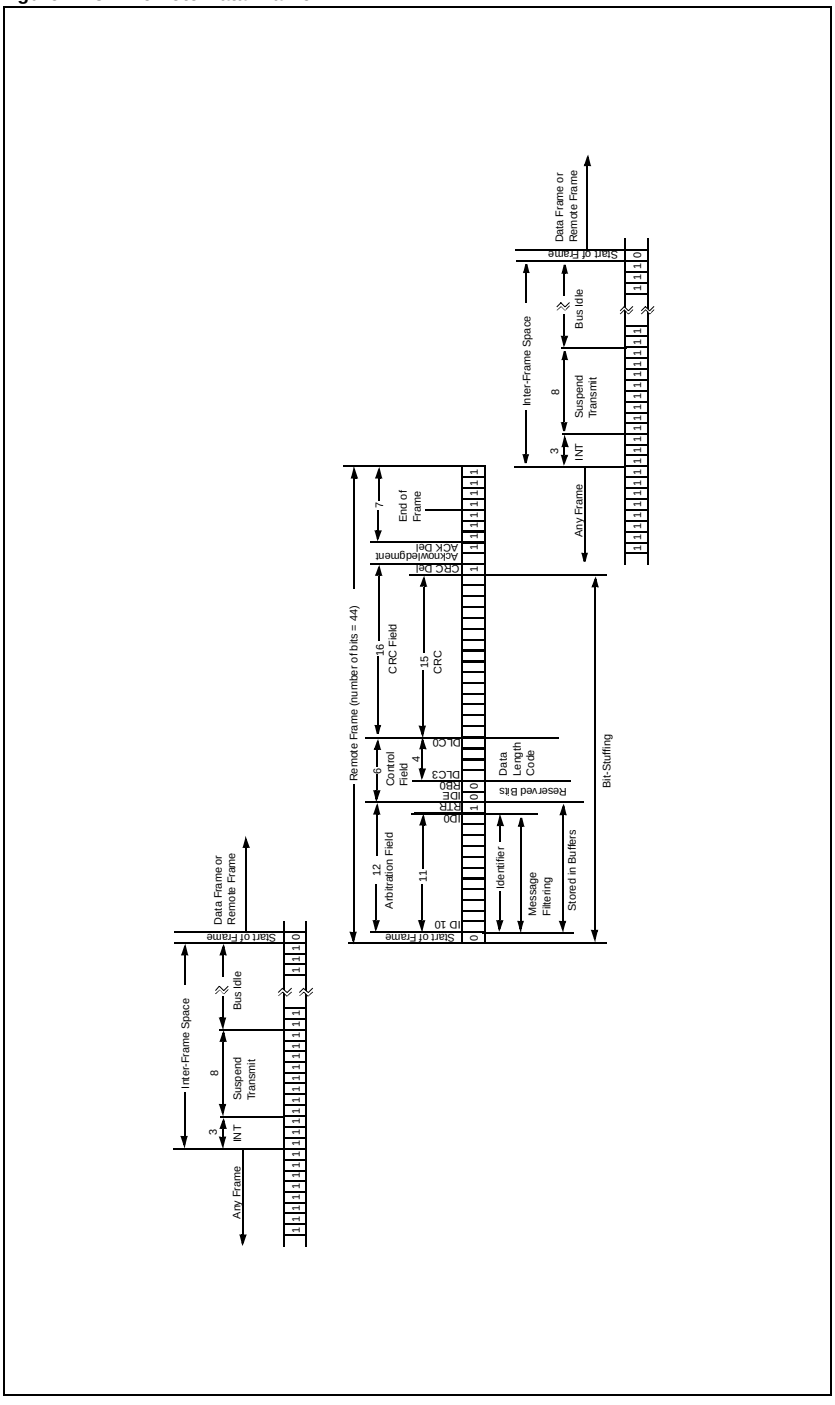


Figure 22-9: Error Frame

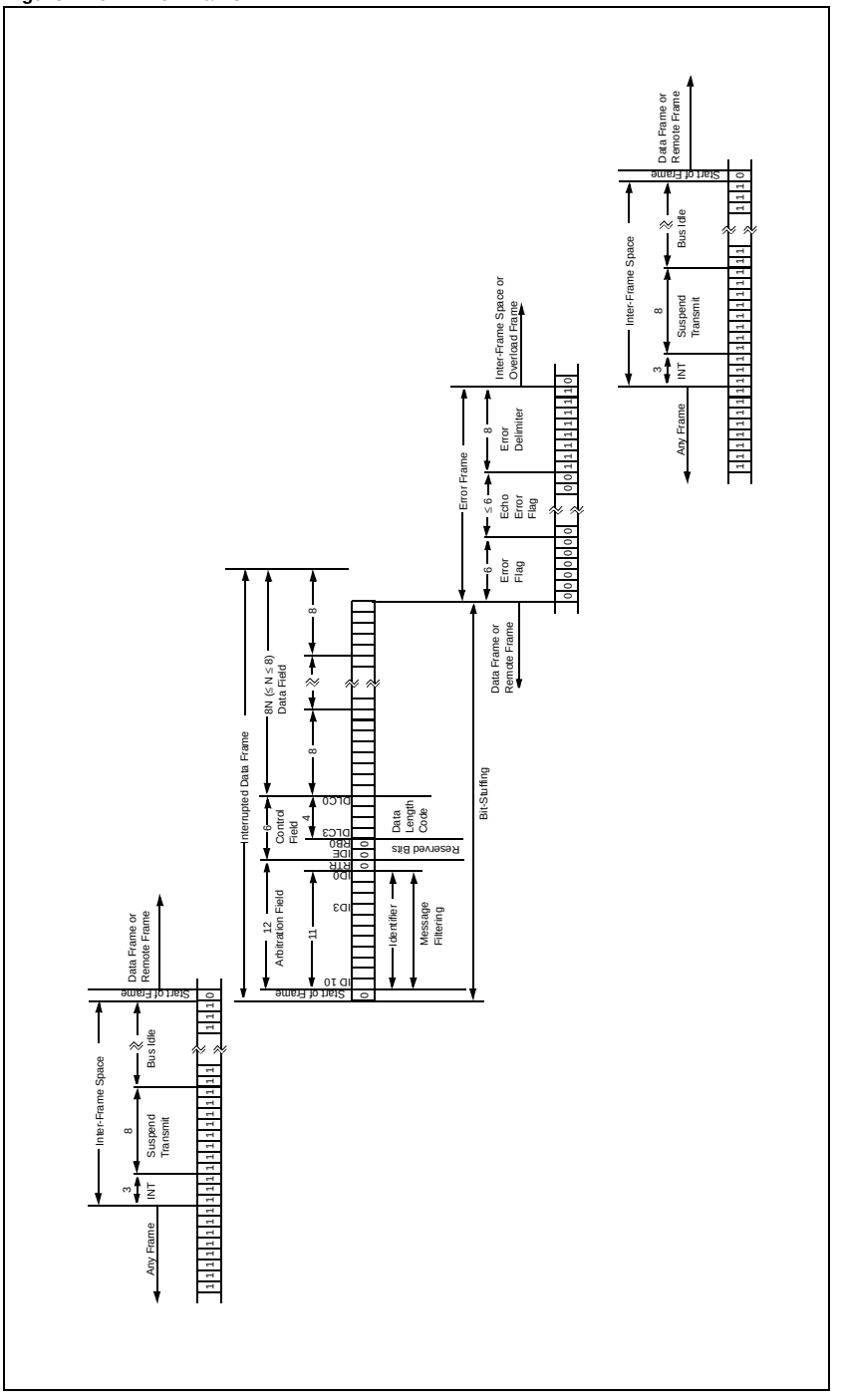
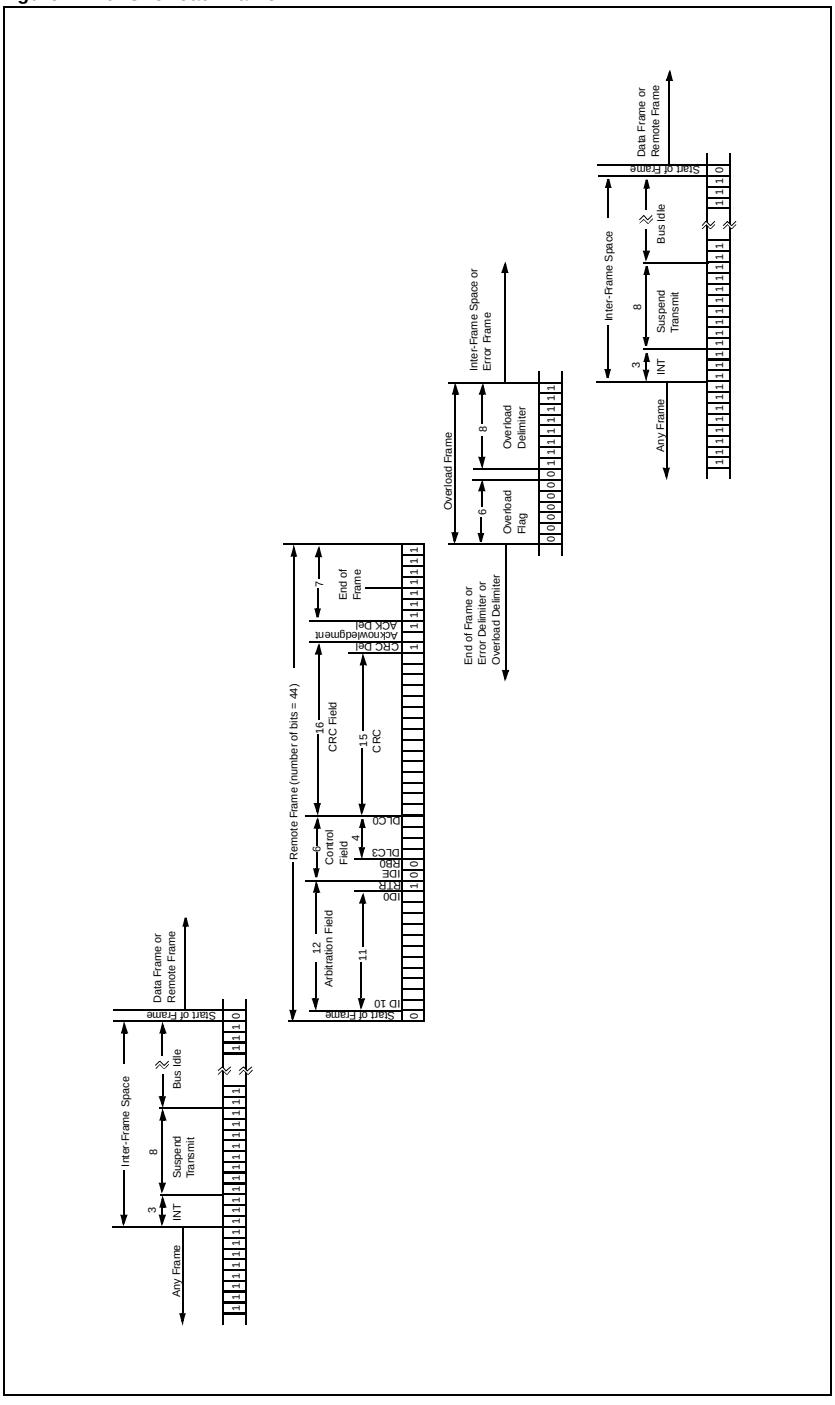


Figure 22-10: Overload Frame



22.7 Modes of Operation

The CAN Module can operate in one of several operation modes selected by the user. These modes include:

- Initialization Mode
- Disable Mode
- Normal Operation Mode
- Listen Only Mode
- Loop Back Mode
- Error Recognition Mode (selected through CANRXM bits)

Modes are requested by setting the REQOP2:REQOP0 bits except the Error Recognition Mode, which is requested through the CANRXM bits. Entry into a mode is acknowledged by monitoring the OPMODE bits. The module will not change the mode and the OPMODE2:OPMODE0 bits until a change in mode is acceptable, generally during bus idle time which is defined as at least 11 consecutive recessive bits.

22.7.1 Initialization Mode

In the initialization mode, the module will not transmit or receive. The error counters are cleared and the interrupt flags remain unchanged. The programmer will have access to configuration registers that are access restricted in other modes. The CAN bus configuration mode is explained in Section **22.8**.

22.7.2 Disable Mode

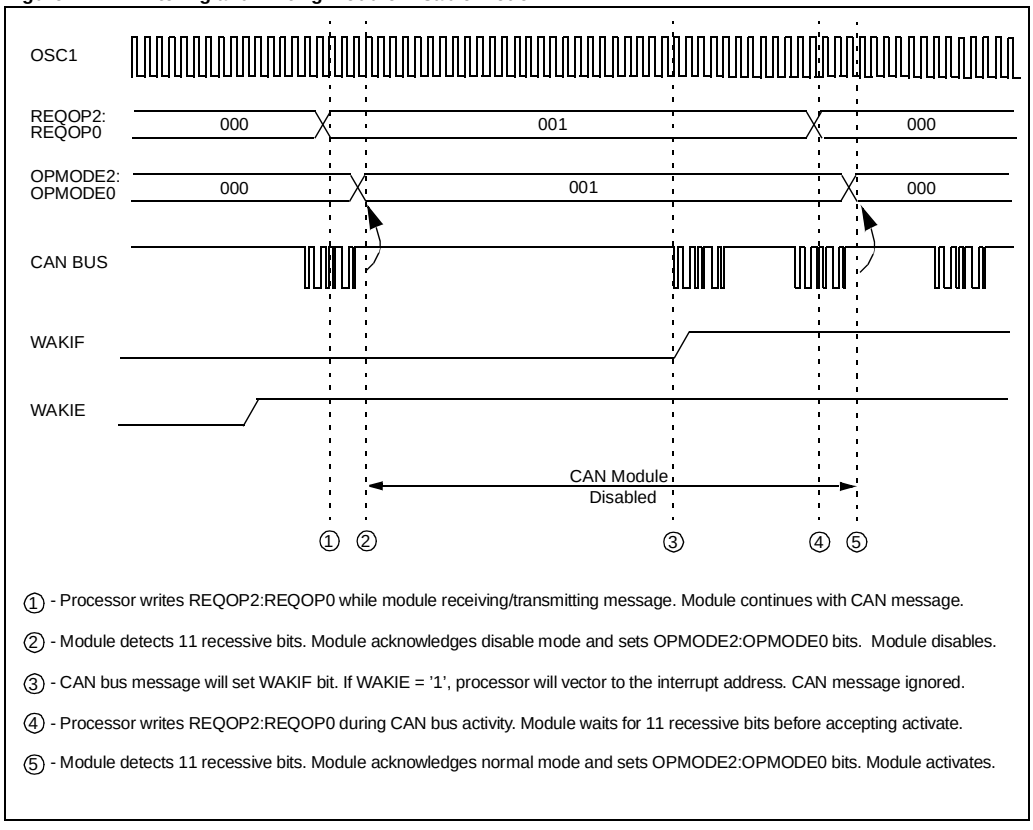
In Disable Mode, the module will not transmit or receive. The module has the ability to set the WAKIF bit due to bus activity, however any pending interrupts will remain and the error counters will retain their value.

If the REQOP2:REQOP0 bits = 001, the module will enter the module disable mode. This mode is similar to disabling other peripheral modules by turning off the module enables. This causes the module internal clock to stop unless the module is active (i.e., receiving or transmitting a message). If the module is active, the module will wait for 11 recessive bits on the CAN bus, detect that condition as an idle bus then accept the module disable command. When the OPMODE2:OPMODE0 bits = 001, that indicates whether the module successfully went into module disable mode (see Figure 22-11).

The WAKIF interrupt is the only module interrupt that is still active in the module disable mode. If the WAKIE is set, the processor will receive an interrupt whenever the CAN bus detects a dominant state, as occurs with a Start of Frame (SOF).

The I/O pins will revert to normal I/O function when the module is in the module disable mode.

Figure 22-11: Entering and Exiting Module Disable Mode



① - Processor writes REQOP2:REQOP0 while module receiving/transmitting message. Module continues with CAN message.

② - Module detects 11 recessive bits. Module acknowledges disable mode and sets OPMODE2:OPMODE0 bits. Module disables.

③ - CAN bus message will set WAKIF bit. If WAKIE = '1', processor will vector to the interrupt address. CAN message ignored.

④ - Processor writes REQOP2:REQOP0 during CAN bus activity. Module waits for 11 recessive bits before accepting activate.

⑤ - Module detects 11 recessive bits. Module acknowledges normal mode and sets OPMODE2:OPMODE0 bits. Module activates.

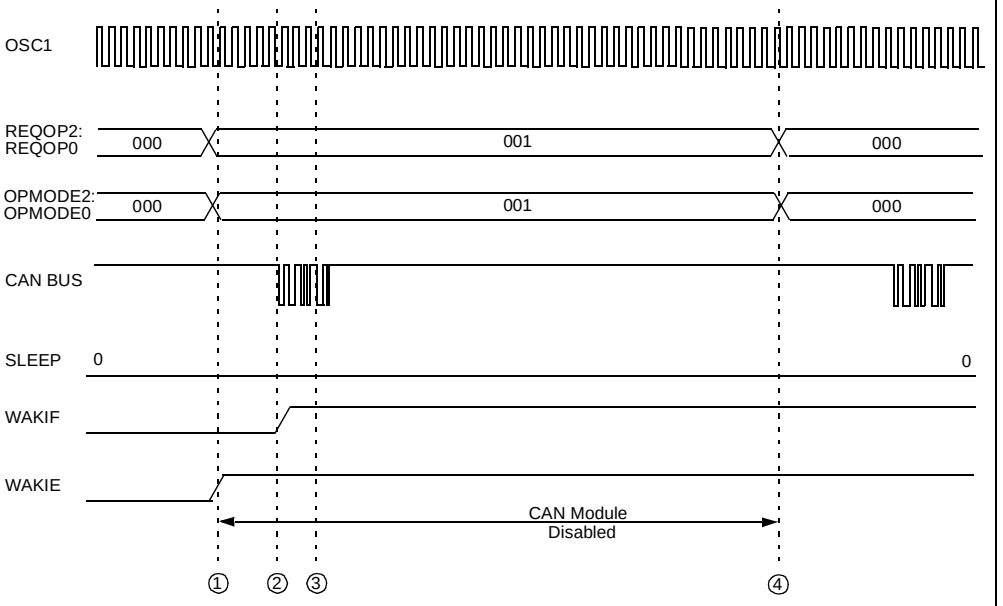
22.7.2.1 SLEEP Mode

A CPU SLEEP instruction will stop the crystal oscillator and shut down all system clocks. The user is responsible to take care that the module is not active when the CPU goes into SLEEP mode. The pins will revert into normal I/O function, dependent on the value in the TRIS register.

The recommended procedure is to bring the module into disabled mode before the CPU SLEEP instruction is executed.

Figure 22-12 illustrates the sequence of events when a CAN message is received during execution of the SLEEP instruction.

Figure 22-12:SLEEP Interrupted By Message



① - Processor requests and receives module disable mode. Wake up interrupt enabled.

② - CAN bus activity sets WAKIF flag. If GIE = '1' processor will vector to interrupt address, bypassing SLEEP instruction.

③ - Processor attempts to execute SLEEP instruction. Since WAKIF = '1', WAKIE = '1' and GIE = '0' processor will execute NOP in place of SLEEP instruction. CAN message ignored.

④ - Processor requests and receives module normal mode. CAN activity resumes.

22.7.2.2 Wake-up from SLEEP

Figure 22-13 depicts how the CAN module will execute the SLEEP instruction and how the module wakes up on bus activity. Upon a Wake-up from SLEEP the WAKIF flag is set.

The module will monitor the RX line for activity while the MCU is in SLEEP mode.

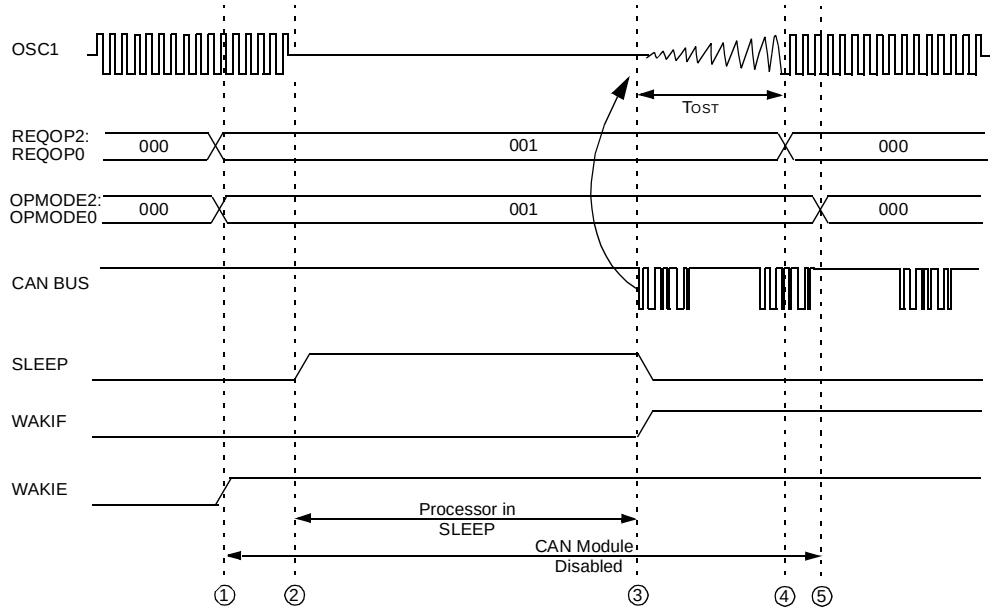
If the module is in CPU SLEEP mode and the WAKIE wake-up interrupt enable is set, the module will generate an interrupt, bringing up the CPU. Due to the delays in starting up the oscillator and CPU, the message activity that caused the wake-up will be lost.

If the module is in CPU SLEEP mode and the WAKIE is not set, no interrupt will be generated and the CPU and the module will continue to sleep.

If the CAN module is in disable mode, the module will wake-up and, depending on the condition of the WAKIE bit, may generate an interrupt. It is expected that the module will correctly receive the message that caused the wake-up from SLEEP mode.

The module can be programmed to apply a low-pass filter function to the RxCAN input line while the module or the CPU is in SLEEP mode. This feature can be used to protect the module from Wake-up due to short glitches on the CAN bus lines. Such glitches can result from electromagnetic inference within noisy environments. The WAKFIL bit enables or disables the filter.

Figure 22-13:Processor SLEEP and CAN Bus Wake-up Interrupt



① - Processor requests and receives module disable mode. Wake-up interrupt enabled.

② - Processor executes SLEEP instruction.

③ - SOF of message wakes up processor. Oscillator start time begins. CAN message lost. WAKIF bit set.

④ - Processor completes oscillator start time. Processor resumes program or interrupt, based on GIE bits. Processor requests normal operating mode. Module waits for 11 recessive bits before accepting CAN bus activity. CAN message lost.

⑤ - Module detects 11 recessive bits. Module will begin to receive messages and transmit any pending messages.

22.7.3 Normal Operation Mode

Normal operating mode is selected when REQOP2:REQOP0 = 000. In this mode, the module is activated, the I/O pins will assume the CAN bus functions. The module will transmit and receive CAN bus messages as described in subsequent paragraphs.

22.7.4 Listen Only Mode

Listen only mode and loopback modes are special cases of normal operation mode to allow system debug. If the listen only mode is activated, the module on the CAN bus is passive. The transmitter buffers revert to Port I/O function. The receive pins remain input. For the receiver, no error flags or acknowledge signals are sent. The error counters are deactivated in this state. The listen only mode can be used for detecting the baud rate on the CAN bus. To use this, it is necessary that there are at least two further nodes that communicate with each other. The baud rate can be detected empirically by testing different values. This mode is also useful as a bus monitor without influencing the data traffic.

22.7.5 Error Recognition Mode

The module can be set to ignore all errors and receive any message. The error recognition mode is activated by setting the RXM1:RXM0 bits in the RXBnCON registers to 11. In this mode the data which is in the message assembly buffer until the error time is copied in the receive buffer and can be read via the CPU interface. In addition the data which was on the internal sampling of the CAN bus at the error time and the state vector of the protocol state machine and the bit counter CntCan are stored in registers and can be read.

22.7.6 Loop Back Mode

If the loopback mode is activated, the module will connect the internal transmit signal to the internal receive signal at the module boundary. The transmit and receive pins revert to their Port I/O function.

The transmitter will receive an acknowledge for its sent messages. Special hardware will generate an acknowledge for the transmitter.

22.8 CAN Bus Initialization

After a RESET the CAN module is in the configuration mode (OPMODE2 is set). The error counters are cleared and all registers contain the reset values. It should be ensured that the initialization is performed before REQOP2 bit is cleared.

22.8.1 Initialization

The CAN module has to be initialized before the activation. This is only possible if the module is in the configuration mode. The configuration mode is requested by setting REQOP2 bit. Only when the status bit OPMODE2 has a high level, the initialization can be performed. Afterwards the configuration registers and the acceptance mask registers and the acceptance filter registers can be written. The module is activated by setting the control bits CFGREQ to zero.

The module will protect the user from accidentally violating the CAN protocol through programming errors. All registers which control the configuration of the module can not be modified while the module is on-line. The CAN module will not be allowed to enter the configuration mode while a transmission is taking place. The CONFIG mode serves as a lock to protect the following registers.

- All Module Control Registers
- Configuration Registers
- Bus Timing Registers
- Identifier Acceptance Filter Registers
- Identifier Acceptance Mask Registers

22.9 Message Reception

This chapter describes the message reception.

22.9.1 Receive Buffers

The CAN bus module has 3 receive buffers. However, one of the receive buffers is always committed to monitoring the bus for incoming messages. This buffer is called the message assembly buffer, MAB. So there are 2 receive buffers visible, RXB0 and RXB1, that can essentially instantaneously receive a complete message from the protocol engine. The CPU can be operating on one while the other is available for reception or holding a previously received message.

The MAB holds the destuffed bit stream from the bus line to allow parallel access to the whole data or Remote Frame for the acceptance match test and the parallel transfer of the frame to the receive buffers. The MAB will assemble all messages received. These messages will be transferred to the RXBn buffers only if the acceptance filter criterion are met. When a message is received, the RXFUL bit will be set. This bit can only be set by the module when a message is received. The bit is cleared by the CPU when it has completed processing the message in the buffer. This bit provides a positive lockout to ensure that the CPU has finished with the message buffer. If the RXnIE bit is set , an interrupt will be generated when a message is received.

There are 2 programmable acceptance filter masks associated with the receive buffers, one for each buffer.

When the message is received, the FILHIT2:FILHIT0 bits (RXBnCON register) indicate the acceptance criterion for the message. The number of the acceptance filter that enabled the reception will be indicated as well as a status bit that indicates that the received message is a remote transfer request.

22.9.1.1 Receive Buffer Priority

To provide flexibility, there are several acceptance filters corresponding to each receive buffer. There is also an implied priority to the receive buffers. RXB0 is the higher priority buffer and has 2 message acceptance filters associated with it. RXB1 is the lower priority buffer and has 4 acceptance filters associated with it. The lower number of possible acceptance filters makes the match on RXB0 more restrictive and implies the higher criticality associated with that buffer. Additionally, if the RXB0 contains a valid message, and another valid message is received, the RXB0 can be setup such that it will not overrun and the new message for RXB0 will be placed into RXB1. Figure 22-14 shows a block diagram of the receive buffer, while Figure 22-15 shows a flow chart for receive operation.

Figure 22-14:The Receive Buffers

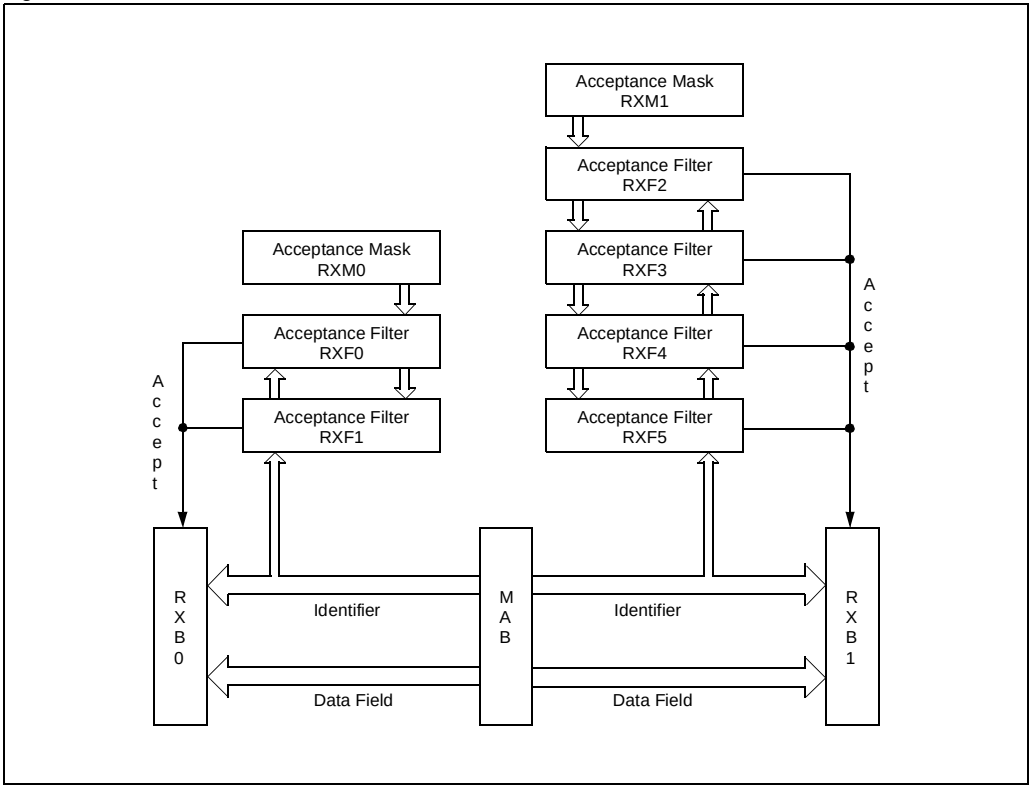
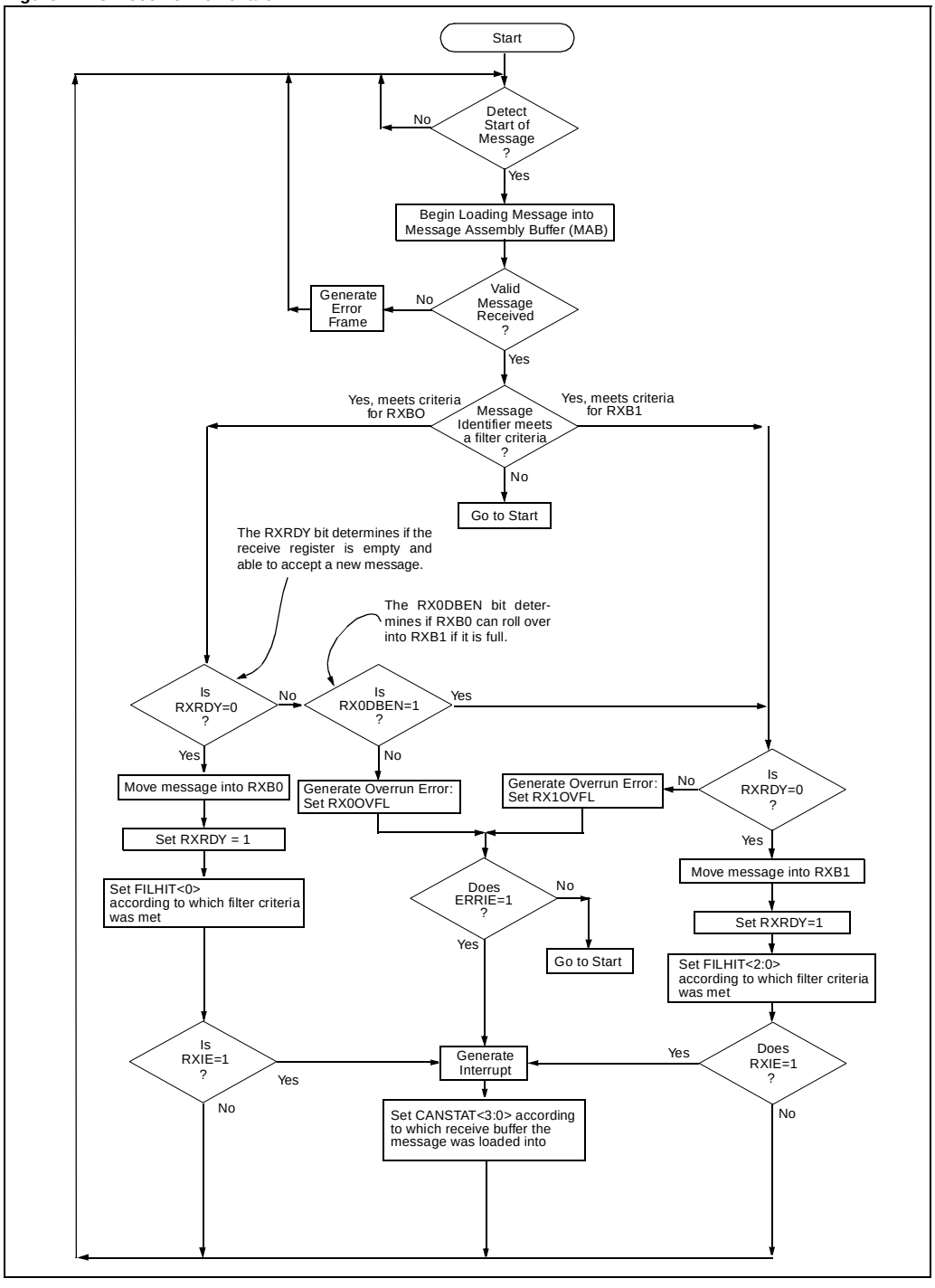


Figure 22-15:Receive Flowchart



22.9.2 Message Acceptance Filters

The message acceptance filters and masks are used to determine if a message in the message assembly buffer should be loaded into either of the receive buffers. Once a valid message has been received into the MAB, the identifier fields of the message are compared to the filter values. If there is a match, that message will be loaded into the appropriate receive buffer. The filter masks are used to determine which bits in the identifiers are examined with the filters. A truth-table is shown in Table 22-2 that indicates how each bit in the identifier is compared to the masks and filters to determine if the message should be loaded into a receive buffer. The mask bit essentially determines which bits to apply the filter to. If any mask bit is set to a zero, then that bit will automatically be accepted regardless of the filter bit.

Table 22-2: Filter/Mask Truth Table

Mask Bit n	Filter Bit n	Message Identifier bit	Accept or reject bit n
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject
1	1	0	Reject
1	1	1	Accept

Legend: x = 0 don't care.

The acceptance filter looks at incoming messages for the EXIDEN bit to determine how to compare the identifiers. If the EXIDEN bit is clear, the message is a standard frame, and only filters with the EXIDEN bit clear are compared. If the EXIDEN bit is set, the message is an extended frame, and only filters with the EXIDEN bit set are compared. Configuring the RXM1:RXM0 bits to 01 or 10 can override the EXIDEN bit.

As shown in the Receive Buffers Block Diagram, Figure 22-14, RXF0 and RXF1 filters with RXM0 mask are associated with RXB0. The filters RXF2, RXF3, RXF4, and RXF5 and the mask RXM1 are associated with RXB1. When a filter matches and a message is loaded into the receive buffer, the number of the filter that enabled the message reception is coded into a portion of the RXBn-CON register. The RXB1CON register contains the FILHIT2:FILHIT0 bits. They are coded as shown in Table 22-3.

Table 22-3: Acceptance Filter

FILHIT2:FILHIT0	Acceptance Filter	Comment
000(1)	RXF0	Only if RX0DBEN = 1
001(1)	RXF1	Only if RX0DBEN = 1
010	RXF2	—
011	RXF3	—
100	RXF4	—
101	RXF5	—

Note 1: Is only valid if the RX0DBEN bit is set.

The coding of the RX0DBEN bit enables these 3 bits to be used similarly to the FILHIT bits and to distinguish a hit on filter RXF0 and RXF1 in either RXB0 or overrun into RXB1.

111 = Acceptance Filter 1 (RXF1)

110 = Acceptance Filter 0 (RXF0)

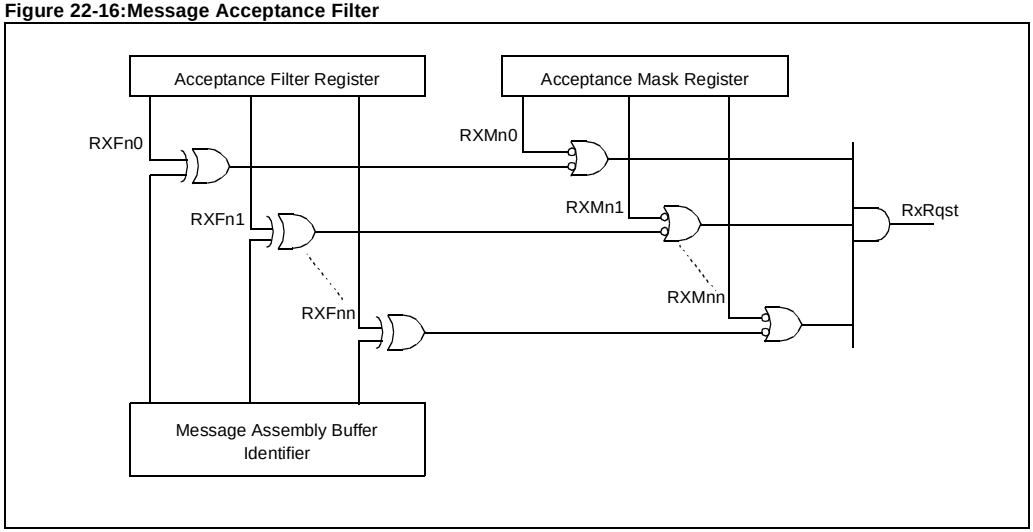
001 = Acceptance Filter 1 (RXF1)

000 = Acceptance Filter 0 (RXF0)

If the RX0DBEN bit is clear, there are 6 codes corresponding to the 6 filters. If the RX0DBEN bit is set, there are 6 codes corresponding to the 6 filters plus 2 additional codes corresponding to RXF0 and RXF1 filters overrun to RXB1.

If more than 1 acceptance filter matches, the FILHIT bits will encode the lowest binary value of the filters that matched. In other words, if filter 2 and filter 4 match, FILHIT will code the value for 2. This essentially prioritizes the acceptance filters with lower numbers having priority. Figure 22-16 shows a block diagram of the message acceptance filters.

Figure 22-16:Message Acceptance Filter



22.9.3 Receiver Overrun

An overrun condition occurs when the MAB has assembled a valid received message, the message is accepted through the acceptance filters, however, the receive buffer associated with the filter has not been designated as clear of the previous message.

The overrun error flag, RXnOVR and the EERIF bit will be set and the message in the MAB will be discarded. While in the overrun situation, the module will stay synchronized with the CAN bus and is able to transmit messages but will discard all incoming messages destined for the overrun buffer.

If the RX0DBEN bit is clear, RXB1 and RXB0 operate independently. When this is the case, a message intended for RXB0 will not be diverted into RXB1 if RXB0 contains an unread message and the RX0OVFL bit will be set.

If the RX0DBEN bit is set, the overrun for RXB0 is handled differently. If a valid message is received for RXB0 and RXFUL = 1 indicates that RXB0 is full and RXFUL = 0 indicates that RXB1 is empty, the message for RXB0 will be loaded into RXB1. An overrun error will not be generated for RXB0. If a valid message is received for RXB0 and RXFUL = 1 and RXFUL = 1 indicating that both RXB0 and RXB1 are full the message will be lost and an overrun will be indicated for RXB1.

If the RX0DBEN bit is clear, there are 6 codes corresponding to the 6 filters. If the RX0DBEN bit is set, there are 6 codes corresponding to the 6 filters plus 2 additional codes corresponding to RXF0 and RXF1 filters overrun to RXB1. These codes are given in Table 22-4.

Table 22-4: Buffer Reception and Overflow Truth Table

Message Matches Filter 0 or 1	Message Matches Filter 2,3,4,5	RXFUL0 Bit	RXFUL1 Bit	RX0DBEN Bit	Action	Action
0	0	X	X	X	None	No message received
0	1	X	0	X	MAB → RXB1	Message for RXB1, RXB1 available
0	1	X	1	X	MAB discarded RX1OVFL = 1	Message for RXB1, RXB1 full
1	0	0	X	X	MAB → RXB0	Message for RXB0, RXB0 available
1	0	1	X	0	MAB discarded RX0OVFL = 1	Message for RXB0, RXB0 full, RX0DBEN not enabled
1	0	1	0	1	MAB → RXB1	Message for RXB0, RXB0 full, RX0DBEN enabled, RXB1 available
1	0	1	1	1	MAB discarded RX1OVFL = 1	Message for RXB0, RXB0 full, RX0DBEN enabled, RXB1 full
1	1	0	X	X	MAB → RXB0	Message for RXB0 and RXB1, RXB0 available
1	1	1	X	0	MAB discarded RX0OVFL = 1	Message for RXB0 and RXB1, RXB0 full, RX0DBEN not enabled
1	1	1	0	1	MAB → RXB1	Message for RXB0 and RXB1, RXB0 full, RX0DBEN enabled, RXB1 available
1	1	1	1	1	MAB discarded RX1OVFL = 1	Message for RXB0 and RXB1, RXB0 full, RX0DBEN enabled, RXB1 full

Legend: X = Don't care.

22.9.4 Effects of a RESET

Upon any RESET the CAN module has to be initialized. All registers are set according to the reset values. The content of a received message is lost. The initialization is discussed in Section **22.8**.

22.9.5 Baud Rate Setting

All nodes on any particular CAN bus must have the same nominal bit rate. The Baud Rate is set once during the initialization mode of the CAN module. After that the baud Rate is not changed again. Section **22.12** explains the setting of the Baud Rate.

22.9.6 Receive Errors

The CAN module will detect the following receive errors:

- Cyclic Redundancy Check (CRC) Error
- Bit Stuffing Error
- Invalid message receive error.

These receive errors do not generate an interrupt. However, the receive error counter is incremented by one in case one of these errors occur. The RXWARN bit indicates that the Receive Error Counter has reached the CPU Warning limit of 96 and an interrupt is generated.

22.9.6.1 Cyclic Redundancy Check (CRC) Error

With the Cyclic Redundancy Check, the transmitter calculates special check bits for the bit sequence from the start of a frame until the end of the Data Field. This CRC sequence is transmitted in the CRC Field. The receiving node also calculates the CRC sequence using the same formula and performs a comparison to the received sequence. If a mismatch is detected, a CRC error has occurred and an Error Frame is generated. The message is repeated. The receive error interrupt counter is incremented by one. An Interrupt will only be generated if the error counter passes a threshold value.

22.9.6.2 Bit Stuffing Error

If in between Start of Frame and CRC Delimiter 6 consecutive bits with the same polarity are detected, the bit-stuffing rule has been violated. A Bit-Stuffing error occurs and an Error Frame is generated. The message is repeated. No Interrupt will be generated upon this event.

22.9.6.3 Invalid Message Received Error

If any type of error occurs during reception of a message, an error will be indicated by the IXRIF bit. This bit can be used (optionally with an interrupt) for autobaud detection with the device in listen-only mode. This error is not an indicator that any action needs to occur, but an indicator that an error has occurred on the CAN bus.

22.9.6.4 Rules for Modifying the Receive Error Counter

The Receive Error Counter is modified according to the following rules:

- When the receiver detects an error, the Receive Error Counter is incremented by 1, except when the detected error was a Bit Error during the transmission of an Active Error Flag or an Overload Flag.
- When the receiver detects a "dominant" bit as the first bit after sending an Error Flag the Receive Error Counter will be incremented by 8.
- If a Receiver detects a Bit Error while sending an Active Error Flag or an Overload Flag the Receive Error Counter is incremented by 8.
- Any Node tolerates up to 7 consecutive "dominant" bits after sending an Active Error Flag, Passive Error Flag or an Overload Flag. After detecting the 14th consecutive "dominant" bit (in case of an Active Error Flag or an Overload flag) or after detecting the 8th consecutive "dominant" following a passive error flag, and after each sequence of additional eight consecutive "dominant" bits every Transmitter increases its Transmission Error Counter and every Receiver increases its Receive Error Counter by 8.
- After a successful reception of a message (reception without error up to the ACK slot and the successful sending of the ACK bit), the Receive Error Counter is decreased by one, if the Receive Error Counter was between 1 and 127. If the Receive Error Counter was 0 it will stay 0. If the Receive Error Counter was greater than 127, it will change to a value between 119 and 127.

22.9.7 Receive Interrupts

Several Interrupts are linked to the message reception. The receive interrupts can be broken up into two separate groups:

- Receive Error Interrupts
- Receive interrupts

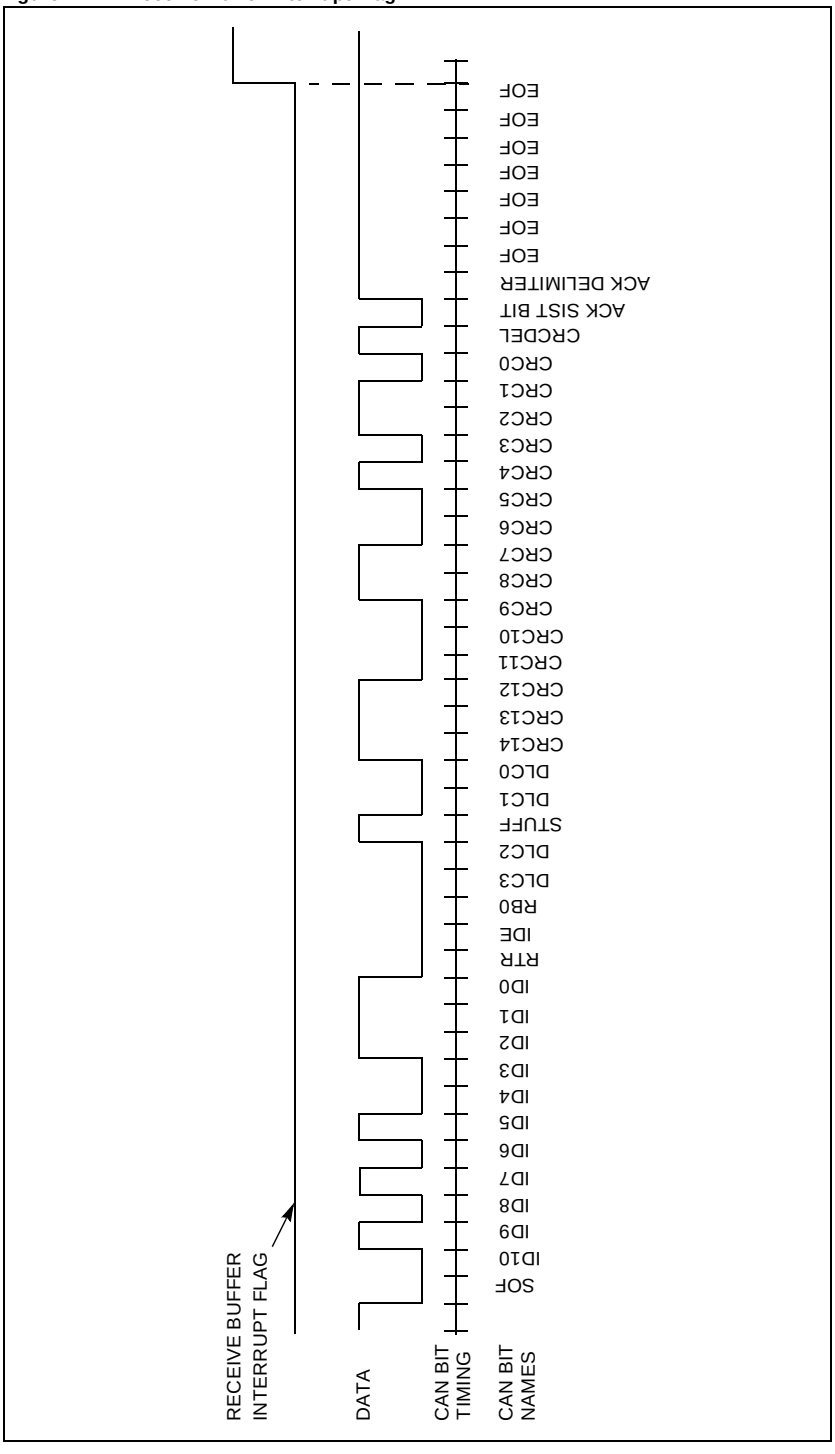
22.9.7.1 Receive Interrupt

A message has been successfully received and loaded into one of the receive buffers. This interrupt is activated immediately after receiving the End of Frame (EOF) field. Reading the RXnIF flag will indicate which receive buffer caused the interrupt. Figure 22-17 depicts when the receive buffer interrupt flag RXnIF will be set.

22.9.7.2 Wake-up Interrupt

The Wake-up Interrupt sequences are described in Section **22.7.2.2**.

Figure 22-17: Receive Buffer Interrupt Flag



22.9.7.3 Receive Error Interrupts

A receive error interrupt will be indicated by the ERRIF bit. This bit shows that an error condition occurred. The source of the error can be determined by checking the bits in the Communication Status Register COMSTAT. The bits in this register are related to receive and transmit errors. The following subsequences will show which flags are linked to the receive errors.

22.9.7.3.1 Invalid Message Received Interrupt

If any type of error occurred during reception of the last message, an error will be indicated by the IXRIF bit. The specific error that occurred is unknown. This bit can be used (optionally with an interrupt) for autobaud detection with the device in listen only mode. This error is not an indicator that any action needs to occur, but an indicator that an error has occurred on the CAN bus.

22.9.7.3.2 Receiver Overrun Interrupt

The RXnOVR bit indicates that an Overrun condition occurred. An overrun condition occurs when the Message Assembly Buffer (MAB) has assembled a valid received message, the message is accepted through the acceptance filters, however, the receive buffer associated with the filter is not clear of the previous message. The overflow error interrupt will be set and the message is discarded. While in the overrun situation, the module will stay synchronized with the CAN bus and is able to transmit and receive messages.

22.9.7.4 Receiver Warning Interrupt

The RXWARN bit indicates that the Receive Error Counter has reached the CPU Warning limit of 96. When RXWARN transitions from a 0 to a 1, it will cause the Error Interrupt Flag ERRIF to become set. This bit cannot be manually cleared, as it should remain an indicator that the Receive Error Counter has reached the CPU Warning limit of 96. The RXWARN bit will become clear automatically if the Receive Error Counter becomes less than or equal to 95. The ERRIF bit can be manually cleared allowing the interrupt service routine to be exited without affecting the RXWARN bit.

22.9.7.5 Receiver Error Passive

The RXBP bit indicates that the Receive Error Counter has exceeded the Error Passive limit of 127 and the module has gone to Error Passive state. When the RXBP bit transitions from a 0 to a 1, it will cause the Error Interrupt Flag to become set. The RXBP bit cannot be manually cleared, as it should remain an indicator that the Bus is in Error State Passive. The RXBP bit will become clear automatically if the Receive Error Counter becomes less than or equal to 127. The ERRIF bit can be manually cleared allowing the interrupt service routine to be exited without affecting the RXBP bit.

22.9.8 Receive Modes

The RXM1:RXM0 bits will set special receive modes. Normally, these bits are set to 00 to enable reception of all valid messages as accepted by the acceptance filters. In this case, the determination of whether or not to receive standard or extended messages is determined by the EXIDEN bit in the Acceptance Filter Registers. If the RXM1:RXM0 bits are set to 01 or 10, the receiver will accept only messages with standard or extended identifiers respectively. If an acceptance filter has the EXIDEN bit such that it does not correspond with the RXM1:RXM0 mode, that acceptance filter is rendered useless. These 2 modes of RXM1:RXM0 bits can be used in systems where it is known that only standard or extended messages will be on the bus. If the RXM1:RXM0 bits are set to 11, the buffer will receive all messages regardless of the values of the acceptance filters. Also, if a message has an error before the End of Frame, that portion of the message assembled in the Message Assembly Buffer (MAB) before the error frame will be loaded into the buffer. This mode may have some value in debugging a CAN system.

22.9.8.1 Listen Only Mode

If the receive only mode is activated, the module on the CAN bus is passive. That means that no error flags or acknowledge signals are sent. The error counters are deactivated in this state. The receive only mode can be used for detecting the baud rate on the CAN bus. For this it is necessary that there are at least two further nodes, which communicate with each other. The baud rate can be detected empirically by testing different values. This mode is also useful as a bus monitor without influencing the data traffic.

22.9.8.2 Error Recognition Mode

The module can be set to ignore all errors and receive any message. The error recognition mode is activated by configuring the RXM1:RXM0 bits (RXBnCON registers) = '11'. In this mode the data which is in the message assembly buffer until the error time is copied in the receive buffer and can be read via the CPU interface.

22.10 Transmission

This subsection describes how the CAN module is used for receiving CAN messages.

22.10.1 Real Time Communication and Transmit Message Buffering

For an application to effectively transmit messages in real time, the CAN nodes must be able to dominate and hold the bus assuming that nodes messages are of a high enough priority to win arbitration on the bus. If a node only has 1 transmission buffer, it must transmit a message, then release the bus while the CPU reloads the buffer. If a node has two transmission buffers, one buffer could be transmitting while the second buffer is being reloaded. However, the CPU would need to maintain tight tracking of the bus activity to ensure that the second buffer is reloaded before the first message completes.

Typical applications require three transmit message buffers. Having three buffers, one buffer can be transmitting, the second buffer can be ready to transmit as soon as the first is complete, and the third can be reloaded by the CPU. This eases the burden of the software to maintain synchronization with the bus (see Figure 22-18).

Additionally, having three buffers allows some degree of prioritizing of the outgoing messages. For example, the application software may have a message enqueued in the second buffer while it is working on the third buffer. The application may require that the message going into the third buffer is of higher importance than the one already enqueued. If only 2 buffers are available, the enqueued message would have to be deleted and replaced with the third. The process of deleting the message may mean losing control of the bus. With 3 buffers, both the second and the third message can be enqueued, and the module can be instructed that the third message is higher priority than the second. The third message will be the next one sent followed by the second.

22.10.2 Transmit Message Buffers

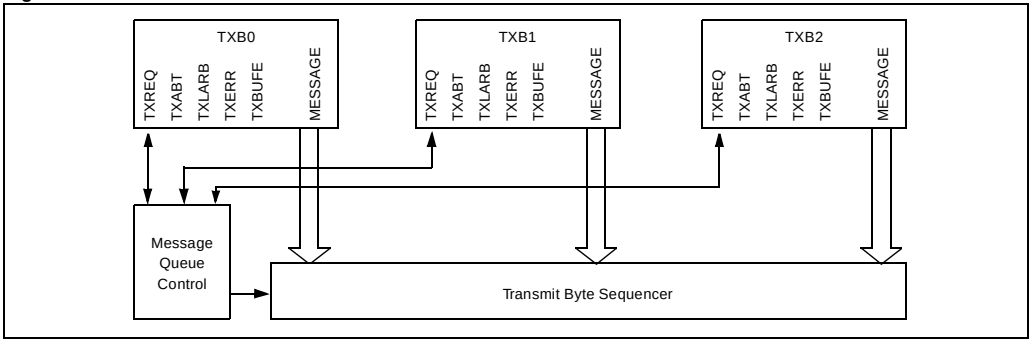
The CAN module has three transmit buffers. Each of the three buffers occupies 14 bytes of data. Eight of the bytes are the maximum 8 bytes of the transmitted message. Five bytes hold the standard and extended identifiers and other message arbitration information.

The last byte is a control byte associated with each message. The information in this byte determines the conditions under which the message will be transmitted and indicates status of the transmission of the message.

The TXBnIF bit will be set and the TXREQ bit will be clear, indicating that the message buffer has completed a transmission. The CPU will then load the message buffer with the contents of the message to be sent. At a minimum, the standard identifier register TXBnSIDH and TXBnSIDL must be loaded. If data bytes are present in the message, the TXBnDm registers are loaded. If the message is to use extended identifiers, the TXBnEIDm registers are loaded and the EXIDEN bit is set.

Prior to sending the message, the user must initialize the TXIE bit to enable or disable an interrupt when the message is sent. The user must also initialize the transmit priority. Figure 22-18 shows a block diagram of the transmit buffers.

Figure 22-18:Transmit Buffers



22.10.3 Transmit Message Priority

Transmit priority is a prioritization within each node of the pending transmittable messages. Prior to sending the SOF (Start of Frame), the priorities of all buffers ready for transmission are compared. The transmit buffer with the highest priority will be sent first. For example, if transmit buffer 0 has a higher priority setting than transmit buffer 1, buffer 0 will be sent first. If two buffers have the same priority setting, the buffer with the highest address will be sent. For example, if transmit buffer 1 has the same priority setting as transmit buffer 0, buffer 1 will be sent first. There are 4 levels of transmit priority. If TXPRI1:TXPRI0 for a particular message buffer is set to 11, that buffer has the highest priority. If TXPRI1:TXPRI0 for a particular message buffer is set to 10 or 01, that buffer has an intermediate priority. If TXPRI1:TXPRI0 for a particular message buffer is 00, that buffer has the lowest priority.

22.10.4 Message Transmission

To initiate transmitting the message, the TXREQ bit must be set. The CAN bus module resolves any timing conflicts between setting of the TXREQ bit and the SOF time, ensuring that if the priority was changed, it is resolved correctly before SOF. When TXREQ is set the TXABT, TXLARB and TXERR flag bits will be cleared.

Setting TXREQ bit does not actually start a message transmission, it flags a message buffer as enqueued for transmission. Transmission will start when the module detects an available bus for SOF. The module will then begin transmission on the message which has been determined to have the highest priority.

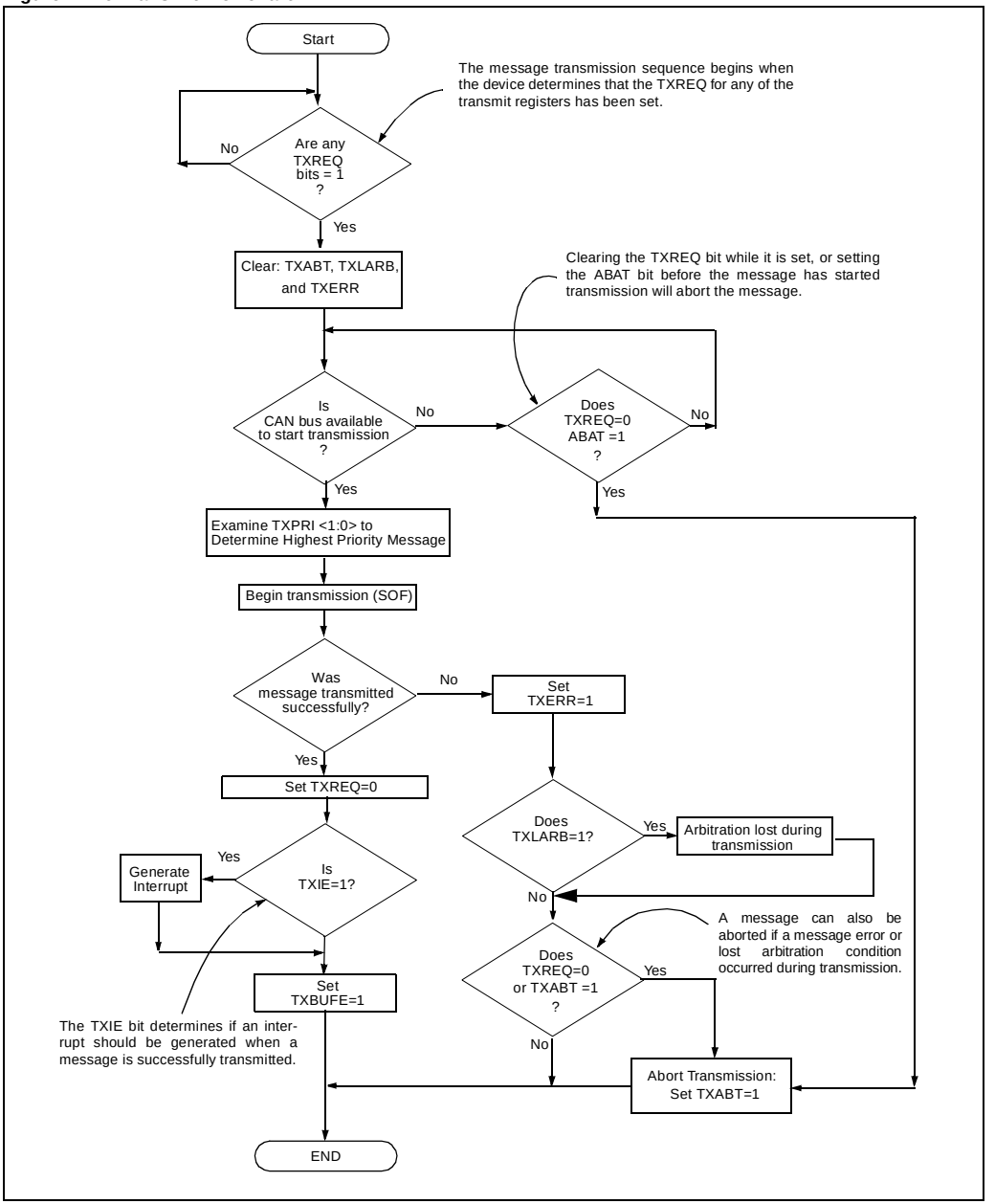
If the transmission completes successfully on the first try, the TXREQ bit will clear and an interrupt will be generated if TXIE was set.

If the message fails to transmit, one of the other condition flags will be set, the TXREQ bit will remain set indicating that the message is still pending for transmission. If the message tried to transmit but encountered an error condition, the TXERR bit will be set. In this case, the error condition can also cause an interrupt. If the message tried to transmit but lost arbitration, the TXLARB bit will be set. In this case, no interrupt is available to signal the loss of arbitration.

22.10.5 Transmit Message Aborting

The system can also abort a message by clearing the TXREQ bit associated with each message buffer. Setting the ABAT bit will request an abort of all pending messages. If the message has not yet started transmission, or if the message started but is interrupted by loss of arbitration or an error; the abort will be processed. The abort is indicated when the module sets the TXABT bit, and the TXnIF flag is not automatically set.

Figure 22-19:Transmit Flowchart



22.10.6 Transmit Boundary Conditions

The module handles transmit commands which are not necessarily synchronized to the CAN bus message framing time.

22.10.6.1 Clearing TXREQ bit as a Message Starts

The TXREQ bit can be cleared just when a message is starting transmission, with the intent of aborting the message. If the message is not being transmitted, the TXABT bit will be set, indicating that the Abort was successfully processed.

When the user clears the TXREQ bit and the TXABT bit is not set two cycles later, the message has already begun transmission.

If the message is being transmitted, the abort is not immediately processed, at some point later, the TXnIF Interrupt Flag or the TXABT bit is set. If transmission has begun the message will only be aborted if either an error or a loss of arbitration occurs.

22.10.6.2 Setting TXABT bit as a Message Starts

Setting the ABAT bit will abort all pending transmit buffers and has the function of clearing all of the TXREQ bits for all buffers. The boundary conditions are the same as clearing the TXREQ bit.

22.10.6.3 Clearing TXREQ bit as a Message Completes

The TXREQ bit can be cleared when a message is just about to successfully complete transmission. Even if the TXREQ bit is cleared by the Data bus a short time before it will be cleared by the successful transmission of the message, the TXnIF flag will still be set due to the successful transmission.

22.10.6.4 Setting TXABT bit as a Message Completes

The boundary conditions are the same as clearing the TXREQ bit.

22.10.6.5 Clearing TXREQ bit as a Message Loses Transmission

The TXREQ bit can be cleared when a message is just about to be lost to arbitration or an error.

If the TXREQ signal falls before the loss of arbitration signal or error signal, the result will be like clearing TXREQ during transmission. When the arbitration is lost or the error is set, the TXABT bit will be set, as it will see that an error has occurred while transmitting, and that the TXREQ bit was not set.

If the TXREQ bit falls after the arbitration signal has entered the block, the result will be like clearing TXREQ during an inactive transmit time. The TXABT bit will be set.

22.10.6.6 Setting TXABT bit as a Message Loses Transmission

The boundary conditions are the same as clearing the TXREQ bit.

22.10.7 Effects of a RESET

Upon any RESET the CAN module has to be initialized. All registers are set according to the reset values. The content of a received message is lost. The initialization is discussed in Section **22.8**.

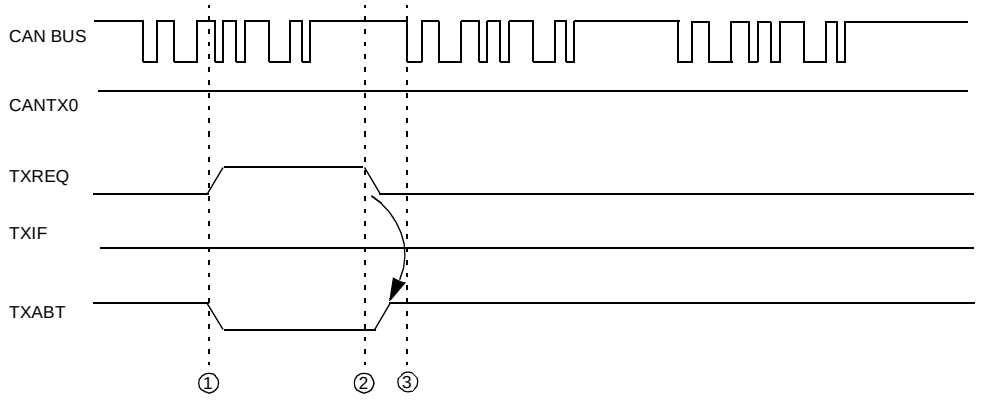
22.10.8 Baud Rate Setting

All nodes on any particular CAN bus must have the same nominal bit rate. The baud rate is set once during the initialization phase of the CAN module. After that, the baud rate is not changed again. Section **22.12** explains the setting of the Baud Rate.

22.10.9 Transmit Message Aborting

The system can also abort a message by clearing the TXREQ bit associated with each message buffer. Setting the ABAT bit will request an abort of all pending messages (see Figure 22-21). A queued message is aborted by clearing the TXREQ bit. Aborting a queued message is illustrated in Figure 22-20. If the message has not yet started transmission, or if the message started but is interrupted by loss of arbitration or an error; the abort will be processed. The abort is indicated when the module sets the TXABT bits. If the message has started to transmit, it will attempt to transmit the current message fully (see Figure 22-22). If the current message is transmitted fully, and is not lost to arbitration or an error, the TXABT bit will not be set, because the message was transmitted successfully. Likewise, if a message is being transmitted during an abort request, and the message is lost to arbitration (see Figure 22-23) or an error, the message will not be re-transmitted, and the TXABT bit will be set, indicating that the message was successfully aborted.

Figure 22-20:Abort Queued Message

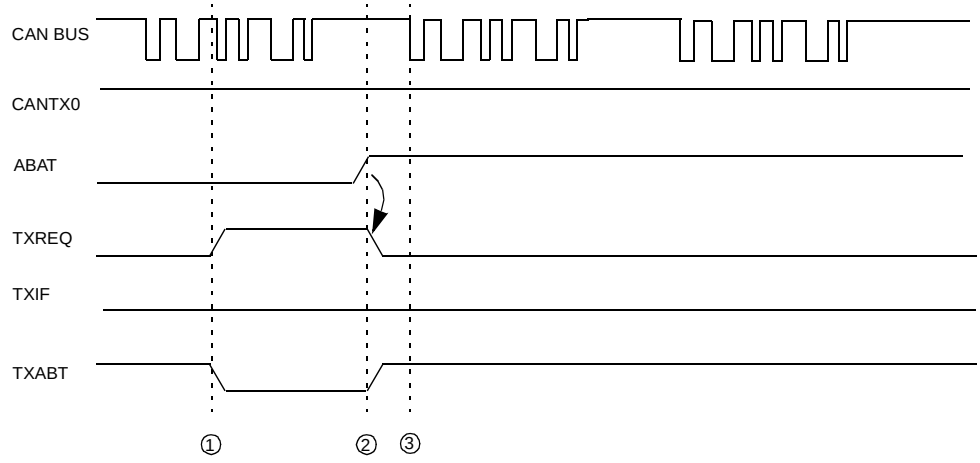


① - Processor sets TXREQ while module receiving/transmitting message. Module continues with CAN message.

② - Processor clears TXREQ while module looking for 11 recessive bits.
Module aborts pending transmission, sets TXABT bit in 2 clocks.

③ - Another module takes the available transmit slot.

Figure 22-21:Abort All Messages

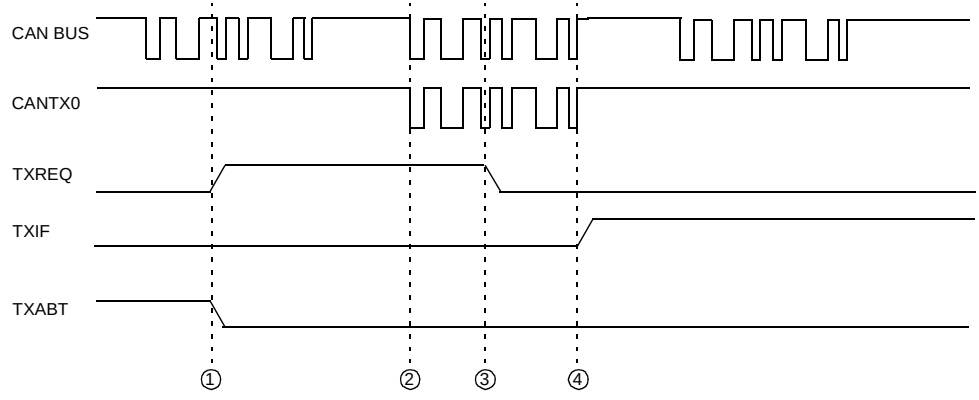


① - Processor sets TXREQ while module receiving/transmitting message. Module continues with CAN message.

② - Processor sets ABAT while module looking for 11 recessive bits. Module clears TXREQ bits. Module aborts pending transmission, sets TXABT bit.

③ - Another module takes the available transmit slot.

Figure 22-22:Failed Abort During Transmission



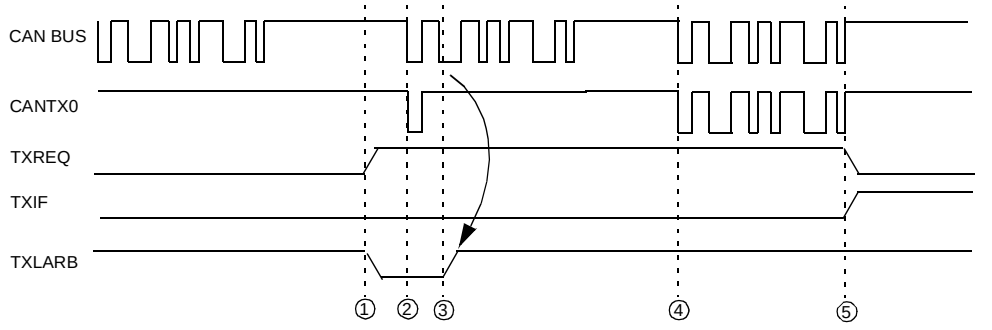
① - Processor sets TXREQ while module receiving/transmitting message. Module continues with CAN message.

② - Module detects 11 recessive bits. Module begins transmission of queued message.

③ - Processor clears TXREQ requesting message abort. Abort cannot be acknowledged.

④ - At successful completion of transmission, TXREQ bit remains clear and TXIF bit set. TXABT remains clear.

Figure 22-23:Loss of Arbitration During Transmission



① - Processor sets TXREQ while module inactive. TXLARB bit cleared.
② - Module in inactive state. Module begins transmission of queued message.
③ - Message loses arbitration. Module releases bus and sets TXLARB bit.
④ - Module waits for 11 recessive bits before re-trying transmission of queued message.
⑤ - At successful completion of transmission, TXREQ bit cleared and TXIF bit set.

22.10.10 Transmission Errors

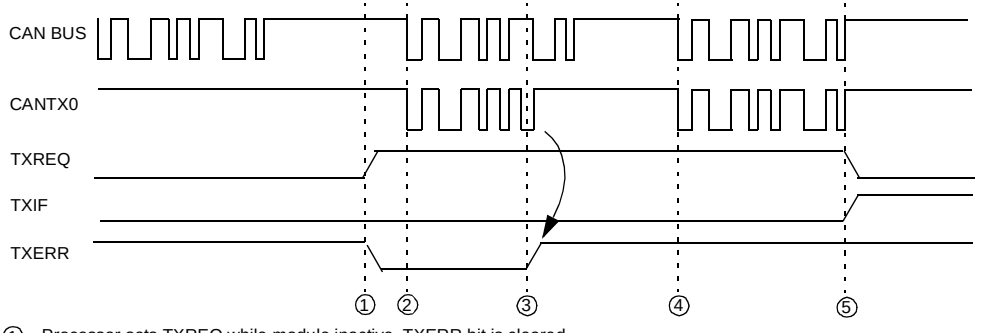
The CAN module will detect the following transmission errors:

- Acknowledge Error
- Form Error
- Bit Error

These transmission errors will not necessarily generate an interrupt but are indicated by the transmission error counter. However, each of these errors will cause the transmission error counter to be incremented by one. Once the value of the error counter exceeds the value of 96, the ERRIF and the TXWARN bit are set. Once the value of the error counter exceeds the value of 96 an interrupt is generated and the TXWARN bit in the error flag register is set.

An example transmission error is illustrated in Figure 22-24.

Figure 22-24:Error During Transmission



① - Processor sets TXREQ while module inactive. TXERR bit is cleared.
② - Module in inactive state. Module begins transmission of queued message.
③ - Module detects error during transmission, releases bus and sets TXERR bit.
④ - Module waits for 11 recessive bits before re-trying transmission of queued message.
⑤ - At successful completion of transmission, TXREQ bit cleared and TXIF bit set.

22.10.10.1 Acknowledge Error

In the Acknowledge Field of a message, the transmitter checks if the Acknowledge Slot (which it has sent out as a recessive bit) contains a dominant bit. If not, no other node has received the frame correctly. An Acknowledge Error has occurred and the message has to be repeated. No Error Frame is generated.

22.10.10.2 Form Error

If a transmitter detects a dominant bit in one of the four segments including End of Frame, Interframe Space, Acknowledge Delimiter or CRC Delimiter; then a Form Error has occurred and an Error Frame is generated. The message is repeated.

22.10.10.3 Bit Error

A Bit Error occurs if a transmitter sends a dominant bit and detects a recessive bit. In the case where the transmitter sends a recessive bit and a dominant bit is detected during the Arbitration Field and the Acknowledge Slot, no bit error is generated because normal arbitration is occurring.

22.10.10.4 Rules for Modifying the Transmit Error Counter

The Transmit Error Counter is modified according to the following rules:

- When the Transmitter sends an error flag the Transmit Error Counter is increased by 8 with the following exceptions. In these two exceptions, the Transmit Error Counter is not changed.
  - If the transmitter is "error passive" and detects an acknowledgment error because of not detecting a "dominant" ACK, and does not detect a "dominant" bit while sending a Passive Error Flag.
  - If the Transmitter sends an Error Flag because of a bit-stuffing Error occurred during arbitration whereby the Stuffbit is located before the RTR bit, and should have been "recessive", and has been sent as "recessive" but monitored as "dominant".
- If a Transmitter detects a Bit Error while sending an Active Error Flag or an Overload Flag the Transmit Error Counter is increased by 8.
- Any Node tolerates up to 7 consecutive "dominant" bits after sending an Active Error Flag, Passive Error Flag or an Overload Flag. After detecting the 14th consecutive "dominant" bit (in case of an Active Error Flag or an Overload flag) or after detecting the 8th consecutive "dominant" following a passive error flag, and after each sequence of eight additional consecutive "dominant" bits, every Transmitter increases its Transmission Error Counter and every Receiver increases its Receive Error Counter by 8.
- After the successful transmission of a message (getting an acknowledge and no error until End of Frame is finished) the Transmit Error Counter is decreased by one unless it was already 0.

22.10.11 Transmission Interrupts

There are several interrupts linked to the message transmission. The transmission interrupts can be broken up into two groups:

- Transmission interrupts
- Transmission error interrupts

22.10.11.1 Transmit Interrupt

At least one of the three transmit buffers is empty (not scheduled) and can be loaded to schedule a message for transmission. Reading the TXIF flags will indicate which transmit buffer is available and caused the interrupt.

22.10.11.2 Transmission Error Interrupts

A transmission error interrupt will be indicated by the ERRIF flag. This flag shows that an error condition occurred. The source of the error can be determined by checking the error flags in the Communication Status register COMSTAT. The flags in this register are related to receive and transmit errors. The following subsequences will show which flags are linked to the transmit errors.

22.10.11.3 Transmitter Warning Interrupt

The TXWARN bit indicates that the Transmit Error Counter has reached the CPU Warning limit of 96. When this bit transitions from a 0 to a 1, it will cause the Error Interrupt Flag to become set. The TXWARN bit cannot be manually cleared, as it should remain as an indicator that the Transmit Error Counter has reached the CPU Warning limit of 96. The TXWARN bit will become clear automatically if the Transmit Error Counter becomes less than or equal to 95. The ERRIF flag can be manually cleared allowing the interrupt service routine to be exited without affecting the TXWARN bit.

22.10.11.4 Transmitter Error Passive

The TXEP bit indicates that the Transmit Error Counter has exceeded the Error Passive limit of 127 and the module has gone to Error Passive state. When this bit transitions from a 0 to a 1, it will cause the Error Interrupt Flag to become set. The TXEP bit cannot be manually cleared, as it should remain as an indicator that the Bus is in Error State Passive. The TXEP bit will become clear automatically if the Transmit Error Counter becomes less than or equal to 127. The ERRIF flag can be manually cleared allowing the interrupt service routine to be exited without affecting the TXEP bit.

22.10.11.5 Bus Off Interrupt

The TXBO bit indicates that the Transmit Error Counter has exceeded 255 and the module has gone to Bus Off state. When this bit transitions from a 0 to a 1, it will cause the Error Interrupt Flag to become set. The TXBO bit cannot be manually cleared, as it should remain as an indicator that the Bus is Off. The ERRIF flag can be manually cleared allowing the interrupt service routine to be exited without affecting the TXBO bit.

22.11 Error Detection

The CAN protocol provides sophisticated error detection mechanisms. The following errors can be detected. These errors are either receive or transmit errors.

Receive errors are:

- Cyclic Redundancy Check (CRC) Error (see Section **22.9.6.1**)
- Bit Stuffing Bit Error (see Section **22.9.6.2**)
- Invalid Message Received Error (see Section **22.9.6.2**)

The transmit errors are

- Acknowledge Error (see Section **22.10.10.1**)
- Form Error (see Section **22.10.10.2**)
- Bit Error (see Section **22.10.10.3**)

22.11.1 Error States

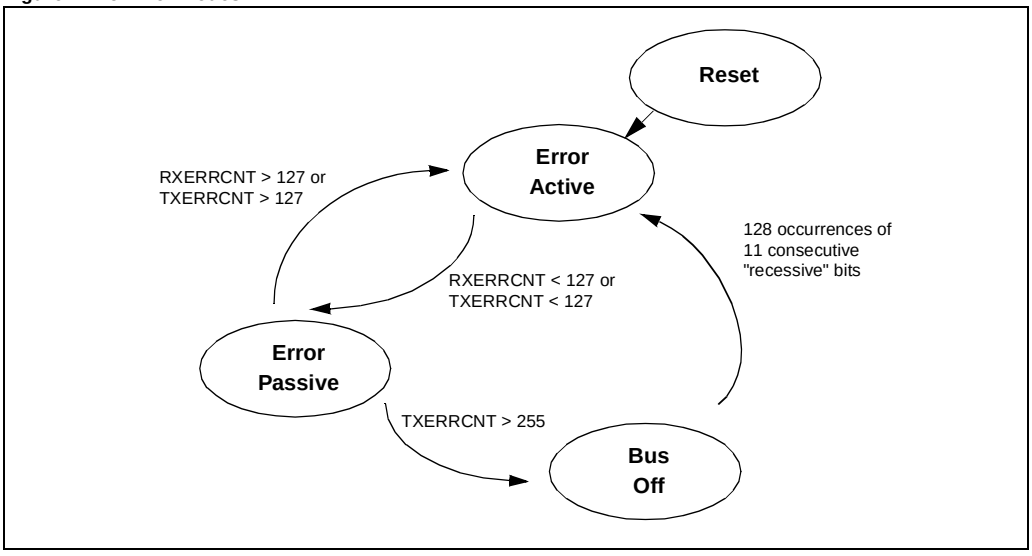
Detected errors are made public to all other nodes via Error Frames. The transmission of the erroneous message is aborted and the frame is repeated as soon as possible. Furthermore, each CAN node is in one of the three error states "error active", "error passive" or "bus off" according to the value of the internal error counters. The error-active state is the usual state where the bus node can transmit messages and active Error Frames (made of dominant bits) without any restrictions. In the error-passive state, messages and passive Error Frames (made of recessive bits) may be transmitted. The bus-off state makes it temporarily impossible for the station to participate in the bus communication. During this state, messages can neither be received nor transmitted.

22.11.2 Error Modes and Error Counters

The CAN controller contains the two error counters Receive Error Counter (RXERRCNT) and Transmit Error Counter (TXERRCNT). The values of both counters can be read by the CPU. These counters are incremented or decremented according to the CAN bus specification.

The CAN controller is error active if both error counters are below the error passive limit of 128. It is error passive if at least one of the error counters equals or exceeds 128. It goes bus off if the Transmit Error Counter equals or exceeds the bus off limit of 256. The device remains in this state, until the bus off recovery sequence is finished, which is 128 consecutive 11 recessive bit times. Additionally, there is a error state warning flag bit, EWARN, which is set if at least one of the error counters equals or exceeds the error warning limit of 96. EWARN is reset if both error counters are less than the error warning limit.

Figure 22-25:Error Modes



22.11.3 Error Flag Register

The values in the error flag register indicate which error(s) caused the Error Interrupt Flag. The RXXOVR Error Flags have a different function than the other Error Flag bits in this register. The RXXOVR bits must be cleared in order to clear the ERRIF interrupt flag. The other Error Flag bits in this register will cause the ERRIF interrupt flag to become set as the value of the Transmit and Receive Error Counters crosses a specific threshold. Clearing the ERRIF interrupt flag in these cases will allow the interrupt service routine to be exited without recursive interrupt occurring. It may be desirable to disable specific interrupts after they have occurred once to stop the device from interrupting repeatedly as the Error Counter moves up and down in the vicinity of a threshold value.

22.12 Baud Rate Setting

All nodes on any particular CAN bus must have the same nominal bit rate. The CAN bus uses NRZ coding which does not encode a clock. Therefore the receivers independent clock must be recovered by the receiving nodes and synchronized to the transmitters clock.

In order to set the baud rate the following bits have to be initialized:

- Synchronization Jump Width (see Section **22.12.6.2**)
- Baud rate prescaler (see Section **22.12.2**)
- Phase segments (see Section **22.12.4**)
- Length determination of Phase segment 2 (see Section **22.12.4**)
- Sample Point (see Section **22.12.5**)
- Propagation segment bits (see Section **22.12.3**)

22.12.1 Bit Timing

As oscillators and transmission time may vary from node to node, the receiver must have some type of PLL synchronized to data transmission edges to synchronize and maintain the receiver clock. Since the data is NRZ coded, it is necessary to include bit-stuffing to ensure that an edge occurs at least every 6 bit times, to maintain the Digital Phase Lock Loop (DPLL) synchronization.

Bus timing functions executed within the bit time frame, such as synchronization to the local oscillator, network transmission delay compensation, and sample point positioning, are defined by the programmable bit timing logic of the DPLL.

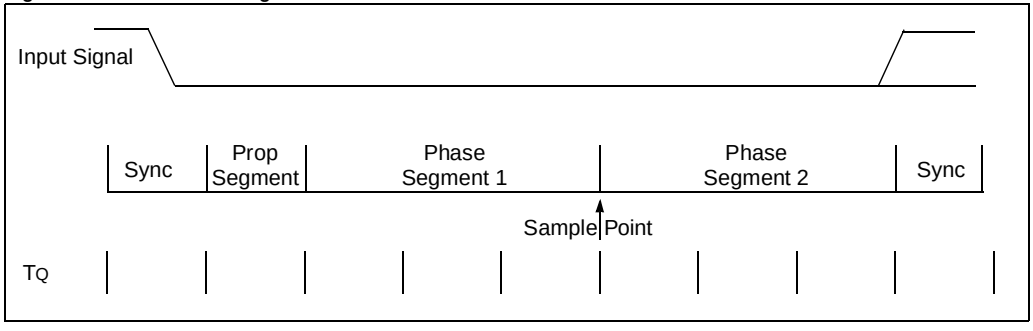
All controllers on the CAN bus must have the same baud rate and bit length. However, different controllers are not required to have the same master oscillator clock. At different clock frequencies of the individual controllers, the baud rate has to be adjusted by adjusting the number of time quanta in each segment.

The Nominal Bit Time can be thought of as being divided into separate non-overlapping time segments. These segments are shown in Figure 22-26.

- Synchronization segment (Sync Seg)
- Propagation time segment (Prop Seg)
- Phase buffer segment 1 (Phase1 Seg)
- Phase buffer segment 2 (Phase2 Seg)

The time segments and also the nominal bit time are made up of integer units of time called time quanta or TQ. By definition, the Nominal Bit Time has a minimum of 8 TQ and a maximum of 25 TQ. Also, by definition the minimum nominal bit time is 1 usec, corresponding to a maximum 1 MHz bit rate.

Figure 22-26:CAN Bit Timing



22.12.2 Prescaler Setting

There is a programmable prescaler, with integral values ranging at least from 1 to 64, in addition to a fixed divide by 2 for clock generation. The Time Quanta (TQ) is a fixed unit of time derived from the oscillator period. Time quanta is defined as:

Equation 22-1:Time Quanta for Clock Generation

$$T_Q = 2 \cdot (BaudRate + 1) \cdot T_{OSC}$$

Where Baud Rate is the binary value of BRP <5:0>

Example 22-1:Calculation for Fosc = 16 MHz

If FOSC = 16 MHz, BRP5:BRP0 = 00h, and Nominal Bit Time = 8 TQ; then TQ = 125 nsec and Nominal Bit Rate = 1 MHz

Example 22-2:Calculation for Fosc = 32 MHz

If FOSC = 32 MHz, BRP5:BRP0 = 01h, and Nominal Bit Time = 8 TQ; then TQ = 125 nsec and Nominal Bit Rate = 1 MHz

Example 22-3:Calculation for Fosc = 32 MHz and 25 TQ

If FOSC = 32 MHz, BRP5:BRP0 = 3Fh, and Nominal Bit Time = 25 TQ; then TQ = 4 usec and Nominal Bit Rate = 10 kHz

The frequencies of the oscillators in the different nodes must be coordinated in order to provide a system-wide specified time quantum. This means that all oscillators must have a Tosc that is a integral divisor of TQ.

22.12.3 Propagation Segment

This part of the bit time is used to compensate physical delay times within the network. These delay times consist of the signal propagation time on the bus line and the internal delay time of the nodes. The delay is calculated as the round trip from transmitter to receiver as twice the signal's propagation time on the bus line, the input comparator delay, and the output driver delay. The Propagation Segment can be programmed from 1 TQ to 8 TQ by setting the PRSE2:PRSEG0 bits.

22.12.4 Phase Segments

The phase segments are used to optimally locate the sampling of the received bit within the transmitted bit time. The sampling point is between Phase1 Segment and Phase2 Segment. These segments are lengthened or shortened by resynchronization. The end of the Phase1 Segment determines the sampling point within a bit period. The segment is programmable from 1 TQ to 8 TQ. Phase2 Segment provides delay to the next transmitted data transition. The segment is programmable from 1 TQ to 8 TQ or it may be defined to be equal to the greater of Phase1 Segment or the Information Processing Time. The phase segment 1 is initialized by setting bits SEG1PH2:SEG1PH0, and phase segment 2 is initialized by setting SEG2PH2:SEG2PH0.

22.12.5 Sample Point

The sample point is the point of time at which the bus level is read and interpreted as the Value of that respective bit. The location is at the end of Phase Segment 1. If the bit timing is slow and contains many TQ, it is possible to specify multiple sampling of the bus line at the sample point. The level determined by the CAN bus then corresponds to the result from the majority decision of three values. The majority samples are taken at the sample point and twice before with a distance of TQ/2. The CAN module allows to chose between sampling three times at the same point or once at the same point. This is done by setting or clearing the SAM bit (BRG2CON register).

22.12.6 Synchronization

To compensate for phase shifts between the oscillator frequencies of the different bus stations, each CAN controller must be able to synchronize to the relevant signal edge of the incoming signal. When an edge in the transmitted data is detected, the logic will compare the location of the edge to the expected time (Synchronous Segment). The circuit will then adjust the values of Phase1 Segment and Phase2 Segment. There are 2 mechanisms used to synchronize.

22.12.6.1 Hard Synchronization

Hard Synchronization is only done whenever there is a 'recessive' to 'dominant' edge during Bus Idle, indicating the start of a message. After hard synchronization, the bit time counters are restarted with Synchronous Segment. Hard synchronization forces the edge which has caused the hard synchronization to lie within the synchronization segment of the restarted bit time. Due to the rules of synchronization, if a hard synchronization is done, there will not be a resynchronization within that bit time.

22.12.6.2 Resynchronization

As a result of resynchronization Phase Segment 1 may be lengthened or Phase Segment 2 may be shortened. The amount of lengthening or shortening (SJW1:SJW0) of the phase buffer segment has an upper bound given by the resynchronization jump width bits. The value of the synchronization jump width will be added to Phase Segment 1 or subtracted from Phase Segment 2. The resynchronization jump width is programmable between 1 TQ and 4 TQ.

Clocking information will only be derived from transitions of recessive to dominant bus states. The property that only a fixed maximum number of successive bits have the same value ensures resynchronizing a bus unit to the bit stream during a frame (e.g. bit-stuffing).

The Phase Error of an edge is given by the position of the edge relative to Synchronous Segment, measured in Time Quanta. The Phase Error is defined in magnitude of TQ as follows:

- e = 0 if the edge lies within Synchronous Segment.
- e > 0 if the edge lies before the Sample Point.
- e < 0 if the edge lies after the Sample Point of the previous bit.

If the magnitude of the phase error is less than or equal to the programmed value of the resynchronization jump width, the effect of a resynchronization is the same as that of a hard synchronization,

If the magnitude of the phase error is larger than the resynchronization jump width, and if the phase error is positive, then Phase Segment 1 is lengthened by an amount equal to the resynchronization jump width.

If the magnitude of the phase error is larger than the resynchronization jump width, and if the phase error is negative, then Phase Segment 2 is shortened by an amount equal to the resynchronization jump width.

Figure 22-27:Lengthening a Bit Period

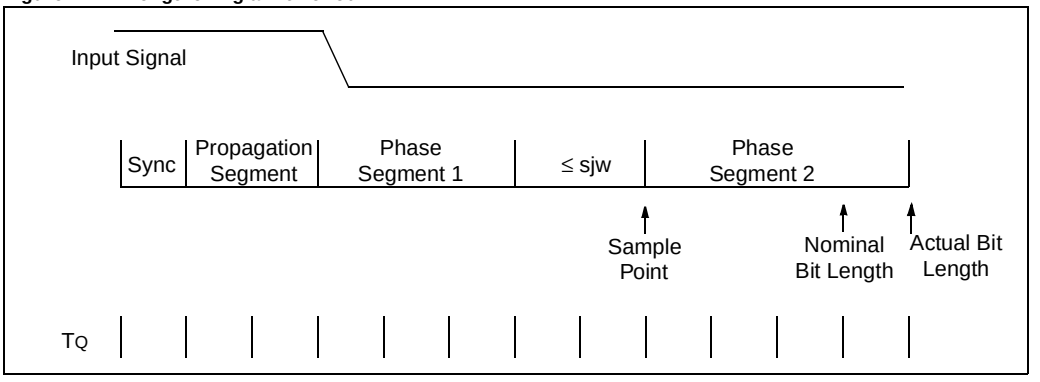
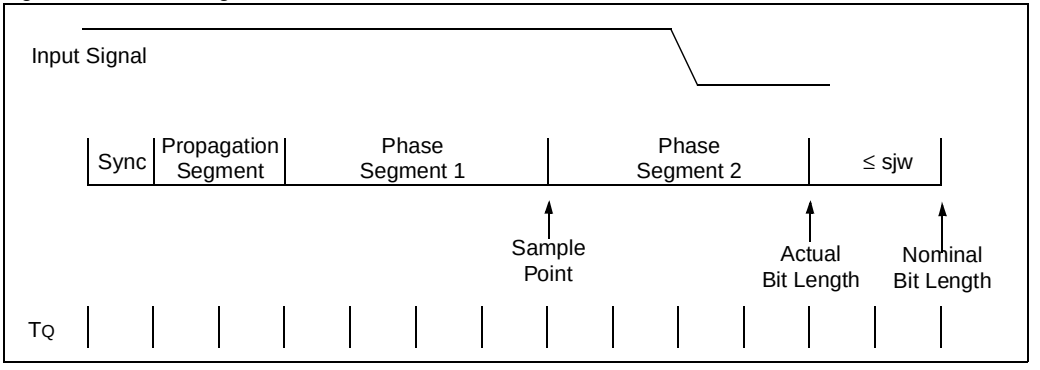


Figure 22-28:Shortening a Bit Period



22.12.7 Programming Time Segments

Some requirements for programming of the time segments:

Propagation Segment + Phase1 Segment > = Phase2 Segment

Phase2 Segment > Synchronous Jump Width

Example 22-4:Segment Time

CAN Baud Rate = 125 kHz
FOSC = 20 MHz

Then:
TOSC = 50 nsec
BRP5:BRP0 = 04h, → TQ = 500 nsec

For:
125 kHz
bit time = 16 TQ

Typically, the sampling of the bit should take place at about 60 - 70% of the bit time, depending on the system parameters.

Synchronous Segment = 1 TQ; Propagation Segment = 2 TQ; So setting Phase Segment 1 = 7 TQ would place the sample at 10 TQ after the transition. This would leave 6 TQ for Phase Segment 2.

Since Phase Segment 2 is 6, by the rules, the SJW1:SJW0 bits could be set to the maximum of 4 TQ. However, normally a large synchronization jump width is only necessary when the clock generation of the different nodes is inaccurate or unstable, such as using ceramic resonators. So a synchronization jump width of 1 is typically enough.

22.13 Interrupts

The module has several sources of interrupts. Each of these interrupts can be individually enabled or disabled. A PIR register contains interrupt flags. A PIE register contains the enables for the 8 main interrupts. A special set of read-only bits in the CANSTAT register (ICODE2:ICODE0) can be used in combination with a jump table for efficient handling of interrupts.

All interrupts have one source, with the exception of the Error Interrupt. Any of the Error Interrupt sources can set the Error Interrupt Flag. The source of the Error Interrupt can be determined by reading the Communication Status (COMSTAT) register.

The interrupts can be broken up into two categories: receive and transmit interrupts.

The receive related interrupts are:

- Receive Interrupt (see Section **22.9.7.1**)
- Wake-up Interrupt (see Section **22.9.7.2**)
- Receiver Overrun Interrupt (see Section **22.9.7.3.2**)
- Receiver Warning Interrupt (see Section **22.9.7.4**)
- Receiver Error Passive Interrupt (Section **22.9.7.5**)

The Transmit related interrupts are

- Transmit interrupt (see Section **22.10.11.1**)
- Transmitter Warning Interrupt (Section **22.10.11.3**)
- Transmitter Error Passive Interrupt (see Section **22.10.11.4**)
- Bus Off Interrupt (see Section **22.10.11.5**)

22.13.1 Interrupt Acknowledge

Interrupts are directly associated with one or more status flags in either a PIR or COMSTAT registers. Interrupts are pending as long as one of the corresponding flags is set. The flags in the registers must be reset within the interrupt handler in order to handshake the interrupt. A flag can not be cleared if the respective condition still prevails, with the exception being interrupts that are caused by a certain value being reached in one of the Error Counter Registers.

22.13.2 The ICODE Bits

The ICODE2:ICODE0 bits are a set of read-only bits designed for efficient handling of interrupts via a jump table. The ICODE2:ICODE0 bits can only display one interrupt at a time because the interrupt bits are multiplexed into this register. Therefore, the pending interrupt with the highest priority and enabled interrupt is reflected in the ICODE2:ICODE0 bits. Once the highest priority interrupt flag has been cleared, the next highest priority interrupt code is reflected in the ICODE2:ICODE0 bits. An interrupt code for a corresponding interrupt can only be displayed if both its interrupt flag and interrupt enable are set. Table 22-5 describes the operation of the ICODE2:ICODE0 bits.

Table 22-5: ICODE Bits Decode Table

ICODE2:ICODE0	Boolean Expression
000	$\overline{ERR}\bullet\overline{WAK}\bullet\overline{TX0}\bullet\overline{TX1}\bullet\overline{TX2}\bullet\overline{RX0}\bullet\overline{RX1}$
001	ERR
100	$\overline{ERR}\bullet TX0$
011	$\overline{ERR}\bullet\overline{TX0}\bullet TX1$
010	$\overline{ERR}\bullet\overline{TX0}\bullet\overline{TX1}\bullet TX2$
110	$\overline{ERR}\bullet\overline{TX0}\bullet\overline{TX1}\bullet\overline{TX2}\bullet RX0$
101	$\overline{ERR}\bullet\overline{TX0}\bullet\overline{TX1}\bullet\overline{TX2}\bullet\overline{RX0}\bullet RX1$
111	$\overline{ERR}\bullet\overline{TX0}\bullet\overline{TX1}\bullet\overline{TX2}\bullet\overline{RX0}\bullet\overline{RX1}\bullet WAK$

Legend: ERR = ERRIF • ERRIE
TX0 = TX0IF • TX0IE
TX1 = TX1IF • TX1IE
TX2 = TX2IF • TX2IE
RX0 = RX0IF • RX0IE
RX1 = RX1IF • RX1IE
WAK = WAKIF • WAKIE

22.14 Timestamping

The CAN module will generate a signal that can be selected to a timer capture input whenever a valid frame has been accepted. Because the CAN specification defines a frame to be valid if no errors occurred before the EOF field has been transmitted successfully, the timer signal will be generated right after the EOF. A pulse of one bit time is generated.

22.15 CAN Module I/O

The CAN bus module communicates on up to 3 I/O pins. There are 1 or 2 transmit pins and 1 receive pin. These pins are multiplexed with normal digital I/O functions of the device. The CIOCON register controls the functions of the I/O pins.

When the module is in the configuration mode, module disable mode or loopback mode, the I/O pins revert to a Port I/O function.

When the module is active, the TX0 pin is always dedicated to the CAN output function. If a single ended driver is needed, then only the TX0 pin is required. If a differential driver is required, then the TX1 pin must be enabled by setting the TX1EN bit. If the bus requires an active pull-up on the line, the ENDRHI bit should be cleared.

The TRIS bits associated with the transmit pins are overridden by the CAN bus modes. If the CAN module expects an output to be driving, it will be regardless of the state of the TRIS bit associated with that pin.

The output buffers for the TX0 and TX1 pin are designed such that the rise and fall rate of the output signal is approximately equal as is necessary for differential drive.

The module can receive the CAN input on one digital input line.

22.16 Design Tips

Question 1: ***My CAN module does not seem to work after a RESET.***

Answer 1:

Ensure that you reinitialize your CAN bus module. After a RESET, the CAN bus module will automatically go into the initialization mode.

Question 2: ***I constantly get a Receive error warning interrupt.***

Answer 2:

Ensure that your CAN module is set up correctly. Check if the Baud rate is set correctly.

22.17 Related Application Notes

This subsection lists application notes that are related to this subsection of the manual. These application notes may not be written for the Mid-range family (that is they may be written for the Baseline, or the High-end), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to this section are:

Title	Application Note #
An Introduction to the CAN Protocol	AN713

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

22.18 Revision History

Revision A

This is the initial released revision of this document.

Section 23. Comparator Voltage Reference

HIGHLIGHTS

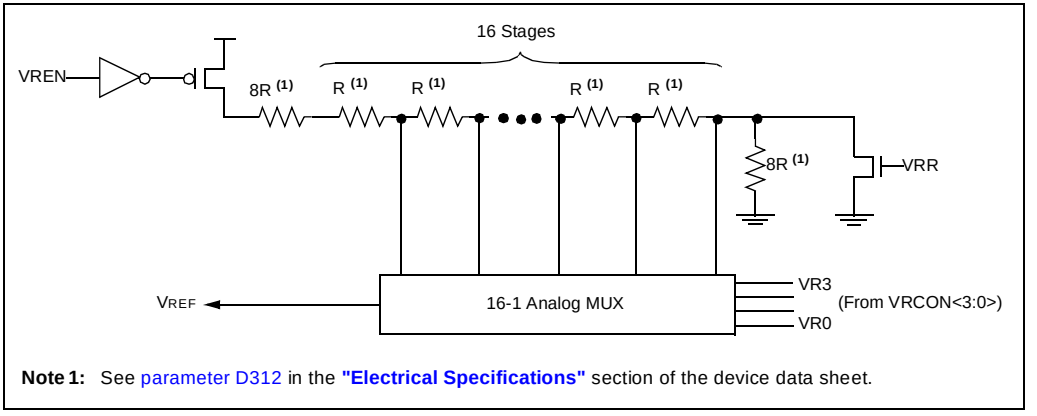
This section of the manual contains the following major topics:

23.1 Introduction

This Voltage Reference module is typically used in conjunction with the Comparator module. The Comparator module's inputs do not require very large drive, and therefore the drive capability of this Voltage Reference is limited.

The Voltage Reference is a 16-tap resistor ladder network that provides a selectable Voltage Reference. The resistor ladder is segmented to provide two ranges of VREF values and has a power-down function to conserve power when the reference is not being used. The VRCON register controls the operation of the reference (shown in Register 23-1). The block diagram is given in Figure 23-1. Within each range, the 16 steps are monotonic (i.e., each increasing code will result in an increasing output).

Figure 23-1: Voltage Reference Block Diagram



Note 1: See parameter D312 in the **"Electrical Specifications"** section of the device data sheet.

Table 23-1: Typical Voltage Reference with VDD = 5.0V

VR3:VR0	VREF	
	VRR = 1	VRR = 0
0000	0.00 V	1.25 V
0001	0.21 V	1.41 V
0010	0.42 V	1.56 V
0011	0.63 V	1.72 V
0100	0.83 V	1.88 V
0101	1.04 V	2.03 V
0110	1.25 V	2.19 V
0111	1.46 V	2.34 V
1000	1.67 V	2.50 V
1001	1.88 V	2.66 V
1010	2.08 V	2.81 V
1011	2.29 V	2.97 V
1100	2.50 V	3.13 V
1101	2.71 V	3.28 V
1110	2.92 V	3.44 V
1111	3.13 V	3.59 V

23.2 Control Register

The Voltage Reference Control register (VRCON) is shown in Register 23-1.

Register 23-1: VRCON Register

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
VREN	VROEN	VRR	—	VR3	VR2	VR1	VR0
bit 7							bit 0

bit 7 **VREN:** VREF Enable
1 = VREF circuit powered on
0 = VREF circuit powered down

bit 6 **VROEN:** VREF Output Enable
1 = VREF is internally connected to Comparator module's VREF. This voltage level is also output on the VREF pin
0 = VREF is not connected to the Comparator module. This voltage is disconnected from the VREF pin

bit 5 **VRR:** VREF Range Selection
1 = 0V to 0.75 VDD, with VDD/24 step size
0 = 0.25 VDD to 0.75 VDD, with VDD/32 step size

bit 4 **Unimplemented:** Read as '0'

bit 3:0 **VR3:VR0:** VREF Value Selection 0 ≤ VR3:VR0 ≤ 15

When VRR = 1:

$V_{REF} = (VR<3:0> / 24) \bullet V_{DD}$

When VRR = 0:

$V_{REF} = 1/4 * V_{DD} + (VR3:VR0 / 32) \bullet V_{DD}$

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

23.3 Configuring the Voltage Reference

The Voltage Reference can output 16 distinct voltage levels for each range.

The equations used to calculate the output of the Voltage Reference are as follows:

if VRR = 1: $V_{REF} = (VR3:VR0 / 24) \times V_{DD}$

if VRR = 0: $V_{REF} = (V_{DD} \times 1/4) + (VR3:VR0 / 32) \times V_{DD}$

The settling time of the Voltage Reference must be considered when changing the V_{REF} output. Example 23-1 shows an example of how to configure the Voltage Reference for an output voltage of 1.25V with V_{DD} = 5.0V.

Generally the V_{REF} and V_{DD} of the system will be known and you need to determine the value to load into VR3:VR0. Equation 23-1 shows how to calculate the VR3:VR0 value. There will be some error since VR3:VR0 can only be an integer, and the V_{REF} and V_{DD} levels must be chosen so that the result is not greater then 15.

Equation 23-1: Calculating VR3:VR0

When VRR = 1

$$VR3:VR0 = \frac{V_{REF}}{V_{DD}} \quad X\ 24$$

When VRR = 0

$$VR3:VR0 = \frac{V_{REF} - V_{DD}/4}{V_{DD}} \quad X\ 32$$

23.4 Voltage Reference Accuracy/Error

The full range of VSS to VDD cannot be realized due to the construction of the module. The transistors on the top and bottom of the resistor ladder network (Figure 23-1) keep VREF from approaching VSS or VDD. The Voltage Reference is VDD derived and therefore, the VREF output changes with fluctuations in VDD. The absolute accuracy of the Voltage Reference can be found in the Electrical Specifications parameter D311.

23.5 Operation During SLEEP

When the device wakes up from SLEEP through an interrupt or a Watchdog Timer time-out, the contents of the VRCON register are not affected. To minimize current consumption in SLEEP mode, the Voltage Reference should be disabled.

23.6 Effects of a RESET

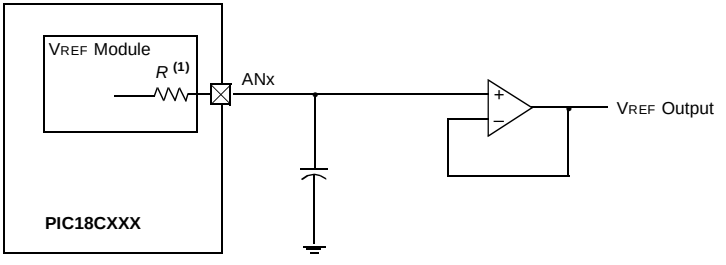
A device RESET disables the Voltage Reference by clearing the VREN bit (VRCON<7>). This RESET also disconnects the reference from the VREF pin by clearing the VROEN bit (VRCON<6>) and selects the high voltage range by clearing the VRR bit (VRCON<5>). The VREF value select bits, VRCON<3:0>, are also cleared.

23.7 Connection Considerations

The Voltage Reference Module operates independently of the Comparator module. The output of the reference generator may be connected to the VREF pin if the corresponding TRIS bit is set and the VROEN bit (VRCON<6>) is set. Enabling the Voltage Reference output onto the VREF pin with an input signal present will increase current consumption. Configuring the VREF as a digital output with VREF enabled will also increase current consumption.

The VREF pin can be used as a simple D/A output with limited drive capability. Due to the limited drive capability, a buffer must be used in conjunction with the Voltage Reference output for external connections to VREF. Figure 23-2 shows an example buffering technique.

Figure 23-2: Voltage Reference Output Buffer Example



Note 1: *R* is the Voltage Reference Output Impedance and is dependent upon the Voltage Reference Configuration (the VR3:VR0 bits and the VRR bit).

23.8 Initialization

Example 23-1 shows a program sequence to configure the Voltage Reference, comparator module, and PORT pins.

Example 23-1: Voltage Reference Configuration

```
MOVLW    0x02         ; 4 Inputs Muxed to 2 comparators
MOVWF    CMCON        ;
MOVLW    PORTxout     ; Select PORTx pins
MOVWF    TRISx        ;   to be output
MOVLW    0xA6         ; enable VREF
MOVWF    VRCON        ; low range set VR3:VR0 = 6
CALL     DELAY10      ; 10 µs delay
```

23.9 Design Tips

Question 1: ***My VREF is not what I expect.***

Answer 1:

Any variation of the device VDD will translate directly onto the VREF pin. Also ensure that you have correctly calculated (specified) the VDD divider which generates the VREF.

Question 2: ***I am connecting VREF into a low impedance circuit, and the VREF is not at the expected level.***

Answer 2:

The Voltage Reference module is not intended to drive large loads. A buffer must be used between the PICmicro's VREF pin and the load.

23.10 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced MCU family (that is they may be written for the Base-Line, Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to Voltage Reference are:

Title	Application Note #
Resistance and Capacitance Meter using a PIC16C622	AN611

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

23.11 Revision History

Revision A

This is the initial released revision of the Voltage Reference description.

Section 24. Comparator

HIGHLIGHTS

This section of the manual contains the following major topics:

24.1 Introduction

The comparator module contains two analog comparators. The inputs to the comparators are multiplexed with the I/O pins. The on-chip Voltage Reference (see the **"Comparator Voltage Reference"** section) can also be an input to the comparators.

The CMCON register, shown in Register 24-1, controls the comparator input and output multiplexers. A block diagram of the comparator is shown in Figure 24-1.

24.2 Control Register

The Comparator Control register (CMCON) is shown in Register 24-1.

Register 24-1: CMCON Register

R-0	R-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7							bit 0

bit 7 **C2OUT**: Comparator2 Output State bit
This bit indicates the output state of comparator 2.
1 = C2 V_{IN+} > C2 V_{IN-}
0 = C2 V_{IN+} < C2 V_{IN-}

bit 6 **C1OUT**: Comparator1 Output State bit
This bit indicates the output state of comparator 1.
1 = C1 V_{IN+} > C1 V_{IN-}
0 = C1 V_{IN+} < C1 V_{IN-}

bit 5 **C2INV:** Comparator2 Inverted Output State bit
1 = Invert the state of C2 output
0 = State of C2 output is not inverted

bit 4 **C1INV:** Comparator1 Inverted Output State bit
1 = Invert the state of C1 output
0 = State of C1 output is not inverted

bit 3 **CIS**: Comparator Input Switch bit
This bit selects which analog inputs are used as the input to the comparator.

When CM2:CM0: = 001:

1 = C1 V_{IN-} connects to ANx_3
0 = C1 V_{IN-} connects to ANx_0

When CM2:CM0 = 010:

1 = C1 V_{IN-} connects to ANx_3
C2 V_{IN-} connects to ANx_2
0 = C1 V_{IN-} connects to ANx_0
C2 V_{IN-} connects to ANx_1

bit 2:0 **CM2:CM0**: Comparator Mode Select bits
This bit selects the configuration of the two comparators with the comparator input pins and the **"Comparator Voltage Reference"**.
See Figure 24-1 to select the CM2:CM0 state for the desired mode. The use of ANx_0 through ANx_3 indicates that there are four analog inputs used with the comparator module. The actual analog inputs connected to the comparator inputs will be device dependent.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

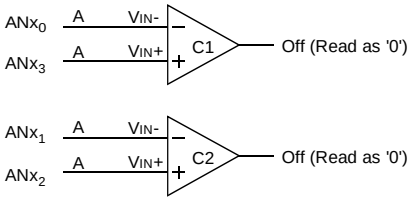
24.3 Comparator Configuration

There are eight modes of operation for the comparators. The CMCON register is used to select the mode. Figure 24-1 shows the eight possible modes. The TRIS register controls the data direction of the comparator I/O pins for each mode. If the comparator mode is changed, the comparator output level may not be valid for the new mode for the delay specified in the electrical specifications of the device.

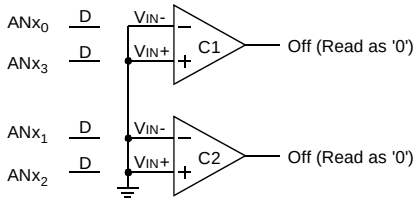
Note:	Comparator interrupts should be disabled during a comparator mode change, otherwise a false interrupt may occur.

Figure 24-1: Comparator I/O Operating Modes

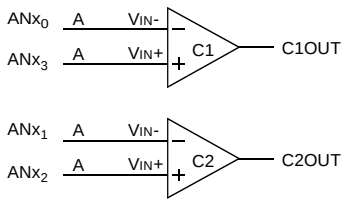
CM2:CM0 = 000
Comparators Reset (POR Default Value)



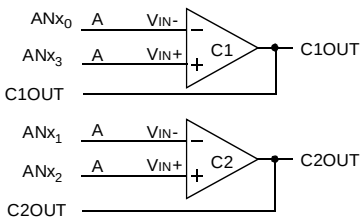
CM2:CM0 = 111
Comparators Off



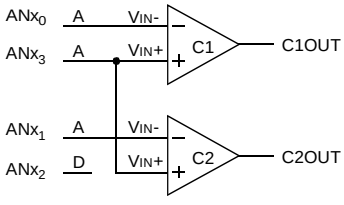
CM2:CM0 = 010
Two Independent Comparators



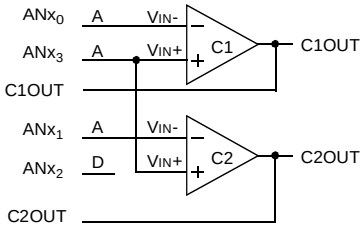
CM2:CM0 = 011
Three Inputs Multiplexed to Two Comparators



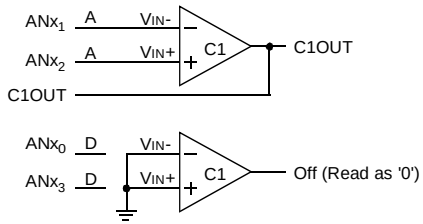
CM2:CM0 = 100
Two Common Reference Comparators



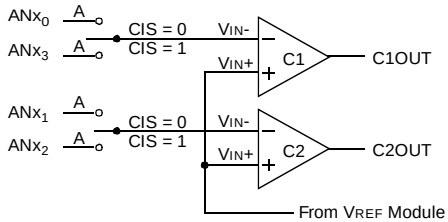
CM2:CM0 = 101
Two Common Reference Comparators with Outputs



CM2:CM0 = 001
One Independent Comparator



CM2:CM0 = 110
Four Inputs Multiplexed to Two Comparators



A = Analog Input, port reads as zeros always.
D = Digital Input.
CIS (CMCON<3>) is the Comparator Input Switch.

24

Comparator

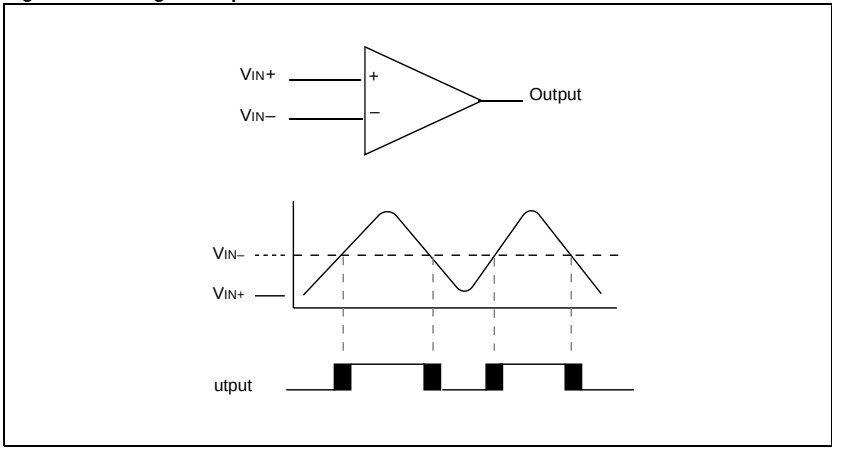
24.4 Comparator Operation

A single comparator is shown in Figure 24-2 along with the relationship between the analog input levels and the digital output. When the analog input at VIN+ is less than the analog input VIN–, the output of the comparator is a digital low level. When the analog input at VIN+ is greater than the analog input VIN–, the output of the comparator is a digital high level. The shaded areas of the output of the comparator (shown in Figure 24-2) represent the uncertainty due to input offsets and response time.

24.5 Comparator Reference

An external or internal reference signal may be used depending on the comparator operating mode. The analog signal that is present at VIN– is compared to the signal at VIN+, and the digital output of the comparator is adjusted accordingly (Figure 24-2).

Figure 24-2: Single Comparator



24.5.1 External Reference Signal

When external voltage references are used, the comparator module can be configured to have the comparators operate from the same or different reference sources. The reference signal must be between VSS and VDD, and can be applied to either pin of the comparator(s).

24.5.2 Internal Reference Signal

The comparator module also allows the selection of an internally generated voltage reference for the comparators. The **"Comparator Voltage Reference"** section contains a detailed description of the Voltage Reference Module that provides this signal. The internal reference signal is used when the comparators are in mode CM2:CM0 = 110 (Figure 24-1). In this mode, the internal voltage reference is applied to the VIN+ input of both comparators.

The internal voltage reference may be used in any comparator mode. The voltage reference is output to the VREF pin. Any comparator input pin may be connected externally to the VREF pin.

24.6 Comparator Response Time

Response time is the minimum time, after selecting a new reference voltage or input source, before the comparator output is guaranteed to have a valid level. If the internal reference is changed, the maximum settling time of the internal voltage reference must be considered when using the comparator outputs. Otherwise the maximum response time of the comparators should be used.

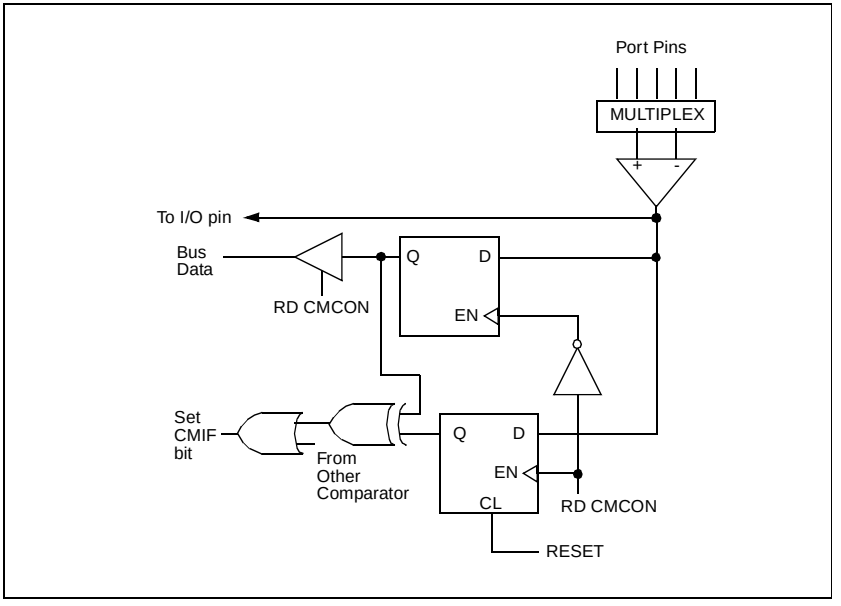
24.7 Comparator Outputs

The comparator outputs are read through the CMCON register. These bits are read only. The comparator outputs may also be directly output to the I/O pins. When CM2:CM0 = 011, multiplexors in the output path of the I/O pins will switch and the output of each pin will be the unsynchronized output of the comparator. The uncertainty of each of the comparators is related to the input offset voltage and the response time given in the specifications. Figure 24-3 shows the comparator output block diagram.

The TRIS bits will still function as the output enable/disable for the I/O pins while in this mode.

Note 1:	When reading the Port register, all pins configured as analog inputs will read as a '0'. Pins configured as digital inputs will convert an analog input according to the Schmitt Trigger input specification.
2:	Analog levels on any pin that is defined as a digital input may cause the input buffer to consume more current than is specified.

Figure 24-3: Comparator Output Block Diagram



24.8 Comparator Interrupts

The comparator interrupt flag is set whenever the comparators value changes relative to the last value loaded into CMxOUT bits. Software will need to maintain information about the status of the output bits, as read from CMCON<7:6>, to determine the actual change that has occurred. The CMIF bit is the comparator interrupt flag. The CMIF bit must be cleared. Since it is also possible to set this bit, a simulated interrupt may be initiated.

The CMIE bit, the PEIE/GIEL bit, and the GIE/GIEH bit must be set to enable the interrupt. If any of these bits are clear, an interrupt from the comparator module will not occur, though the CMIF bit will still be set if an interrupt condition occurs. Table 24-1 shows the state of the comparator interrupt bits to enable an interrupt to vector to the interrupt vector address. If these conditions are not met, the comparator module will set the CMIF bit, but the program execution will not go to the interrupt vector address.

The user, in the interrupt service routine, can clear the interrupt in the following manner:

a) Any read or write of the CMCON register. This will load the CMCON register with the new value with the CMxOUT bits.

b) Clear the CMIF flag bit.

An interrupt condition will continue to set the CMIF flag bit. Reading CMCON will end the interrupt condition, and allow the CMIF flag bit to be cleared.

Table 24-1: How State of Interrupt Control Bits Determine Action After Comparator Trip (CMIF is Set)

GIE	GIEH	PEIE	GIEL	CMIE	IPEN	CMIP	Comment
1	—	1	—	1	0	—	CMIF set Branch to ISR
x	—	x	—	0	0	—	CMIF set
x	—	0	—	x	0	—	CMIF set
0	—	x	—	x	0	—	CMIF set
—	x	—	1	1	1	0	CMIF set Branch to ISR
—	1	—	x	1	1	1	CMIF set Branch to ISR
—	x	—	x	0	1	x	CMIF set
—	x	—	0	x	1	0	CMIF set
—	x	0	—	x	1	1	CMIF set
—	0	x	—	x	1	1	CMIF set

24.9 Comparator Operation During SLEEP

When a comparator is active and the device is placed in SLEEP mode, the comparator remains active and the interrupt is functional if enabled. This interrupt will wake-up the device from SLEEP mode when enabled. While the comparator is powered up, each comparator that is operational will consume additional current as shown in the comparator specifications. To minimize power consumption while in SLEEP mode, turn off the comparators (CM2:CM0 = 111), before entering SLEEP. If the device wakes up from SLEEP, the contents of the CMCON register are not affected.

24.10 Effects of a RESET

A device RESET forces the CMCON register to its reset state. This forces the comparator module to be in the comparator reset mode, CM2:CM0 = 000. This ensures that all potential inputs are analog inputs. Device current is minimized when analog inputs are present at RESET time. The comparators will be powered down disabled during the RESET interval.

24.11 Analog Input Connection Considerations

A simplified circuit for an analog input is shown in Figure 24-4. Since the analog pins are connected to a digital output, they have reverse biased diodes to VDD and VSS. The analog input therefore, must be between VSS and VDD. If the input voltage deviates from this range by more than 0.6V in either direction, one of the diodes is forward biased and a latch-up may occur. A maximum source impedance of 10 kΩ is recommended for the analog sources.

Figure 24-4: Analog Input Model

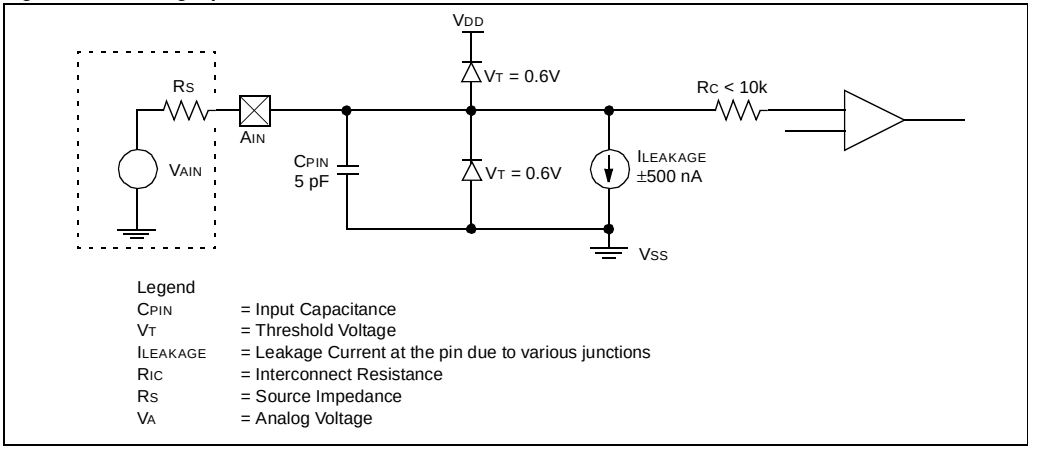


Table 24-2: Registers Associated with Comparator Module

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other Resets
CMCON	C2OUT	C1OUT	—	—	CIS	CM2	CM1	CM0	00-- 0000	00-- 0000
VRCON	VREN	VROE	VRR	—	VR3	VR2	VR1	VR0	000- 0000	000- 0000
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000x
PIR	CMIF (1)								0	0
PIE	CMIE (1)								0	0
IPE	CMIP (1)								0	0

Legend: x = unknown, - = unimplemented locations read as '0'.
Shaded cells are not used for Comparator Module.

Note 1: The position of this bit is device dependent.

24.12 Initialization

The code in Example 24-1 depicts example steps required to configure the comparator module. The Port registers (PORTx, LATx, and TRISx) need to be configured appropriately depending on the mode selected. For CM2:CM0 = 100, the I/O multiplied with ANx0, ANx1, and ANx2 needs to be configured for analog inputs. Other I/O may be digital.

Example 24-1: Initializing Comparator Module

```
FLAG_REG   EQU    0x020
;
   CLRF    FLAG_REG      ; Init flag register
   CLRF    PORTx         ; Init the desired port
   MOVF    CMCON, W      ;
   ANDLW   0xC0          ; Mask comparator bits
   IORWF   FLAG_REG,F    ; Store bits in flag register
   MOVLW   0x04          ; Init comparator mode
   MOVWF   CMCON         ; CM<2:0> = 100
   MOVLW   PORTxDIR      ; Initialize data direction of the ANx0, ANx1,
   MOVWF   TRISx         ;   and ANx2. Set as inputs, other I/O
                         ;   on port as desired (either inputs or outputs)
   CALL    DELAY10       ; 10us delay
   MOVF    CMCON, F      ; Read  CMCON to end change condition
   BCF     PIR1,CMIF     ; Clear pending interrupts
   BSF     PIE1,CMIE     ; Enable comparator interrupts
   BSF     INTCON,PEIE   ; Enable peripheral interrupts
   BSF     INTCON,GIE    ; Global interrupt enable
```

24.13 Design Tips

Question 1: ***My program appears to lock up.***

Answer 1:

You may be getting stuck in an infinite loop with the comparator interrupt service routine if you did not follow the proper sequence to clear the CMIF flag bit. First, you must read the CMCON register and then you can clear the CMIF flag bit.

24.14 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced MCU family (that is they may be written for the Base-Line, Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the comparator module are:

Title	Application Note #
Resistance and Capacitance Meter using a PIC16C622	AN611

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

24.15 Revision History

Revision A

This is the initial released revision of the Comparator module description.

Section 25. Compatible 10-bit A/D Converter

HIGHLIGHTS

This section of the manual contains the following major topics:

25.1 Introduction

The compatible analog-to-digital (A/D) converter module is software compatible with the Standard 10-bit A/D converter and can have up to sixteen analog inputs.

The analog input charges a sample and hold capacitor. The output of the sample and hold capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation. This A/D conversion of the analog input signal results in a corresponding 10-bit digital number.

The analog reference voltages (positive and negative supply) are software selectable to either the device's supply voltages (AV_{DD}, AV_{SS}) or the voltage level on the AN3/V_{REF+} and AN2/V_{REF-} pins.

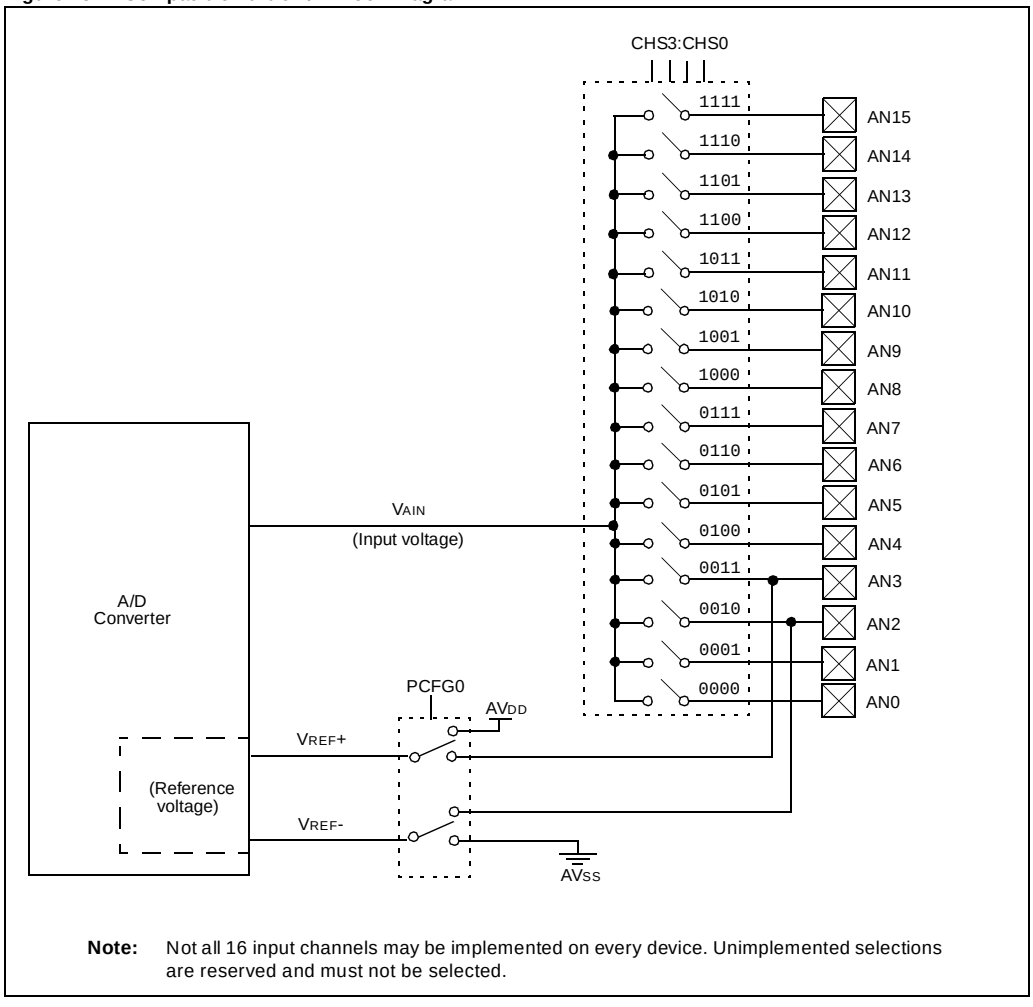
The A/D converter has the unique feature of being able to convert while the device is in SLEEP mode.

The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)

The ADCON0 register, shown in Register 25-1, controls the operation of the A/D module. The ADCON1 register, shown in Register 25-2, configures the functions of the port pins. The port pins can be configured as analog inputs (AN3 and AN2 can also be the voltage references) or as digital I/O.

Figure 25-1: Compatible 10-bit A/D Block Diagram



Note: Not all 16 input channels may be implemented on every device. Unimplemented selections are reserved and must not be selected.

25.2 Control Register

ADCON0 (Register 25-1) is used to select the clock and the analog channel. ADCON1 (Register 25-2) configures the port logic to either analog or digital inputs and the format of the result.

Register 25-1: ADCON0 Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	CHS3	ADON
bit 7							bit 0

bit 7-6: **ADCS1:ADCS0:** A/D Conversion Clock Select bits (shown in bold)

Three bits are required to select the A/D clock source. These bits are ADCS2:ADCS0.

0**00** = FOSC/2
0**01** = FOSC/8
0**10** = FOSC/32
0**11** = FRC (clock derived from the internal A/D RC oscillator)
1**00** = FOSC/4
1**01** = FOSC/16
1**10** = FOSC/64
1**11** = FRC (clock derived from the internal A/D RC oscillator)

Note: The ADCS2 bit is located in the ADCON1 register.

bit 5-3: **CHS2:CHS0:** Analog Channel Select bits

There are four bits that select the A/D channel. These are CHS3:CHS0.

0000 = channel 0, (AN0)
0001 = channel 1, (AN1)
0010 = channel 2, (AN2)
0011 = channel 3, (AN3)
0100 = channel 4, (AN4)
0101 = channel 5, (AN5)
0110 = channel 6, (AN6)
0111 = channel 7, (AN7)
1000 = channel 8, (AN8)
1001 = channel 8, (AN9)
1010 = channel 10, (AN10)
1011 = channel 11, (AN11)
1100 = channel 12, (AN12)
1101 = channel 13, (AN13)
1110 = channel 14, (AN14)
1111 = channel 15, (AN15)

Note: For devices that do not implement the full 16 A/D channels, the unimplemented selections are reserved. Do not select any unimplemented channel.

bit 2: **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit

When ADON = 1

1 = A/D conversion in progress. Setting this bit starts an A/D conversion cycle.
This bit is automatically cleared by hardware when the A/D conversion is completed.

0 = A/D conversion not in progress

bit 1: **CHS3:** Analog Channel Select bit

The CHS2:CHS0 bits are located in positions bit 5 to bit 3. See the CHS2:CHS0 description for operational details.

bit 0: **ADON:** A/D On bit

1 = A/D converter module is powered up
0 = A/D converter module is shut off and consumes no operating current

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 25-2: ADCON1 Register

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7: **ADFM:** A/D Result Format Select (also see Figure 25-6)

1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.
0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.

bit 6: **ADCS2:** A/D Conversion Clock Select bits (shown in bold)

Three bits are required to select the A/D clock source. These bits are ADCS2:ADCS0.

000 = FOSC/2
001 = FOSC/8
010 = FOSC/32
011 = FRC (clock derived from the internal A/D RC oscillator)
100 = FOSC/4
101 = FOSC/16
110 = FOSC/64
111 = FRC (clock derived from the internal A/D RC oscillator)

Note: The ADCS1:ADCS0 bits are located in the ADCON0 register.

bit 5-4: **Unimplemented:** Read as '0'

bit 3-0: **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	CH/REF
0000	A	A	A	A	A	A	A	A	AVDD	AVSS	8 / 0
0001	A	A	A	A	VREF+	A	A	A	AN3	AVSS	7 / 1
0010	D	D	D	A	A	A	A	A	AVDD	AVSS	5 / 0
0011	D	D	D	A	VREF+	A	A	A	AN3	AVSS	4 / 1
0100	D	D	D	D	A	D	A	A	AVDD	AVSS	3 / 0
0101	D	D	D	D	VREF+	D	A	A	AN3	AVSS	2 / 1
011x	D	D	D	D	D	D	D	D	—	—	0 / 0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6 / 2
1001	D	D	A	A	A	A	A	A	AVDD	AVSS	6 / 0
1010	D	D	A	A	VREF+	A	A	A	AN3	AVSS	5 / 1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4 / 2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3 / 2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2 / 2
1110	D	D	D	D	D	D	D	A	AVDD	AVSS	1 / 0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1 / 2

A = Analog input D = Digital I/O
Ch/Ref = # of analog input channels / # of A/D voltage references

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Note 1: On any device RESET, the port pins that are multiplexed with analog functions (ANx) are forced to be an analog input.

25.3 Operation

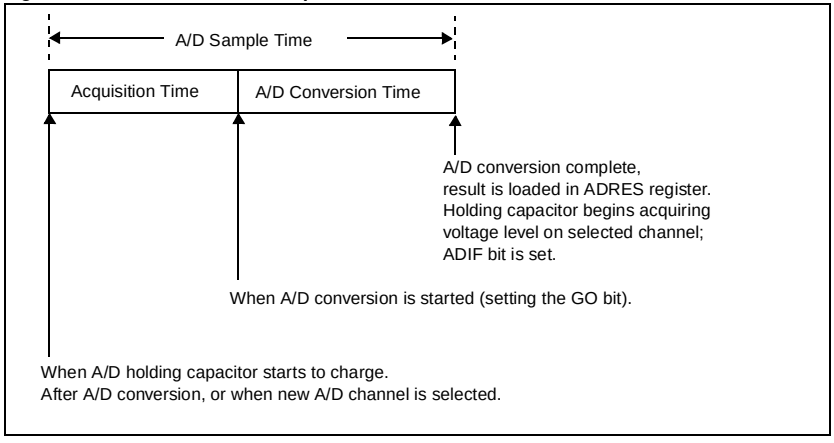
The ADRESH:ADRESL registers contain the 10-bit result of the A/D conversion. When the A/D conversion is complete, the result is loaded into this A/D result register pair (ADRESH:ADRESL), the GO/$\overline{\text{DONE}}$ bit (ADCON0) is cleared, and A/D interrupt flag bit, ADIF, is set. The block diagram of the A/D module is shown in Figure 25-1.

After the A/D module has been configured, the signal on the selected channel must be acquired before the conversion is started. The analog input channels must have their corresponding TRIS bits selected as inputs. To determine acquisition time, see Subsection **25.4 "A/D Acquisition Requirements."** After this acquisition time has elapsed, the A/D conversion can be started. The following steps should be followed for doing an A/D conversion:

1. Configure the A/D module:
   - Configure analog pins, voltage reference, and digital I/O (ADCON1)
   - Select A/D input channel (ADCON0)
   - Select A/D conversion clock (ADCON0)
   - Turn on A/D module (ADCON0)
2. Configure A/D interrupt (if desired):
   - Clear the ADIF bit
   - Set the ADIE bit
   - Set/Clear the ADIP bit
   - Set the GIE/GIEH or PEIE/GIEL bit
3. Wait the required acquisition time.
4. Start conversion:
   - Set the GO/$\overline{\text{DONE}}$ bit (ADCON0)
5. Wait for the A/D conversion to complete, by either:
   - Polling for the GO/$\overline{\text{DONE}}$ bit to be cleared or the ADIF bit to be set, or
   - Waiting for the A/D interrupt
6. Read A/D Result register pair (ADRESH:ADRESL): clear the ADIF bit, if required.
7. For next conversion, go to step 1 or step 2 as required.

Figure 25-2 shows the conversion sequence and the terms that are used. Acquisition time is the time that the A/D module's holding capacitor is connected to the external voltage level. When the GO bit is set, the conversion time of 12 T_{AD} is started. The sum of these two times is the sampling time. There is a minimum acquisition time to ensure that the holding capacitor is charged to a level that will give the desired accuracy for the A/D conversion.

Figure 25-2: A/D Conversion Sequence



25.4 A/D Acquisition Requirements

For the A/D converter to meet its specified accuracy, the charge holding capacitor (CHOLD) must be allowed to fully charge to the input channel voltage level. The analog input model is shown in Figure 25-3. The source impedance (RS) and the internal sampling switch (RSS) impedance directly affect the time required to charge the capacitor CHOLD. The sampling switch (RSS) impedance varies over the device voltage (VDD), Figure 25-3. The source impedance affects the offset voltage at the analog input (due to pin leakage current). **The maximum recommended impedance for analog sources is 2.5 kΩ.** As the impedance is decreased, the acquisition time may be decreased. After the analog input channel is selected (changed), this acquisition must pass before the conversion can be started.

To calculate the minimum acquisition time, Equation 25-1 may be used. This equation assumes that 1/2 LSb error is used (1024 steps for the A/D). The 1/2 LSb error is the maximum error allowed for the A/D to meet its specified resolution.

Note:	When the conversion is started, the holding compacitor is disconnected from the input pin.

Equation 25-1: Acquisition Time

TACQ	equals	Amplifier Settling Time (TAMP) plus Holding Capacitor Charging Time (TC) plus Temperature Coefficient (TCOFF)
TACQ	=	TAMP + TC + TCOFF

Equation 25-2: A/D Minimum Charging Time

$$V_{HOLD} = (V_{REF} - (V_{REF}/2048)) \cdot (1 - e^{(-T_C/C_{HOLD}(R_{IC} + R_{SS} + R_S))})$$

or

$$T_C = -(120\ pF)(1\ k\Omega + R_{SS} + R_S)\ \ln(1/2047)$$

Example 25-1 shows the calculation of the minimum required acquisition time TACQ. This calculation is based on the following application system assumptions.

CHOLD	=	120 pF	
RS	=	2.5 kΩ	
Conversion Error	≤	1/2 LSb	
VDD	=	5V → RSS = 7 kΩ	(see graph in Figure 25-3)
Temperature	=	50°C (system max.)	
VHOLD	=	0V @ time = 0	

Example 25-1: Calculating the Minimum Required Acquisition Time (Case 1)

TACQ = TAMP + TC + TCOFF

Temperature coefficient is only required for temperatures > 25°C.

TACQ = 2 µs + TC + [(Temp - 25°C)(0.05 µs/°C)]

TC = -CHOLD (RIC + RSS + RS) ln(1/2047)
-120 pF (1 kΩ + 7 kΩ + 2.5 kΩ) ln(0.0004885)
-120 pF (10.5 kΩ) ln(0.0004885)
-1.26 µs (-7.6241)
9.61 µs

TACQ = 2 µs + 9.61µs + [(50°C - 25°C)(0.05 µs/°C)]
11.61 µs + 1.25 µs
12.86 µs

Now to get an idea what happens to the acquisition time when the source impedance is a minimal value (RS = 50 Ω). Example 25-2 shows the same conditions as in Example 25-1 with only the source impedance changed to the minimal value.

Example 25-2: Calculating the Minimum Required Acquisition Time (Case 2)

TACQ = TAMP + TC + TCOFF

Temperature coefficient is only required for temperatures > 25°C.

TACQ = 2 µs + Tc + [(Temp - 25°C)(0.05 µs/°C)]

TC = -Chold (Ric + Rss + Rs) ln(1/2047)
-120 pF (1 kΩ + 7 kΩ + 50 Ω) ln(0.0004885)
-120 pF (8050 Ω) ln(0.0004885)
-0.966 µs (-7.6241)
7.36 µs

TACQ = 2 µs + 16.47 µs + [(50°C - 25°C)(0.05 µs/°C)]
9.36 µs + 1.25 µs
10.61 µs

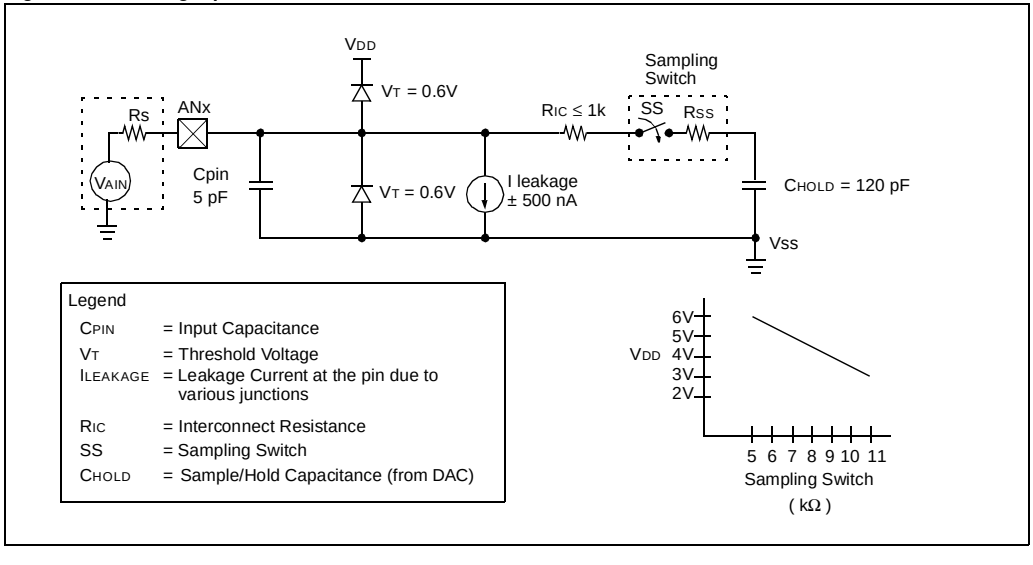
Note 1: The reference voltage (VREF) has no effect on the equation, since it cancels itself out.

2: The charge holding capacitor (Chold) is not discharged after each conversion.

3: The maximum recommended impedance for analog sources is 2.5 kΩ. This is required to meet the pin leakage specification.

4: After a conversion has completed, a 2 TAD delay must complete before acquisition can begin again. During this time the holding capacitor is not connected to the selected A/D input channel.

Figure 25-3: Analog Input Model



25.5 Selecting the A/D Conversion Clock

The A/D conversion time per bit is defined as TAD. The A/D conversion requires 11.5TAD per 10-bit conversion. The source of the A/D conversion clock is software selectable. The seven possible options for TAD are:

- 2TOSC
- 4TOSC
- 8TOSC
- 16TOSC
- 32TOSC
- 64TOSC
- Internal A/D RC oscillator

For correct A/D conversions, the A/D conversion clock (TAD) must be selected to ensure a minimum TAD time of 1.6 µs as shown in Electrical Specifications parameter 130.

Table 25-1 and Table 25-2 show the resultant TAD times derived from the device operating frequencies and the selected A/D clock source.

Table 25-1: TAD vs. Device Operating Frequencies (for Standard, C, Devices)

AD Clock Source (TAD)		Device Frequency			
Operation	**ADCS2:ADCS0**	**20 MHz**	**5 MHz**	**1.25 MHz**	**333.33 kHz**
2TOSC	000	100 ns (2)	400 ns (2)	1.6 µs	6 µs
4TOSC	100	200 ns (2)	800 ns (2)	3.2 µs	12 µs
8TOSC	001	400 ns (2)	1.6 µs	6.4 µs	24 µs (3)
16TOSC	101	800 ns (2)	3.2 µs	12.8 µs	48 µs (3)
32TOSC	010	1.6 µs	6.4 µs	25.6 µs (3)	96 µs (3)
64TOSC	110	3.2 µs	12.8 µs	51.2 µs (3)	192 µs (3)
RC	011	2 - 6 µs (1,4)	2 - 6 µs (1,4)	2 - 6 µs (1,4)	2 - 6 µs (1)

Legend: Shaded cells are outside of recommended range.

Note 1: The RC source has a typical TAD of 4 µs.

2: These values violate the minimum required TAD.

3: For faster conversion times, the selection of another clock source is recommended.

4: For device frequencies above 1 MHz, the device must be in SLEEP for the entire conversion, or the A/D accuracy may be out of specification.

Table 25-2: TAD vs. Device Operating Frequencies (for Extended, LC, Devices)

AD Clock Source (TAD)		Device Frequency			
Operation	**ADCS2:ADCS0**	**4 MHz**	**2 MHz**	**1.25 MHz**	**333.33 kHz**
2TOSC	000	500 ns (2)	1.0 µs (2)	1.6 µs (2)	6 µs
4TOSC	100	1.0 µs (2)	2.0 µs (2)	3.2 µs (2)	12 µs
8TOSC	001	2.0 µs (2)	4.0 µs	6.4 µs	24 µs (3)
16TOSC	101	4.0 µs (2)	8.0 µs	12.8 µs	48 µs (3)
32TOSC	010	8.0 µs	16.0 µs	25.6 µs (3)	96 µs (3)
64TOSC	110	16.0 µs	32.0 µs	51.2 µs (3)	192 µs (3)
RC	011	3 - 9 µs (1,4)	3 - 9 µs (1,4)	3 - 9 µs (1,4)	3 - 9 µs (1)

Legend: Shaded cells are outside of recommended range.

Note 1: The RC source has a typical TAD of 6 µs.

2: These values violate the minimum required TAD.

3: For faster conversion times, the selection of another clock source is recommended.

4: For device frequencies above 1 MHz, the device must be in SLEEP for the entire conversion, or the A/D accuracy may be out of specification.

25.6 Configuring Analog Port Pins

The ADCON1 and TRIS registers control the operation of the A/D port pins. The port pins that are desired as analog inputs must have their corresponding TRIS bits set (input). If the TRIS bit is cleared (output), the digital output level (V_{OH} or V_{OL}) will be converted. After a device RESET, pins that are multiplexed with analog inputs will be configured as an analog input. The corresponding TRIS bit will be set.

The A/D operation is independent of the state of the CHS2:CHS0 bits and the TRIS bits.

Note 1:	When reading the port register, any pin configured as an analog input channel will read as cleared (a low level). Pins configured as digital inputs, will convert an analog input. Analog levels on a digitally configured input will not affect the conversion accuracy.
2:	Analog levels on any pin that is defined as a digital input (including the AN7:AN0 pins), may cause the input buffer to consume current that is out of the devices specification.

25.7 A/D Conversions

Example 25-3 shows how to perform an A/D conversion. The port pins are configured as analog inputs. The analog references (V_{REF+} and V_{REF-}) are the device AV_{DD} and AV_{SS}. The A/D interrupt is enabled, and the A/D conversion clock is FRC. The conversion is performed on the AN0 pin (channel 0). The result of the conversion is left justified.

Note:	The GO/$\overline{\text{DONE}}$ bit should **NOT** be set in the same instruction that turns on the A/D, due to the required acquisition time.

Clearing the GO/$\overline{\text{DONE}}$ bit during a conversion will abort the current conversion. The A/D result register pair will NOT be updated with the partially completed A/D conversion sample. That is, the ADRESH:ADRESL registers will continue to contain the value of the last completed conversion (or the last value written to the ADRESH:ADRESL registers). After the A/D conversion is aborted, a $2T_{AD}$ wait is required before the next acquisition is started. After this $2T_{AD}$ wait, acquisition on the selected channel is automatically started.

Example 25-3: A/D Conversion

```
    CLRF    ADCON1          ; Configure A/D inputs,
                            ;    result is left justified
    BSF     IPR1, ADIP      ; High priority
    BSF     PIE1, ADIE      ; Enable A/D interrupts
    MOVLW   0xC1            ; RC Clock, A/D is on,
    MOVWF   ADCON0          ;    Channel 0 is selected
    BCF     PIR1, ADIF      ; Clear A/D interrupt flag bit
    BSF     INTCON, PEIE    ; Enable peripheral interrupts
    BSF     INTCON, GIE     ; Enable all interrupts
;
;  Ensure that the required sampling time for the selected input
;  channel has elapsed. Then the conversion may be started.
;
    BSF     ADCON0, GO      ; Start A/D Conversion
      :                     ; The ADIF bit will be set and the
      :                     ;     GO/DONE bit is cleared upon
      :                     ;     completion of the A/D Conversion.
```

Figure 25-4: A/D Conversion T_{AD} Cycles

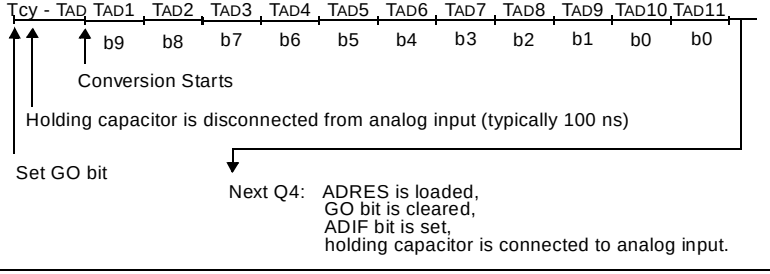
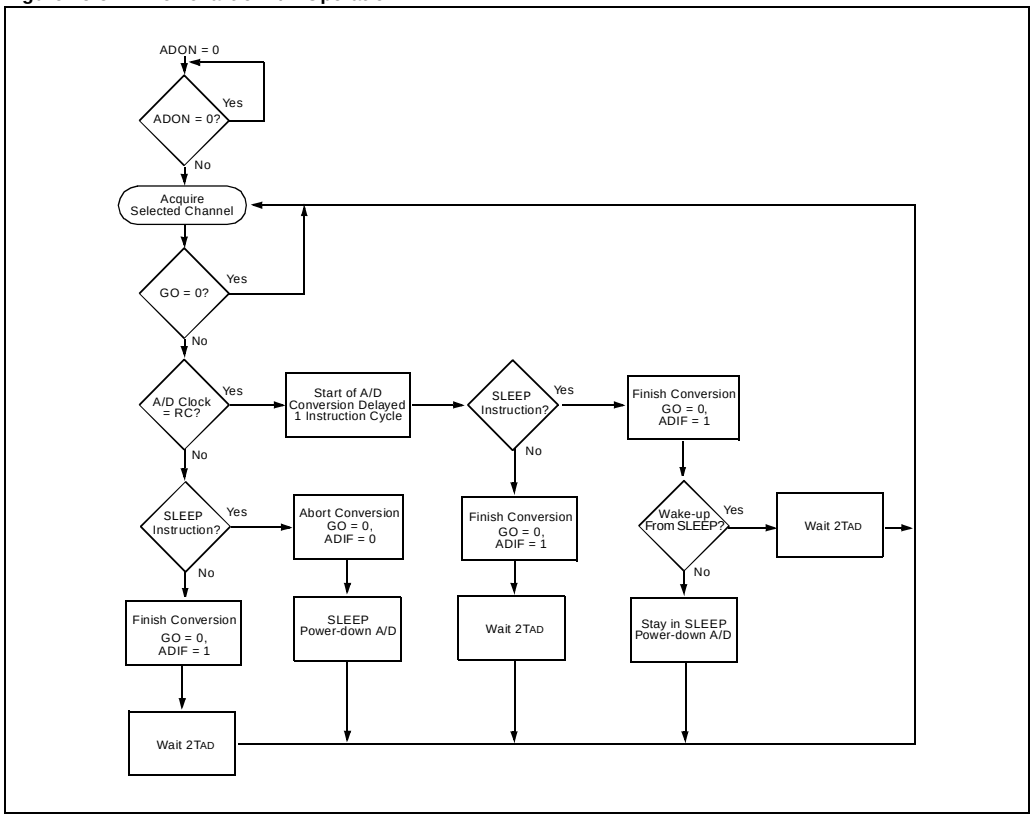


Figure 25-5: Flowchart of A/D Operation



25.7.1 Faster Conversion - Lower Resolution Trade-off

Not all applications require a result with 10-bits of resolution, but may instead require a faster conversion time. The A/D module allows users to make the trade-off of conversion speed to resolution. Regardless of the resolution required, the acquisition time is the same. To speed up the conversion, the clock source of the A/D module may be switched so that the TAD time violates the minimum specified time (see electrical specification parameter 130). Once the TAD time violates the minimum specified time, all the following A/D result bits are not valid (see A/D Conversion Timing in the Electrical Specifications section). The clock sources may only be switched between the three oscillator versions (cannot be switched from/to RC). The equation to determine the time before the oscillator can be switched is as follows:

Since the TAD is based from the device oscillator, the user must use some method (a timer, software loop, etc.) to determine when the A/D oscillator may be changed. Example 25-4 shows a comparison of time required for a conversion with 4-bits of resolution, versus the 10-bit resolution conversion. The example is for devices operating at 20 MHz (the A/D clock is programmed for 32TOSC), and assumes that immediately after 6TAD, the A/D clock is programmed for 2TOSC.

The 2TOSC violates the minimum TAD time since the last 6 bits will not be converted to correct values.

Example 25-4: 4-bit vs. 10-bit Conversion Times

	Freq. (MHz)(1)	Resolution	
		4-bit	10-bit
TAD	40	1.6 µs	1.6 µs
TOSC	40	25 ns	25 ns
TAD + N • TAD + (11 - N)(2TOSC)	40	8.5 µs	17.7 µs

Note 1: A minimum TAD time of 1.6 µs is required.
2: If the full 10-bit conversion is required, the A/D clock source should not be changed.

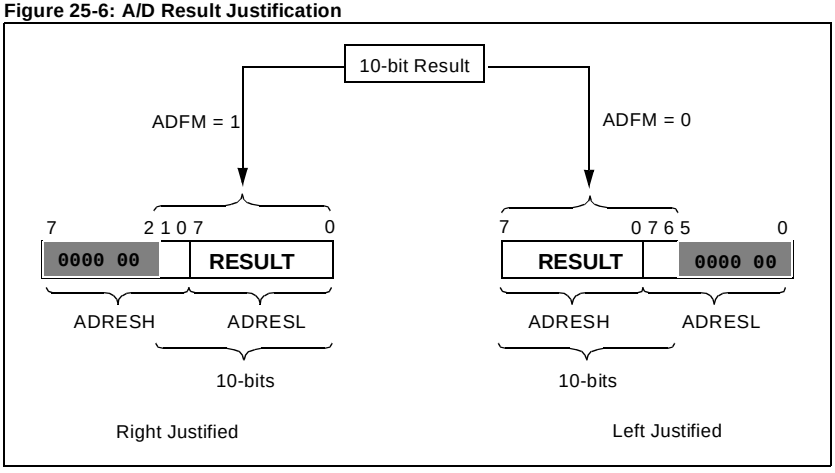
Equation 25-3: Resolution/Speed Conversion Trade-off

Conversion time = TAD + N • TAD + (11 - N)(2TOSC)
Where: N = number of bits of resolution required

25.7.2 A/D Result Registers

The ADRESH:ADRESL register pair is the location where the 10-bit A/D result is loaded at the completion of the A/D conversion. This register pair is 16-bits wide. The A/D module gives the flexibility to left or right justify the 10-bit result in the 16-bit result register. The A/D Format Select bit (ADFM) controls this justification. Figure 25-6 shows the operation of the A/D result justification. The extra bits are loaded with '0's'. When the A/D module is disabled these registers may be used as two general purpose 8-bit registers.

Figure 25-6: A/D Result Justification



25.8 Operation During SLEEP

The A/D module can operate during SLEEP mode. This requires that the A/D clock source be set to RC (ADCS2:ADCS0 = x11). When the RC clock source is selected, the A/D module waits one instruction cycle before starting the conversion. This allows the SLEEP instruction to be executed, which eliminates all internal digital switching noise from the conversion. When the conversion is completed, the GO/$\overline{\text{DONE}}$ bit will be cleared and the result is loaded into the ADRESH:ADRESL registers. If the A/D interrupt is enabled, the device will wake-up from SLEEP. If the A/D interrupt is not enabled, the A/D module will be turned off, although the ADON bit will remain set.

When the A/D clock source is another clock option (not RC), a SLEEP instruction will cause the present conversion to be aborted and the A/D module to be turned off (to conserve power), though the ADON bit will remain set.

Turning off the A/D places the A/D module in its lowest current consumption state.

Note:	For the A/D module to operate in SLEEP, the A/D clock source must be set to RC (ADCS2:ADCS0 = x11). To allow the conversion to occur during SLEEP, ensure the SLEEP instruction immediately follows the instruction that sets the GO/$\overline{\text{DONE}}$ bit.

25.9 Effects of a RESET

A device RESET forces all registers to their RESET state. This forces the A/D module to be turned off, and any conversion is aborted. All pins that are multiplexed with analog inputs will be configured as an analog input. The corresponding TRIS bits will be set.

The value that is in the ADRESH:ADRESL registers is not initialized from a Power-on Reset. The ADRESH:ADRESL registers will contain unknown data after a Power-on Reset.

25.10 A/D Accuracy/Error

In systems where the device frequency is low, use of the A/D RC clock is preferred. At moderate to high frequencies, TAD should be derived from the device oscillator.

For a given range of analog inputs, the output digital code will be the same. This is due to the quantization of the analog input to a digital code. Quantization error is typically ± 1/2 LSb and is inherent in the analog to digital conversion process. The only way to reduce quantization error is to increase the resolution of the A/D converter.

Offset error measures the first actual transition of a code versus the first ideal transition of a code. Offset error shifts the entire transfer function. Offset error can be calibrated out of a system or introduced into a system, through the interaction of the total leakage current and source impedance at the analog input.

Gain error measures the maximum deviation of the last actual transition and the last ideal transition, adjusted for offset error. This error appears as a change in slope of the transfer function. The difference in gain error to full scale error, is that full scale does not take offset error into account. Gain error can be calibrated out in software.

Linearity error refers to the uniformity of the code changes. Linearity errors cannot be calibrated out of the system. Integral non-linearity error measures the actual code transition versus the ideal code transition, adjusted by the gain error for each code.

Differential non-linearity measures the maximum actual code width versus the ideal code width. This measure is unadjusted.

The maximum pin leakage current is specified in Electrical Specifications parameter D060.

TAD must not violate the minimum and should be minimized to reduce inaccuracies due to noise and sampling capacitor bleed off.

In systems where the device will enter SLEEP mode after the start of the A/D conversion, the RC clock source selection is required. In this mode, the digital noise from the modules in SLEEP are stopped. This method gives high accuracy.

25.11 Connection Considerations

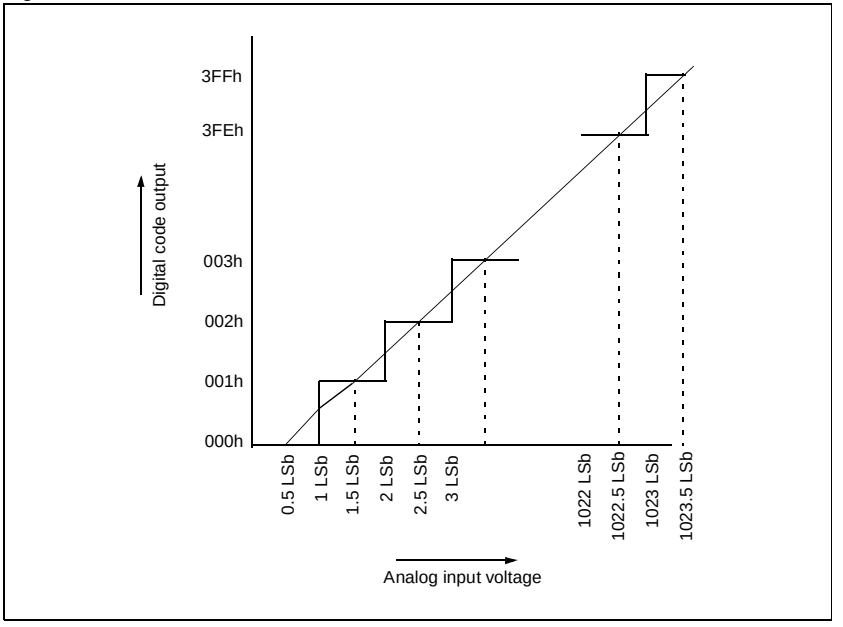
If the input voltage exceeds the rail values (V_{SS} or V_{DD}) by greater than 0.3V, then the accuracy of the conversion is out of specification.

An external RC filter is sometimes added for anti-aliasing of the input signal. The R component should be selected to ensure that the total source impedance is kept under the 2.5 kΩ recommended specification. Any external components connected (via hi-impedance) to an analog input pin (capacitor, zener diode, etc.) should have very little leakage current at the pin.

25.12 Transfer Function

The ideal transfer function of the A/D converter is as follows: the first transition occurs when the analog input voltage (V_{AIN}) is 1 LSb (or Analog V_{REF} / 1024) (Figure 25-7).

Figure 25-7: A/D Transfer Function



25.13 Initialization

Example 25-5 shows an initialization of the A/D module.

Example 25-5: A/D Initialization

```
  CLRF    ADCON1           ; Configure A/D inputs
  BSF     PIE1, ADIE       ; Enable A/D interrupts
  BSF     IPR1, ADIP       ; High Priority
  MOVLW   0xC1             ; RC Clock, A/D is on,
  MOVWF   ADCON0           ; Channel 0 is selected
  MOVLW   0x4E             ; Left Justified, AN0 is analog
  MOVWF   ADCON1           ; Vref comes from AVDD and AVSS
  BCF     PIR1, ADIF       ; Clear A/D interrupt flag bit
  BSF     INTCON, PEIE     ; Enable peripheral interrupts
  BSF     INTCON, GIE      ; Enable all interrupts
;
; Ensure that the required sampling time for the selected input
; channel has elapsed. Then the conversion may be started.
;
  BSF     ADCON0, GO       ; Start A/D Conversion
    :                      ; The ADIF bit will be set and the
    :                      ;  GO/DONE bit is cleared upon
    :                      ;  completion of the A/D conversion.
```

25.14 Design Tips

Question 1: ***I find that the Analog to Digital Converter result is not always accurate. What can I do to improve accuracy?***

Answer 1:

1. Make sure you are meeting all of the timing specifications. If you are turning the module off and on, there is a minimum delay you must wait before taking a sample. If you are changing input channels, there is a minimum delay you must wait for this as well, and finally there is TAD, which is the time selected for each bit conversion. This is selected in ADCON0 and should be between 1.6 and 6 µs. If TAD is too short, the result may not be fully converted before the conversion is terminated, and if TAD is made too long, the voltage on the sampling capacitor can decay before the conversion is complete. These timing specifications are provided in the **"Electrical Specifications"** section. See the device data sheet for device specific information.
2. Often the source impedance of the analog signal is high (greater than 1 kOhms), so the current drawn from the source to charge the sample capacitor can affect accuracy. If the input signal does not change too quickly, try putting a 0.1 µF capacitor on the analog input. This capacitor will charge to the analog voltage being sampled and supply the instantaneous current needed to charge the 120 pF internal holding capacitor.
3. In systems where the device frequency is low, use of the A/D clock derived from the device oscillator is preferred...this reduces, to a large extent, the effects of digital switching noise. In systems where the device will enter SLEEP mode after start of A/D conversion, the RC clock source selection is required.This method gives the highest accuracy.

Question 2: ***After starting an A/D conversion may I change the input channel (for my next conversion)?***

Answer 2:

After the holding capacitor is disconnected from the input channel, typically 100 ns after the GO bit is set, the input channel may be changed.

Question 3: ***Do you know of a good reference on A/D's?***

Answer 3:

A good reference for understanding A/D conversions is the "Analog-Digital Conversion Handbook" third edition, published by Prentice Hall (ISBN 0-13-03-2848-0).

Question 4: ***I migrated my code from a PIC18CXX8 device with 10-bit A/D to another device with a 10-bit A/D (such as a PIC18CXX2) and the A/D does not seem to operate the same. What's going on?***

Answer 4:

The 10-bit A/D on the PIC18CXX2 device is the compatible 10-bit A/D module. This module has its ADCON bits in the same locations as the PICmicro's Mid-Range 10-bit A/D module. The standard PIC18CXXX 10-bit A/D module (as found on the PIC18CXX8 device) has optimized the bit locations to ease configuration of the module.

25.15 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is they may be written for the Base-Line, the Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the 10-bit A/D module are:

Title	Application Note #
Using the Analog to Digital Converter	AN546
Four Channel Digital Voltmeter with Display and Keyboard	AN557

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

25.16 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Compatible 10-bit A/D module description.

Section 26. 10-bit A/D Converter

HIGHLIGHTS

This section of the manual contains the following major topics:

26.1 Introduction

The 10-bit Analog-to-Digital (A/D) Converter module can have up to sixteen analog inputs.

The analog input charges a sample and hold capacitor. The output of the sample and hold capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation. This A/D conversion of the analog input signal results in a corresponding 10-bit digital number.

The analog reference voltages (positive and negative supply) are software selectable to either the device's supply voltages (AVDD, AVss) or the voltage level on the AN3/VREF+ and AN2/VREF- pins.

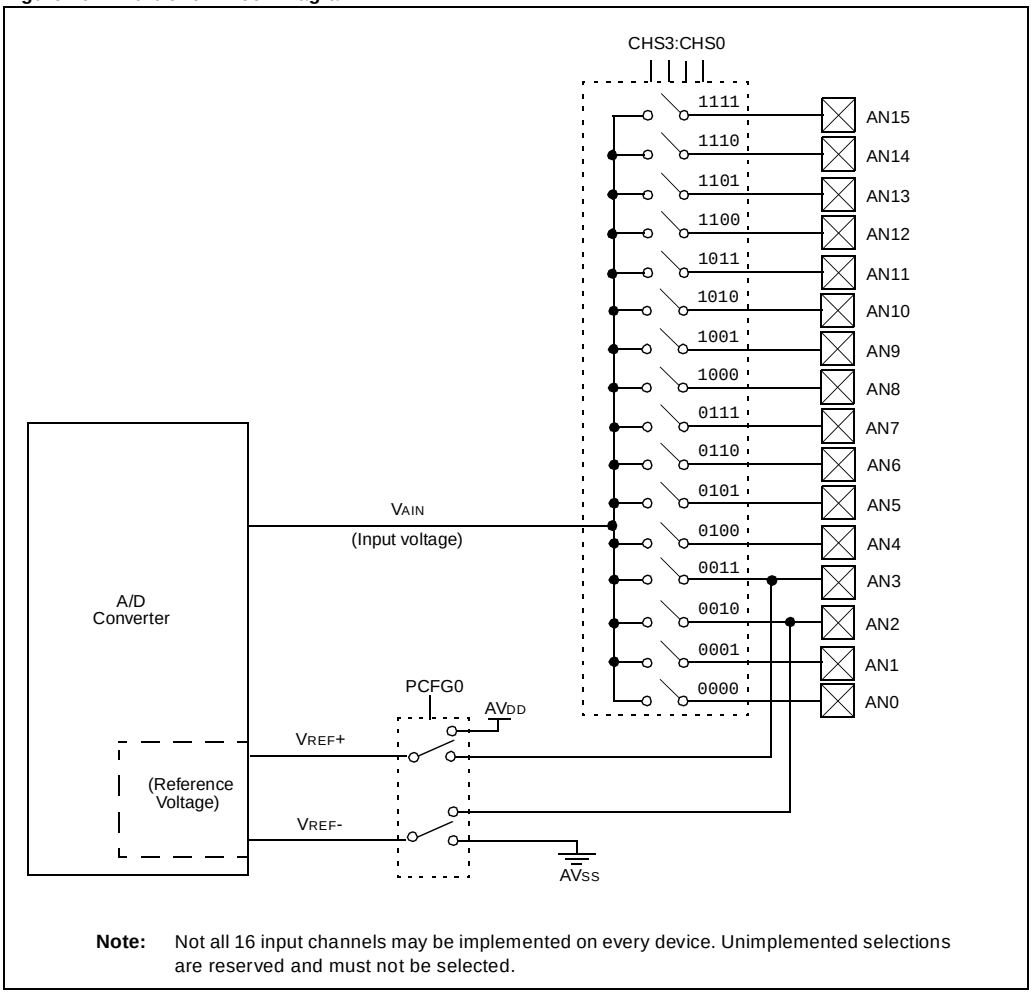
The A/D converter has the unique feature of being able to convert while the device is in SLEEP mode.

The A/D module has five registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)
- A/D Control Register2 (ADCON2)

The ADCON0 register, shown in Register 26-1, selects the input channel of the A/D module. The ADCON1 register, shown in Register 26-2, configures the functions of the port pins and the Voltage Reference for the A/D module. The port pins can be configured as analog inputs (AN3 and AN2 can also be the Voltage References) or as digital I/O. ADCON2 selects the A/D conversion clock source and the format of the A/D result.

Figure 26-1: 10-bit A/D Block Diagram



Note: Not all 16 input channels may be implemented on every device. Unimplemented selections are reserved and must not be selected.

26.2 Control Register

ADCON0 (Register 26-1) is used to select the analog channel. ADCON1 (Register 26-2) configures the port logic to either analog or digital inputs and the voltage reference source for the A/D. ADCON2 (Register 26-3) selects the source of the A/D clock and the justification of the result.

Register 26-1: ADCON0 Register

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

bit 7-6: **Unimplemented:** Read as '0'

bit 5-2: **CHS3:CHS0:** Analog Channel Select bits

0000 = channel 0, (AN0)
0001 = channel 1, (AN1)
0010 = channel 2, (AN2)
0011 = channel 3, (AN3)
0100 = channel 4, (AN4)
0101 = channel 5, (AN5)
0110 = channel 6, (AN6)
0111 = channel 7, (AN7)
1000 = channel 8, (AN8)
1001 = channel 8, (AN9)
1010 = channel 10, (AN10)
1011 = channel 11, (AN11)
1100 = channel 12, (AN12)
1101 = channel 13, (AN13)
1110 = channel 14, (AN14)
1111 = channel 15, (AN15)

bit 1: **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit

1 = A/D conversion in progress. Setting this bit starts an A/D conversion cycle.
This bit is automatically cleared by hardware when the A/D conversion is completed.
0 = A/D conversion not in progress

bit 0: **ADON:** A/D On bit

1 = A/D converter module is operating
0 = A/D converter module is shut off and consumes no operating current

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 26-2: ADCON1 Register

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7-6: **Unimplemented:** Read as '0'

bit 5-4: **VCFG1:VCFG0:** Voltage Reference Configuration bits

	A/D VREFH	**A/D VREFL**
00	AVDD	AVSS
01	External VREF+	AVSS
10	AVDD	External VREF-
11	External VREF+	External VREF-

bit 3-0: **PCFG3:PCFG0:** A/D Port Configuration Control bits **(1)**

	AN15	AN14	AN13	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
0000	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	D	D	A	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	D	D	D	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	D	D	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	D	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	D	D	D	A	A	A	A	A	A	A	A	A
0111	D	D	D	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Note 1: Selection of an unimplemented channel produces a result of 0xFFF.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

Register 26-3: ADCON2 Register

R/W-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	—	ADCS2	ADCS1	ADCS0
bit 7							bit 0

bit 7: **ADFM:** A/D Result Format Select bit

1 = Right justified
0 = Left justified

bit 6-3: **Unimplemented:** Read as '0'

bit 2-0: **ADCS2:ADCS0:** A/D Conversion Clock Select bits

000 = FOSC/2
001 = FOSC/8
010 = FOSC/32
011 = FRC (clock derived from an internal RC oscillator, 1 MHz maximum frequency)
100 = FOSC/4
101 = FOSC/16
110 = FOSC/64
111 = FRC (clock derived from an RC oscillator, 1 MHz maximum frequency)

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

26.3 Operation

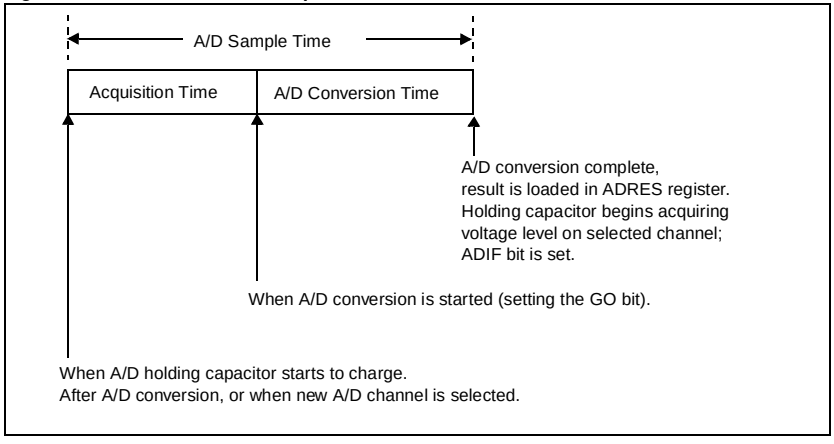
The ADRESH:ADRESL registers contain the 10-bit result of the A/D conversion. When the A/D conversion is complete, the result is loaded into this A/D result register pair (ADRESH:ADRESL), the GO/$\overline{\text{DONE}}$ bit (ADCON0 register) is cleared, and A/D interrupt flag bit, ADIF, is set. The block diagram of the A/D module is shown in Figure 26-1.

After the A/D module has been configured, the signal on the selected channel must be acquired before the conversion is started. The analog input channels must have their corresponding TRIS bits selected as inputs. To determine acquisition time, see Subsection **26.4 "A/D Acquisition Requirements."** After this acquisition time has elapsed, the A/D conversion can be started. The following steps should be followed for doing an A/D conversion:

1. Configure the A/D module:
   - Configure analog pins, Voltage Reference, and digital I/O (ADCON1)
   - Select A/D input channel (ADCON0)
   - Select A/D conversion clock (ADCON0)
   - Turn on A/D module (ADCON0)
2. Configure A/D interrupt (if desired):
   - Clear the ADIF bit
   - Set the ADIE bit
   - Set/Clear the ADIP bit
   - Set the GIE/GIEH or PEIE/GIEL bit
3. Wait the required acquisition time.
4. Start conversion:
   - Set the GO/$\overline{\text{DONE}}$ bit (ADCON0)
5. Wait for the A/D conversion to complete, by either:
   - Polling for the GO/$\overline{\text{DONE}}$ bit to be cleared or the ADIF bit to be set, or
   - Waiting for the A/D interrupt
6. Read A/D Result register pair (ADRESH:ADRESL): clear the ADIF bit, if required.
7. For next conversion, go to step 1 or step 2 as required.

Figure 26-2 shows the conversion sequence, and the terms that are used. Acquisition time is the time that the A/D module's holding capacitor is connected to the external voltage level. When the GO bit is set, the conversion time of 12 TAD is started. The sum of these two times is the sampling time. There is a minimum acquisition time to ensure that the holding capacitor is charged to a level that will give the desired accuracy for the A/D conversion.

Figure 26-2: A/D Conversion Sequence



26.4 A/D Acquisition Requirements

For the A/D converter to meet its specified accuracy, the charge holding capacitor (CHOLD) must be allowed to fully charge to the input channel voltage level. The analog input model is shown in Figure 26-3. The source impedance (RS) and the internal sampling switch (RSS) impedance directly affect the time required to charge the capacitor CHOLD. The sampling switch (RSS) impedance varies over the device voltage (VDD), Figure 26-3. The source impedance affects the offset voltage at the analog input (due to pin leakage current). **The maximum recommended impedance for analog sources is 2.5 kΩ.** As the impedance is decreased, the acquisition time may be decreased. After the analog input channel is selected (changed), this acquisition must pass before the conversion can be started.

To calculate the minimum acquisition time, Equation 26-1 may be used. This equation assumes that 1/2 LSb error is used (1024 steps for the A/D). The 1/2 LSb error is the maximum error allowed for the A/D to meet its specified resolution.

Note:	When the conversion is started, the holding capacitor is disconnected from the input pin.

Equation 26-1: Acquisition Time

TACQ	equals	Amplifier Settling Time (TAMP) plus Holding Capacitor Charging Time (TC) plus Temperature Coefficient (TCOFF)
TACQ	=	TAMP + TC + TCOFF

Equation 26-2: A/D Minimum Charging Time

$$V_{HOLD} = (V_{REF} - (V_{REF}/2048)) \cdot (1 - e^{(-T_C/C_{HOLD}(R_{IC} + R_{SS} + R_S))})$$

or

$$T_C = -(120\ pF)(1\ k\Omega + R_{SS} + R_S) \ln(1/2047)$$

Example 26-1 shows the calculation of the minimum required acquisition time TACQ. This calculation is based on the following application system assumptions.

CHOLD	=	120 pF	
RS	=	2.5 kΩ	
Conversion Error	≤	1/2 LSb	
VDD	=	5V → RSS = 7 kΩ	(see graph in Figure 26-3)
Temperature	=	50°C (system max.)	
VHOLD	=	0V @ time = 0	

Example 26-1: Calculating the Minimum Required Acquisition Time (Case 1)

TACQ = TAMP + TC + TCOFF

Temperature coefficient is only required for temperatures > 25°C.

TACQ = 2 µs + Tc + [(Temp - 25°C)(0.05 µs/°C)]

TC = -CHOLD (RIC + RSS + RS) ln(1/2047)
-120 pF (1 kΩ + 7 kΩ + 2.5 kΩ) ln(0.0004885)
-120 pF (10.5 kΩ) ln(0.0004885)
-1.26 µs (-7.6241)
9.61 µs

TACQ = 2 µs + 9.61µs + [(50°C - 25°C)(0.05 µs/°C)]
11.61 µs + 1.25 µs
12.86 µs

Now to get an idea what happens to the acquisition time when the source impedance is a minimal value (RS = 50 Ω). Example 26-2 shows the same conditions as in Example 26-1 with only the source impedance changed to the minimal value.

Example 26-2: Calculating the Minimum Required Acquisition Time (Case 2)

TACQ	=	TAMP + TC + TCOFF
Temperature coefficient is only required for temperatures > 25°C.		
TACQ	=	2 µs + Tc + [(Temp - 25°C)(0.05 µs/°C)]
TC	=	-Chold (Ric + Rss + Rs) ln(1/2047) -120 pF (1 kΩ + 7 kΩ + 50 Ω) ln(0.0004885) -120 pF (8050 Ω) ln(0.0004885) -0.966 µs (-7.6241) 7.36 µs
TACQ	=	2 µs + 16.47 µs + [(50°C - 25°C)(0.05 µs/°C)] 9.36 µs + 1.25 µs 10.61 µs

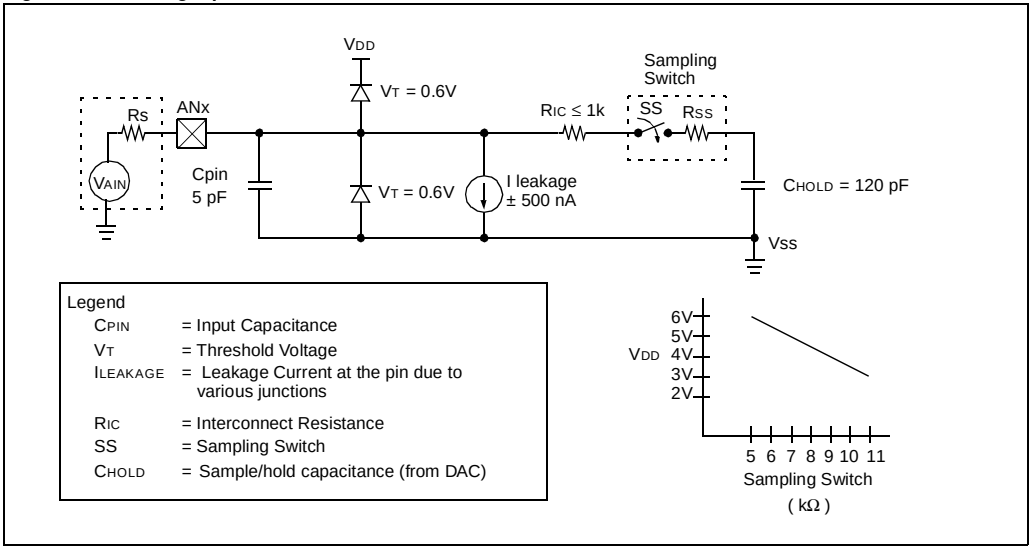
Note 1: The reference voltage (VREF) has no effect on the equation, since it cancels itself out.

2: The charge holding capacitor (CHOLD) is not discharged after each conversion.

3: The maximum recommended impedance for analog sources is 2.5 kΩ. This is required to meet the pin leakage specification.

4: After a conversion has completed, a 2 TAD delay must complete before acquisition can begin again. During this time the holding capacitor is not connected to the selected A/D input channel.

Figure 26-3: Analog Input Model



26.5 Selecting the A/D Conversion Clock

The A/D conversion time per bit is defined as TAD. The A/D conversion requires 11.5TAD per 10-bit conversion. The source of the A/D conversion clock is software selectable. The seven possible options for TAD are:

- 2TOSC
- 4TOSC
- 8TOSC
- 16TOSC
- 32TOSC
- 64TOSC
- Internal A/D RC oscillator

For correct A/D conversions, the A/D conversion clock (TAD) must be selected to ensure a minimum TAD time of 1.6 µs as shown in Electrical Specifications parameter 130.

Table 26-1 and Table 26-2 show the resultant TAD times derived from the device operating frequencies and the selected A/D clock source.

Table 26-1: TAD vs. Device Operating Frequencies (for Standard, C, Devices)

AD Clock Source (TAD)		Device Frequency			
Operation	**ADCS2:ADCS0**	**20 MHz**	**5 MHz**	**1.25 MHz**	**333.33 kHz**
2TOSC	000	100 ns (2)	400 ns (2)	1.6 µs	6 µs
4TOSC	100	200 ns (2)	800 ns (2)	3.2 µs	12 µs
8TOSC	001	400 ns (2)	1.6 µs	6.4 µs	24 µs (3)
16TOSC	101	800 ns (2)	3.2 µs	12.8 µs	48 µs (3)
32TOSC	010	1.6 µs	6.4 µs	25.6 µs (3)	96 µs (3)
64TOSC	110	3.2 µs	12.8 µs	51.2 µs (3)	192 µs (3)
RC	011	2 - 6 µs (1,4)	2 - 6 µs (1,4)	2 - 6 µs (1,4)	2 - 6 µs (1)

Legend: Shaded cells are outside of recommended range.

Note 1: The RC source has a typical TAD of 4 µs.

2: These values violate the minimum required TAD.

3: For faster conversion times, the selection of another clock source is recommended.

4: For device frequencies above 1 MHz, the device must be in SLEEP for the entire conversion, or the A/D accuracy may be out of specification.

Table 26-2: TAD vs. Device Operating Frequencies (for Extended, LC, Devices)

AD Clock Source (TAD)		Device Frequency			
Operation	**ADCS2:ADCS0**	**4 MHz**	**2 MHz**	**1.25 MHz**	**333.33 kHz**
2TOSC	000	500 ns (2)	1.0 µs (2)	1.6 µs (2)	6 µs
4TOSC	100	1.0 µs (2)	2.0 µs (2)	3.2 µs (2)	12 µs
8TOSC	001	2.0 µs (2)	4.0 µs	6.4 µs	24 µs (3)
16TOSC	101	4.0 µs	8.0 µs	12.8 µs	48 µs (3)
32TOSC	010	8.0 µs	16.0 µs	25.6 µs (3)	96 µs (3)
64TOSC	110	16.0 µs	32.0 µs	51.2 µs (3)	192 µs (3)
RC	011	3 - 9 µs (1,4)	3 - 9 µs (1,4)	3 - 9 µs (1,4)	3 - 9 µs (1)

Legend: Shaded cells are outside of recommended range.

Note 1: The RC source has a typical TAD of 6 µs.

2: These values violate the minimum required TAD.

3: For faster conversion times, the selection of another clock source is recommended.

4: For device frequencies above 1 MHz, the device must be in SLEEP for the entire conversion, or the A/D accuracy may be out of specification.

26.6 Configuring Analog Port Pins

The ADCON1 and TRIS registers control the operation of the A/D port pins. The port pins that are desired as analog inputs must have their corresponding TRIS bits set (input). If the TRIS bit is cleared (output), the digital output level (V_{OH} or V_{OL}) will be converted. After a device RESET, pins that are multiplexed with analog inputs will be configured as an analog input. The corresponding TRIS bit will be set.

The A/D operation is independent of the state of the CHS2:CHS0 bits and the TRIS bits.

Note 1:	When reading the port register, any pin configured as an analog input channel will read as cleared (a low level). Pins configured as digital inputs, will convert an analog input. Analog levels on a digitally configured input will not affect the conversion accuracy.
2:	Analog levels on any pin that is defined as a digital input (including the AN7:AN0 pins), may cause the input buffer to consume current that is out of the devices specification.

26.7 A/D Conversions

Example 26-3 shows how to perform an A/D conversion. The port pins are configured as analog inputs. The analog references (VREF+ and VREF-) are the device AVDD and AVSS. The A/D interrupt is enabled, and the A/D conversion clock is FRC. The conversion is performed on the AN0 pin (channel 0). The result of the conversion is left justified.

Note:	The GO/$\overline{\text{DONE}}$ bit should **NOT** be set in the instruction that turns on the A/D, due to the required acquisition time.

Clearing the GO/$\overline{\text{DONE}}$ bit during a conversion will abort the current conversion. The A/D result register pair will NOT be updated with the partially completed A/D conversion sample. That is, the ADRESH:ADRESL registers will continue to contain the value of the last completed conversion (or the last value written to the ADRESH:ADRESL registers). After the A/D conversion is aborted, a 2TAD wait is required before the next acquisition is started. After this 2TAD wait, acquisition on the selected channel is automatically started.

Example 26-3: A/D Conversion

```
    CLRF    ADCON1          ; Configure A/D inputs,
                            ;    result is left justified
    BSF     IPR1, ADIP      ; High Priority.
    BSF     PIE1, ADIE      ; Enable A/D interrupts
    MOVLW   0xC1            ; RC Clock, A/D is on,
    MOVWF   ADCON0          ;    Channel 0 is selected
    BCF     PIR1, ADIF      ; Clear A/D interrupt flag bit
    BSF     INTCON, PEIE    ; Enable peripheral interrupts
    BSF     INTCON, GIE     ; Enable all interrupts
;
;  Ensure that the required sampling time for the selected input
;  channel has elapsed. Then the conversion may be started.
;
    BSF     ADCON0, GO      ; Start A/D Conversion
      :                     ; The ADIF bit will be set and the
      :                     ;     GO/DONE bit is cleared upon
      :                     ;     completion of the A/D Conversion.
```

Figure 26-4: A/D Conversion TAD Cycles

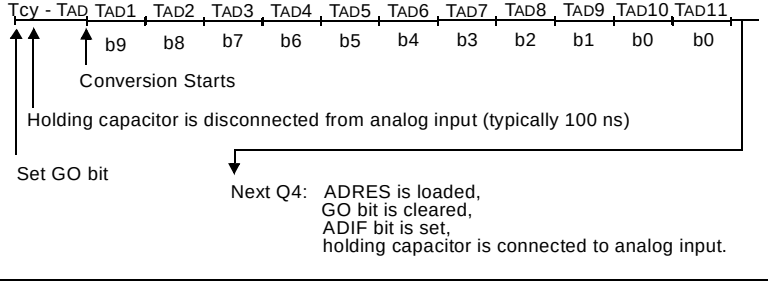
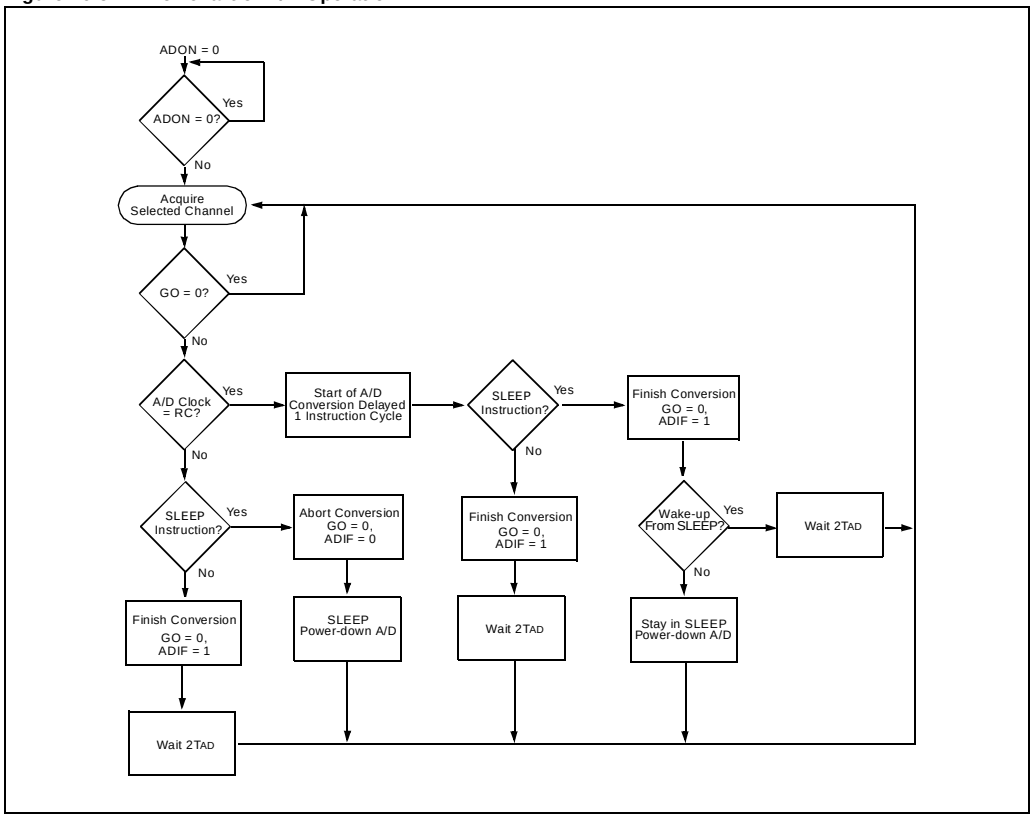


Figure 26-5: Flowchart of A/D Operation



26.7.1 Faster Conversion - Lower Resolution Trade-off

Not all applications require a result with 10-bits of resolution, but may instead require a faster conversion time. The A/D module allows users to make the trade-off of conversion speed to resolution. Regardless of the resolution required, the acquisition time is the same. To speed up the conversion, the clock source of the A/D module may be switched so that the TAD time violates the minimum specified time (Electrical Specifications parameter 130). Once the TAD time violates the minimum specified time, all the following A/D result bits are not valid (see A/D Conversion Timing in the Electrical Specifications section). The clock sources may only be switched between the three oscillator versions (cannot be switched from/to RC). The equation to determine the time before the oscillator can be switched is as follows:

Since the TAD is based from the device oscillator, the user must use some method (a timer, software loop, etc.) to determine when the A/D oscillator may be changed. Example 26-4 shows a comparison of time required for a conversion with 4-bits of resolution, versus the 10-bit resolution conversion. The example is for devices operating at 20 MHz (the A/D clock is programmed for 32TOSC), and assumes that immediately after 6TAD, the A/D clock is programmed for 2TOSC.

The 2TOSC violates the minimum TAD time since the last 6 bits will not be converted to correct values.

Example 26-4: 4-bit vs. 10-bit Conversion Times

	Freq. (MHz)(1)	Resolution	
		4-bit	10-bit
TAD	40	1.6 µs	1.6 µs
TOSC	40	25 ns	25 ns
TAD + N • TAD + (11 - N)(2TOSC)	40	8.5 µs	17.7 µs

Note 1: A minimum TAD time of 1.6 µs is required.
2: If the full 10-bit conversion is required, the A/D clock source should not be changed.

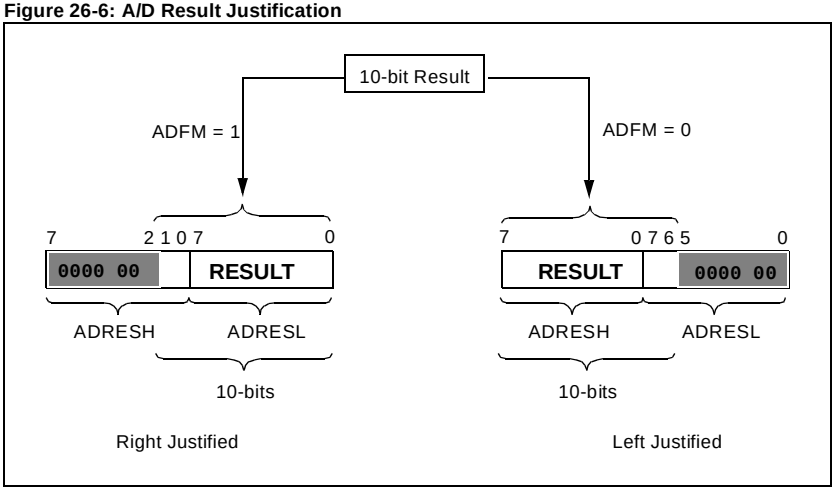
Equation 26-3: Resolution/Speed Conversion Trade-off

Conversion time = TAD + N • TAD + (11 - N)(2TOSC)
Where: N = number of bits of resolution required

26.7.2 A/D Result Registers

The ADRESH:ADRESL register pair is the location where the 10-bit A/D result is loaded at the completion of the A/D conversion. This register pair is 16-bits wide. The A/D module gives the flexibility to left or right justify the 10-bit result in the 16-bit result register. The A/D Format Select bit (ADFM) controls this justification. Figure 26-6 shows the operation of the A/D result justification. The extra bits are loaded with '0's'. When the A/D module is disabled, these registers may be used as two general purpose 8-bit registers.

Figure 26-6: A/D Result Justification



26.8 Operation During SLEEP

The A/D module can operate during SLEEP mode. This requires that the A/D clock source be set to RC (ADCS2:ADCS0 = x11). When the RC clock source is selected, the A/D module waits one instruction cycle before starting the conversion. This allows the SLEEP instruction to be executed, which eliminates all internal digital switching noise from the conversion. When the conversion is completed, the GO/$\overline{\text{DONE}}$ bit will be cleared and the result is loaded into the ADRESH:ADRESL registers. If the A/D interrupt is enabled, the device will wake-up from SLEEP. If the A/D interrupt is not enabled, the A/D module will be turned off, although the ADON bit will remain set.

When the A/D clock source is another clock option (not RC), a SLEEP instruction will cause the present conversion to be aborted and the A/D module to be turned off (to conserve power), though the ADON bit will remain set.

Turning off the A/D places the A/D module in its lowest current consumption state.

Note:	For the A/D module to operate in SLEEP, the A/D clock source must be set to RC (ADCS2:ADCS0 = x11). To allow the conversion to occur during SLEEP, ensure the SLEEP instruction immediately follows the instruction that sets the GO/$\overline{\text{DONE}}$ bit.

26.9 Effects of a RESET

A device RESET forces all registers to their RESET state. This forces the A/D module to be turned off, and any conversion is aborted. All pins that are multiplexed with analog inputs will be configured as an analog input. The corresponding TRIS bits will be set.

The value that is in the ADRESH:ADRESL registers is not initialized from a Power-on Reset. The ADRESH:ADRESL registers will contain unknown data after a Power-on Reset.

26.10 A/D Accuracy/Error

In systems where the device frequency is low, use of the A/D RC clock is preferred. At moderate to high frequencies, TAD should be derived from the device oscillator.

For a given range of analog inputs, the output digital code will be the same. This is due to the quantization of the analog input to a digital code. Quantization error is typically ± 1/2 LSb and is inherent in the analog to digital conversion process. The only way to reduce quantization error is to increase the resolution of the A/D converter.

Offset error measures the first actual transition of a code versus the first ideal transition of a code. Offset error shifts the entire transfer function. Offset error can be calibrated out of a system or introduced into a system, through the interaction of the total leakage current and source impedance at the analog input.

Gain error measures the maximum deviation of the last actual transition and the last ideal transition, adjusted for offset error. This error appears as a change in slope of the transfer function. The difference in gain error to full scale error, is that full scale does not take offset error into account. Gain error can be calibrated out in software.

Linearity error refers to the uniformity of the code changes. Linearity errors cannot be calibrated out of the system. Integral non-linearity error measures the actual code transition versus the ideal code transition, adjusted by the gain error for each code.

Differential non-linearity measures the maximum actual code width versus the ideal code width. This measure is unadjusted.

The maximum pin leakage current is specified in Electrical Specifications parameter D060.

TAD must not violate the minimum and should be minimized to reduce inaccuracies due to noise and sampling capacitor bleed off.

In systems where the device will enter SLEEP mode after the start of the A/D conversion, the RC clock source selection is required. In this mode, the digital noise from the modules in SLEEP are stopped. This method gives high accuracy.

26.11 Connection Considerations

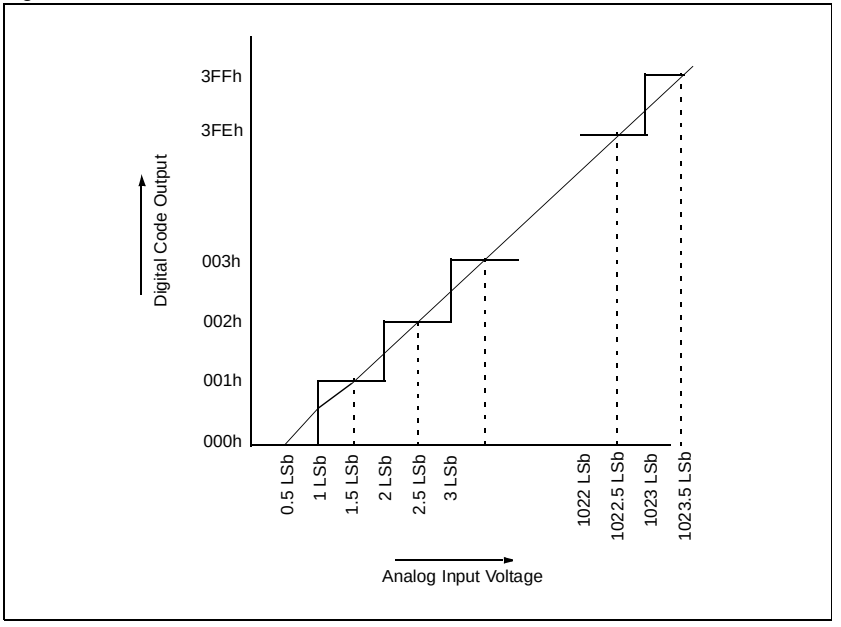
If the input voltage exceeds the rail values (V_{SS} or V_{DD}) by greater than 0.3V, then the accuracy of the conversion is out of specification.

An external RC filter is sometimes added for anti-aliasing of the input signal. The R component should be selected to ensure that the total source impedance is kept under the 2.5 kΩ recommended specification. Any external components connected (via hi-impedance) to an analog input pin (capacitor, zener diode, etc.) should have very little leakage current at the pin.

26.12 Transfer Function

The ideal transfer function of the A/D converter is as follows: the first transition occurs when the analog input voltage (V_{AIN}) is 1 LSb (or Analog V_{REF} / 1024) (Figure 26-7).

Figure 26-7: A/D Transfer Function



26.13 Initialization

Example 26-5 shows an initialization of the A/D module.

Example 26-5: A/D Initialization

```
  CLRF    ADCON1           ; Configure A/D inputs
  BSF     PIE1, ADIE       ; Enable A/D interrupts
  BSF     IPR1, ADIP       ; High Priority
  MOVLW   0xC1             ; RC Clock, A/D is on,
  MOVWF   ADCON0           ; Channel 0 is selected
  MOVLW   0x4E             ; Left Justified, AN0 is analog
  MOVWF   ADCON1           ; Vref comes from AVDD and AVSS
  BCF     PIR1, ADIF       ; Clear A/D interrupt flag bit
  BSF     INTCON, PEIE     ; Enable peripheral interrupts
  BSF     INTCON, GIE      ; Enable all interrupts
;
; Ensure that the required sampling time for the selected input
; channel has elapsed. Then the conversion may be started.
;
  BSF     ADCON0, GO       ; Start A/D Conversion
    :                      ; The ADIF bit will be set and the
    :                      ;  GO/DONE bit is cleared upon
    :                      ;  completion of the A/D conversion.
```

26.14 Design Tips

Question 1: ***I find that the Analog to Digital Converter result is not always accurate. What can I do to improve accuracy?***

Answer 1:

1. Make sure you are meeting all of the timing specifications. If you are turning the module off and on, there is a minimum delay you must wait before taking a sample. If you are changing input channels, there is a minimum delay you must wait for this as well, and finally there is TAD, which is the time selected for each bit conversion. This is selected in ADCON0 and should be between 1.6 and 6 µs. If TAD is too short, the result may not be fully converted before the conversion is terminated, and if TAD is made too long, the voltage on the sampling capacitor can decay before the conversion is complete. These timing specifications are provided in the **"Electrical Specifications"** section. See the device data sheet for device specific information.
2. Often the source impedance of the analog signal is high (greater than 1 kOhms), so the current drawn from the source to charge the sample capacitor can affect accuracy. If the input signal does not change too quickly, try putting a 0.1 µF capacitor on the analog input. This capacitor will charge to the analog voltage being sampled and supply the instantaneous current needed to charge the 120 pF internal holding capacitor.
3. In systems where the device frequency is low, use of the A/D clock derived from the device oscillator is preferred...this reduces, to a large extent, the effects of digital switching noise. In systems where the device will enter SLEEP mode after start of A/D conversion, the RC clock source selection is required.This method gives the highest accuracy.

Question 2: ***After starting an A/D conversion may I change the input channel (for my next conversion)?***

Answer 2:

After the holding capacitor is disconnected from the input channel, typically 100 ns after the GO bit is set, the input channel may be changed.

Question 3: ***Do you know of a good reference on A/D's?***

Answer 3:

A good reference for understanding A/D conversions is the "Analog-Digital Conversion Handbook" third edition, published by Prentice Hall (ISBN 0-13-03-2848-0).

Question 4: ***I migrated my code from a PIC18CXX2 device with 10-bit A/D to another device with a 10-bit A/D (such as a PIC18CXX8) and the A/D does not seem to operate the same. What's going on?***

Answer 4:

The 10-bit A/D on the PIC18CXX2 device is the compatible 10-bit A/D module. This module has its ADCON bits in the same locations as the PICmicros Mid-Range 10-bit A/D module. The standard PIC18CXXX 10-bit A/D module (as found on the PIC18CXX8 device) has optimized the bit locations to ease configuration of the module.

26.15 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is they may be written for the Base-Line, the Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the 10-bit A/D module are:

Title	Application Note #
Using the Analog to Digital Converter	AN546
Four Channel Digital Voltmeter with Display and Keyboard	AN557

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

26.16 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Compatible 10-bit A/D module description.

Section 27. Low Voltage Detect

HIGHLIGHTS

This section of the manual contains the following major topics:

27.1 Introduction

In many applications, the ability to determine if the device voltage (V_{DD}) is below a specified voltage level is a desirable feature. A window of operation for the application can be created where the application software can do "housekeeping tasks" before the device voltage exits the valid operating range. This can be done using the Low Voltage Detect module.

This module is software programmable circuitry, where a device voltage trip point can be specified. When the voltage of the device becomes lower than the specified point, an interrupt flag is set. If the interrupt is enabled, the program execution will branch to the interrupt vector address, and the software can then respond to that interrupt source.

The Low Voltage Detect circuitry is completely under software control. This allows the circuitry to be "turned off" by the software, which minimizes the current consumption for the device.

Figure 27-1 shows a possible application voltage curve (typically for batteries). Over time the device voltage decreases. When the device voltage equals voltage V_A, the LVD logic generates an interrupt. This occurs at time T_A. The application software then has until the device voltage is no longer in valid operating range to have shut down the system. Voltage point V_B is the minimum valid operating voltage specification. This gives a time T_B. The total time for shutdown is $T_B - T_A$.

Figure 27-1: Typical Low Voltage Detect Application

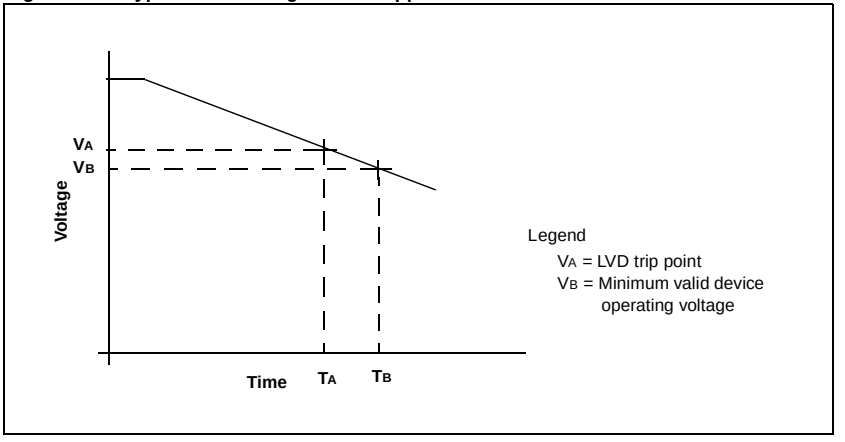
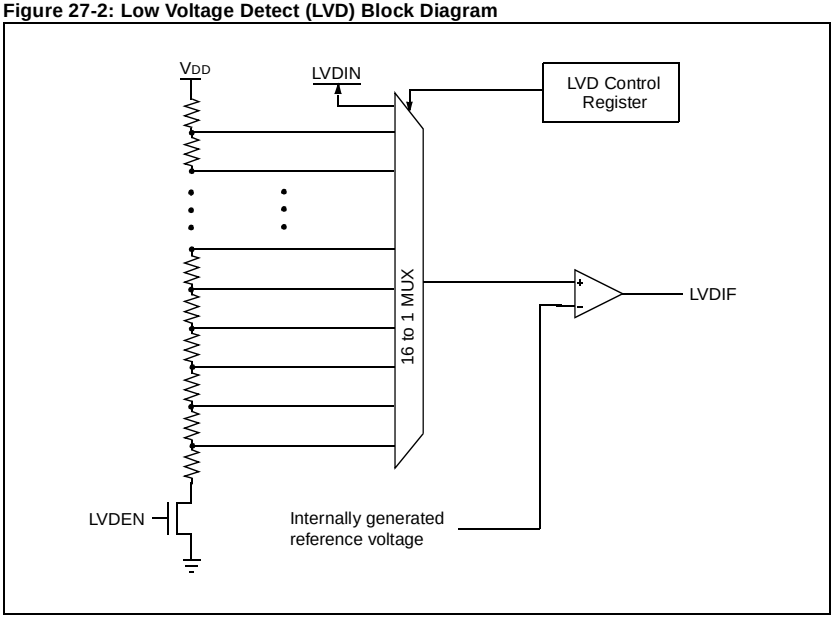


Figure 27-2 shows the block diagram for the LVD module. A comparator uses an internally generated reference voltage as the set point. When the selected tap output of the device voltage crosses the set point (is lower then), the LVDIF bit is set.

Each node in the resister divider represents a “trip point” voltage.

The “trip point” voltage is the minimum supply voltage level at which the device can operate before the LVD module asserts an interrupt. When the supply voltage is equal to the trip point, the voltage tapped off of the resistor array is equal to the voltage generated by the internal voltage reference module. The comparator then generates an interrupt signal setting the LVDIF bit. This voltage is software programmable to any one of 16 values (see Figure 27-2). The trip point is selected by programming the LVDL3:LVDL0 bits (LVDCON<3:0>).

Figure 27-2: Low Voltage Detect (LVD) Block Diagram



27.2 Control Register

The Low Voltage Detect Control register controls the operation of the Low Voltage Detect circuitry.

Register 27-1: LVDCON Register

U-0	U-0	R-0	R/W-0	R/W-0	R/W-1	R/W-0	R/W-1
—	—	IRVST	LVDEN	LVDL3	LVDL2	LVDL1	LVDL0
bit 7							bit 0

bit 7:6 **Unimplemented:** Read as '0'

bit 5 **IRVST:** Internal Reference Voltage Stable Flag bit

1 = Indicates that the Low Voltage Detect logic will generate the interrupt flag at the specified voltage range.
0 = Indicates that the Low Voltage Detect logic will not generate the interrupt flag at the specified voltage range, and LVD interrupt should not be enabled

bit 4 **LVDEN:** Low-voltage Detect Power Enable bit

1 = Enables LVD, powers up LVD circuit
0 = Disables LVD, powers down LVD circuit

bit 3:0 **LVDL3:LVDL0:** Low Voltage Detection Limit bits
The following shows the typical limits for the low voltage detect circuitry. Refer to the device data sheet electrical specifications for the actual tested limit.

1111 = External analog input is used (input comes from the LVDIN pin)
1110 = 4.5V min - 4.77V max.
1101 = 4.2V min - 4.45V max.
1100 = 4.0V min - 4.24V max.
1011 = 3.8V min - 4.03V max.
1010 = 3.6V min - 3.82V max.
1001 = 3.5V min - 3.71V max.
1000 = 3.3V min - 3.50V max.
0111 = 3.0V min - 3.18V max.
0110 = 2.8V min - 2.97V max.
0101 = 2.7V min - 2.86V max.
0100 = 2.5V min - 2.65V max.
0011 = 2.4V min - 2.54V max.
0010 = 2.2V min - 2.33V max.
0001 = 2.0V min - 2.12V max.
0000 = 1.8V min - 1.91V max.

Note 1: LVDL3:LVDL0 modes which result in a trip point below the valid operating voltage of the device are not tested.

2: See the **"Electrical Specifications"** section, parameter 32 in the Device Data Sheet for tested limits.

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

27.3 Operation

The LVD module is useful to add robustness into the application. The device can monitor the state of the device voltage. When the device voltage enters the voltage window near the lower limit of the valid operating voltage range, the device can save values to ensure a "clean" shut-down through the brown-out.

Note:	The system design should be done to ensure that the application software is given adequate time to save values before the device exits the valid operating range or is forced into a Brown-out Reset.

Depending on the power source for the device voltage, the voltage normally decreases relatively slowly. This means that the LVD module does not need to be constantly operating. To decrease the current requirements, the LVD circuitry only needs to be enabled for short periods, where the voltage is checked. After doing the check, the LVD module may be disabled.

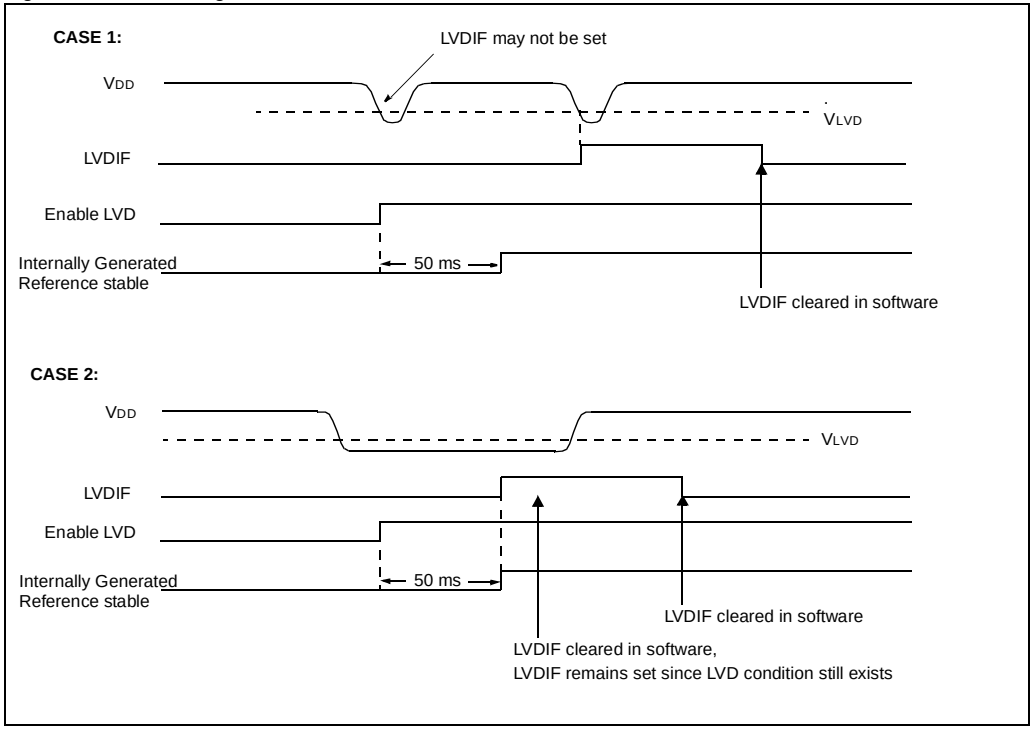
Each time that the LVD module is enabled, the circuitry requires some time to stabilize. After the circuitry has stabilized, all status flags may be cleared. The module will then indicate the proper state of the system.

Steps to setup the LVD module:

1. Write the value to the LVDL3:LVDL0 bits (LVDCON register) which selects the desired LVD Trip Point.
2. Ensure that LVD interrupts are disabled (the LVDIE bit is cleared or the GIE bit is cleared).
3. Enable the LVD module (set the LVDEN bit in the LVDCON register).
4. Wait for the LVD module to stabilize (the IRVST bit to become set).
5. Clear the LVD interrupt flag which may have falsely become set while the LVD module stabilized (clear the LVDIF bit).
6. Enable the LVD interrupt (set the LVDIE and the GIE bits).

Figure 27-3 shows some waveforms that the LVD module may be used to detect.

Figure 27-3: Low Voltage Detect Waveforms



27.3.1 Reference Voltage Set Point

The internal reference voltage of the LVD module may be used by other internal circuitry (e.g., the programmable Brown-out Reset). If these circuits are disabled (lower power consumption), the reference voltage circuit requires stabilization time before a low voltage condition can be reliably detected. This time is specified in electrical specification parameter # 36. The low-voltage interrupt flag will not be enabled until a stable reference voltage is reached. Refer to the timing diagram in Figure 27-3.

27.3.2 Current Consumption

When the module is enabled the LVD comparator and voltage divider are enabled and will consume static current. The voltage divider can be tapped from multiple places in the resistor array. Total current consumption when enabled is specified in electrical specification parameter D022B (typically < 50 µA).

27.4 Operation During SLEEP

When enabled, the LVD circuitry continues to operate during SLEEP. If the device voltage crosses the trip point, the LVDIF bit will be set and the device will wake-up from SLEEP. Device execution will continue from the interrupt vector address, if interrupts have been globally enabled.

27.5 Effects of a RESET

A device RESET forces all registers to their RESET state. This forces the LVD module to be turned off.

27.6 Initialization

Example 27-1 shows an initialization of the LVD module.

Example 27-1: LVD Initialization

```
    MOVLW   0x14            ; Enable LVD, Trip point = 2.5V
    MOVWF   LVDCON          ;
LVD_STABLE
    BTFSS   LVDCON, IRVST   ; Has LVD circuitry stabilized?
    GOTO    LVD_STABLE      ; NO, Wait longer
    BCF     PIR, LVDIF      ; YES, clear LVD interrupt flag
    BSF     PIE, LVDIE      ; Enable LVD interrupt
```

27.7 Design Tips

Question 1: ***The LVD circuitry seems to be generating random interrupts?***

Answer 1:

Ensure that the LVD circuitry is stable before enabling the LVD interrupt. This is done by monitoring the IRVST bit. Once the IRVST bit is set, the LVDIF bit should be cleared and then the LVDIE bit may be set.

Question 2: ***How can I reduce the current consumption of the module?***

Answer 2:

Low Voltage Detect is used to monitor the device voltage. The power source is normally a battery that ramps down slowly. This means that the LVD circuity can be disabled for most of the time, and only enabled occasionally to do the device voltage check.

27.8 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is they may be written for the Base-Line, the Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the LVD module are:

Title	Application Note #
No related application notes at this time	

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

27.9 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Low Voltage Detect module description.

Section 28. WDT and SLEEP Mode

HIGHLIGHTS

This section of the manual contains the following major topics:

28.1 Introduction

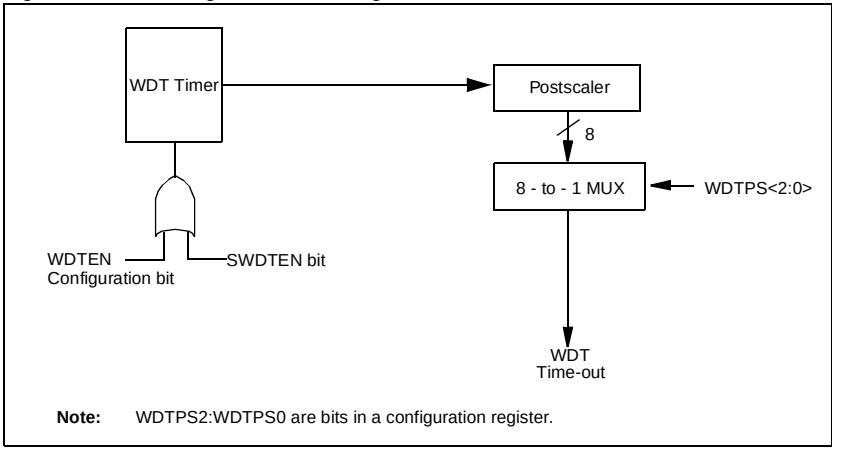
The Watchdog Timer and SLEEP functions are two functions that can enhance the system.

The Watchdog Timer may be used to return to operating mode, or to cause a controller RESET if the program begins to behave erratically. This enhances the overall operation of the system.

The Watchdog Timer (WDT) is a free running on-chip RC oscillator that does not require any external components. The block diagram is shown in Figure 28-1. This RC oscillator is separate from the device RC oscillator of the OSC1/CLKI pin. This means that the WDT will run, even if the clock on the OSC1/CLKI and OSC2/CLKO pins has been stopped, for example, by execution of a SLEEP instruction.

The Watchdog Timer (WDT) is enabled/disabled by a device configuration bit. If the WDT is enabled, software execution may not disable this function. When the WDTEN configuration bit is cleared, the SWDTEN bit enables/disables the operation of the WDT.

Figure 28-1: Watchdog Timer Block Diagram



Note: WDTPS2:WDTPS0 are bits in a configuration register.

The SLEEP function halts controller activity and reduces current consumption to a minimum. The SLEEP mode is a reduced power state, where it is possible to halt almost all activity in the controller. In this mode, power consumption is very low, allowing for long term operation from battery powered applications. Normal operation may be resumed when any of several interrupts occur, the WDT times out, or a RESET occurs.

28.2 Control Register

Register 28-1 shows the WDTCON register. This is a readable and writable register that contains the SWDTEN control bit. If the WDT enable configuration bit has been cleared, this software controlled bit enables or disables the WDT.

Register 28-1: WDTCON Register

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
—	—	—	—	—	—	—	SWDTEN
bit 7							bit 0

bit 7:1 **Unimplemented**: Read as '0'

bit 0 **SWDTEN:** Software Controlled Watchdog Timer Enable bit
1 = Watchdog Timer is on
0 = Watchdog Timer is turned off if the WDTEN configuration bit is '0'

Legend			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared	x = bit is unknown

28.3 Watchdog Timer (WDT) Operation

During normal operation, a WDT time-out generates a device RESET. If the device is in SLEEP mode, a WDT time-out causes the device to wake-up and continue with normal operation. This is known as a WDT wake-up.

The WDT can be permanently enabled by setting the WDTEN configuration bit. If the WDT configuration bit disables the WDT, then software can be used to enable/disable the WDT through setting/clearing the SWDTEN bit.

28.3.1 WDT Period

The WDT has a nominal time-out period of 18 ms with no postscaler (see the **"Electrical Specifications"** section, parameter 31). The time-out period varies with temperature, VDD and process variations from part to part (see DC parameters in the **"Electrical Specifications"** section). If longer time-outs are desired, a postscaler with a division ratio of up to 1:128 can be assigned to the WDT. Thus, time-out periods of up to 2.3 seconds can be realized.

The postscaler assignment is specified at time of device programming through the device configuration bits.

The CLRWDT and SLEEP instructions clear the WDT counter and the WDT postscaler which prevents it from timing out and generating a device RESET.

When a CLRWDT instruction is executed and the prescaler is assigned to the WDT, the prescaler count will be cleared, but the prescaler assignment is not changed.

The $\overline{TO}$ bit in the RCON register will be cleared upon a Watchdog Timer time-out (WDT Reset and WDT wake-up).

28.3.2 Clearing the WDT Counter

The CLRWDT instruction will force the count value of the WDT counter to '0'. When the WDT is disabled (WDTEN configuration bit = '0' and SWDTEN is clear), the WDT counter is forced to '0' and the internal WDT clock source is disabled. Then, when the WDT is enabled (setting the SWDTEN bit when previously cleared), the WDT counter starts from a value of '0'.

28.3.3 WDT Considerations

It should also be taken in account that under worst case conditions (VDD = Minimum, Temperature = Maximum, WDT postscaler = Maximum), it may take several seconds before a WDT time-out occurs.

28.3.4 Effects of a RESET

When a device RESET occurs, the Watchdog Timer counter and postscaler counter are cleared and the $\overline{TO}$ bit is set.

Table 28-1: Summary of Watchdog Timer Registers

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CONFIG2H	—	—	—	—	WDTPS2	WDTPS1	WDTPS0	WDTEN
WDTCON	—	—	—	—	—	—	—	SWDTEN
RCON	IPEN	LWRT	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$

Legend: Shaded cells are not used by the Watchdog Timer.

28.4 SLEEP (Power-Down) Mode

SLEEP (Power-down) mode is the lowest current consumption state and is entered by executing a SLEEP instruction. The device oscillator is turned off, so no system clocks are occurring in the device.

If enabled, the Watchdog Timer will be cleared but keeps running, the $\overline{PD}$ bit in the RCON register is cleared, the $\overline{TO}$ bit is set, and the oscillator driver is turned off. The I/O ports maintain the status they had before the SLEEP instruction was executed (driving high, low, or hi-impedance).

For lowest current consumption in this mode, all I/O pins should be either at V_{DD} or V_{SS}, with no external circuitry drawing current from the I/O pin and modules that are specified to have a delta SLEEP current, should be disabled. I/O pins that are hi-impedance inputs should be pulled high or low externally, to avoid switching currents caused by floating inputs. The contribution from on-chip pull-ups on PORTB should be considered.

During SLEEP, the $\overline{MCLR}$ pin must be at a valid high level.

Some features of the device consume a delta current. These are enabled/disabled by device configuration bits. These features include the Watchdog Timer (WDT), LVD, and Brown-out Reset (BOR) circuitry modules.

28.4.1 Wake-up from SLEEP

There are several ways to wake the controller from SLEEP. The WDT can wake-up the controller when it times out. A RESET will wake the controller and cause the program to restart, and interrupts (from peripherals or external sources) will wake the controller from SLEEP.

The device can wake-up from SLEEP through one of the following events:

1. Any device RESET, such as $\overline{MCLR}$ pin = V_{IL}, V_{DD} = V_{BOR} (if enabled).
2. Watchdog Timer Wake-up (if WDT was enabled).
3. Any peripheral module which can set its interrupt flag while in SLEEP, such as:
   - An external INT pin
   - Change on Port pin
   - Comparators
   - A/D
   - Timer1
   - Timer3
   - LVD
   - MSSP
   - Capture
   - PSP read or write
   - CCP1
   - CCP2
   - Addressable USART
   - PORTB Interrupt on Change
   - External Interrupts
   - Parallel Slave Port
   - Voltage Reference (bandgap)
   - WDT

The first event will RESET the device upon wake-up. However, the latter two events will wake the device and then resume program execution. The $\overline{TO}$ and $\overline{PD}$ bits in the RCON register can be used to determine the cause of device RESET. The $\overline{PD}$ bit, which is set on power-up, is cleared when SLEEP is invoked. The $\overline{TO}$ bit is cleared if WDT time-out occurred (and caused a wake-up).

When the SLEEP instruction is being executed, the next instruction (PC + 2) is pre-fetched. For the device to wake-up through an interrupt event, the corresponding interrupt enable bit must be set (enabled). Wake-up is regardless of the state of the GIE bit. If the GIE bit is clear (disabled), the device continues execution at the instruction after the SLEEP instruction. If the GIE bit is set (enabled), the device executes the instruction after the SLEEP instruction and then branches to the interrupt address. In cases where the execution of the instruction following SLEEP is not desirable, the user should have a NOP after the SLEEP instruction.

28.4.2 Wake-up Using Interrupts

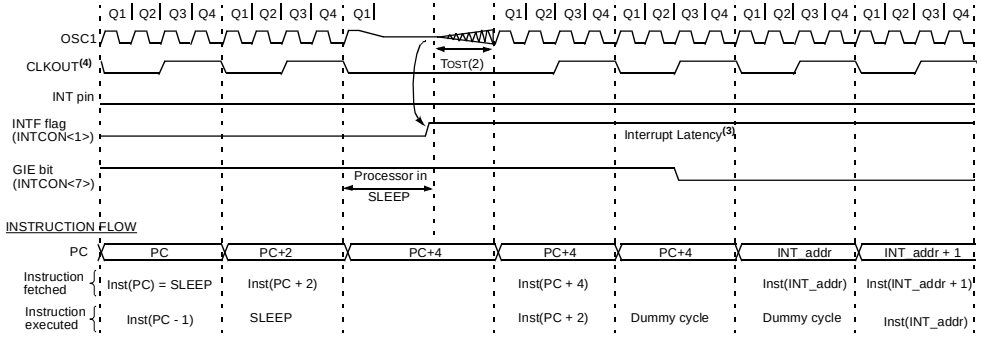
When interrupts are globally disabled (GIE cleared) and any interrupt source has both its interrupt enable bit and interrupt flag bit set, one of the following events will occur:

- If an interrupt condition (interrupt flag bit and interrupt enable bits are set) occurs before the execution of a SLEEP instruction, the SLEEP instruction will complete as a NOP. Therefore, the WDT and WDT postscaler will not be cleared, the $\overline{TO}$ bit will not be set and $\overline{PD}$ bit will not be cleared.
- If the interrupt condition occurs during or after the execution of a SLEEP instruction, the device will immediately wake-up from SLEEP. The SLEEP instruction will be completely executed before the wake-up. Therefore, the WDT and WDT postscaler will be cleared, the $\overline{TO}$ bit will be set and the $\overline{PD}$ bit will be cleared.

Even if the flag bits were checked before executing a SLEEP instruction, it may be possible for flag bits to become set before the SLEEP instruction completes. To determine whether a SLEEP instruction executed, test the $\overline{PD}$ bit. If the $\overline{PD}$ bit is set, the SLEEP instruction was executed as a NOP.

To ensure that the WDT is clear, a CLRWDT instruction should be executed before a SLEEP instruction.

Figure 28-2: Wake-up from SLEEP Through Interrupt



Note 1: XT, HS or LP oscillator mode assumed.
2: TOST = 1024TOSC (drawing not to scale). This delay will not occur for RC and EC osc modes.
3: GIE = '1' assumed. In this case, after wake-up, the processor jumps to the interrupt routine. If GIE = '0', execution will continue in-line.
4: CLKOUT is not available in these osc modes, but shown here for timing reference.

Interrupt sources can wake the controller from SLEEP without actually causing an interrupt. The interrupt source must have its interrupt enable flag set, but GIE does not need to be set. If GIE is clear, the controller will wake without vectoring to an interrupt. If GIE is set, the controller will vector to an interrupt.

If interrupt priority is not used, all interrupt priority bits are set. If interrupt priority is used (any interrupt priority bit is cleared), GIEH controls high priority interrupts and GIEL controls low priority interrupts. Table 28-2 shows the response to the interrupt flag bits depending on the state of the interrupt enable and priority bits.

Table 28-2: SLEEP Mode, Interrupt Enable Bits, and Interrupt Results

Interrupt Source	GIE/GIEH	PEIE/GIEL	Interrupt Priority	Peripheral Interrupt Flag	Response to Interrupt
Any interrupt source that operates during SLEEP	X	X	X	0	SLEEP
	1	0	0 low priority	1	wake
	0	1	1 high priority	1	wake
	0	0	X	1	wake
	1	0	1	1	High priority vector followed
	1	1	0	1	Low priority vector followed

Legend: X is don't care.

28.4.3 Effects of SLEEP Mode on the On-Chip Oscillator

When the device executes a SLEEP instruction, the Watchdog Timer and prescaler counter are cleared (if the WDT is enabled), the on-chip clocks and oscillator are turned off and the controller is held at the beginning of an instruction cycle (Q1 state). With the oscillator off, the OSC1 and OSC2 signals will stop oscillating. Since all the transistor switching currents have been removed, SLEEP mode achieves the lowest current consumption of the device (only leakage currents). Enabling any on-chip feature that will operate during SLEEP will increase the current consumed during SLEEP. The user can wake from SLEEP through external RESET, Brown-out Reset (if enabled), external interrupt, Watchdog Timer time-out or a peripheral interrupt.

Table 28-3: Oscillator Selections, SLEEP Mode, and Waking from SLEEP

OSC Mode	OSC1 Pin in SLEEP	OSC2 Pin in SLEEP	Waking Delays	OSC1 in Run	OSC2 in Run
RC	Floating, pulled high	At logic low	None	R and C set frequency	CLKO (4Tosc)
RCIO	Floating, pulled high	Configured as I/O pin	None	R and C set frequency	Configured as I/O pin
LP	At quiescent voltage level	At quiescent voltage level	TOST (1)	XTAL/res	XTAL/res
XT	At quiescent voltage level	At quiescent voltage level	TOST (1)	XTAL/res	XTAL/res
HS	At quiescent voltage level	At quiescent voltage level	TOST (1)	XTAL/res	XTAL/res
HS w/PLL	At quiescent voltage level	At quiescent voltage level	TOST (1) + TPLL (2)	XTAL/res	XTAL/res
EC	Driven by external clock source	At logic low	None	Driven by external clock source	CLKO (4TOSC)
ECIO	Driven by external clock source	Configured as I/O pin	None	Driven by external clock source	Configured as I/O pin

Note 1: OST (Oscillator Start-up Timer) counts 1024 oscillator cycles before allowing controller clocks to resume. This provides time for the oscillator to start-up and stabilize.

2: A TPLL delay is required to allow the PLL to lock to the oscillator frequency.

28.4.4 Wake-up Delays

Several factors affect how much time the controller requires to return to operating mode from SLEEP. These include oscillator mode and the use of the PLL.

The Oscillator Start-up Timer, OST, counts 1024 oscillator cycles to allow the oscillator to start-up and stabilize before allowing system clocks to resume. The OST is not enabled for RC and EC oscillator modes.

28.4.4.1 Oscillator With PLL Enabled Time-out Sequence After Wake-up

The Oscillator Start-up Timer (OST) provides a 1024 oscillator cycle delay after a wake-up from SLEEP has occurred. 1024 oscillator cycles are not a sufficient amount of time to allow the PLL to lock at high frequencies. An additional TPLL time is required to allow the PLL to lock before allowing system clocks to resume.

28.4.5 Peripheral Module Operation During SLEEP

Table 28-4 gives an overview of which devices operate during SLEEP. For further details, refer to the individual sections in this reference manual.

Table 28-4: Peripheral Modules Active in SLEEP Mode

Peripheral Module	Operates During SLEEP?	Mode of Operation	Wakes from SLEEP?
Timer1, Timer3	Yes	External clock/U.S.C.G., Asynchronous Counter mode	Yes
A/D	Yes	A/D clock = RC clock	Yes
CCP1, CCP2	Yes	Only capture available.	Yes, do not rely on capture value
MSSP	Yes	I^2C – Non-master modes SPI – Slave mode	Yes Yes
Addressable USART	Yes	Synchronous slave mode	Yes
PORTB Interrupt on Change	Yes	All	Yes
External Interrupts	Yes	All	Yes
Parallel Slave Port	Yes	All	Yes
LVD	Yes	All	Yes
Volt Reference (bandgap)	Yes	If required to support LVD, and A/D	No
WDT	Yes	All	Yes

28.4.6 Effects of a WDT Time-out

If the WDT has been enabled, either by the WDTEN configuration bit (='1') or by the SWDTEN bit being set, the WDT will wake-up the controller from SLEEP mode and clear the $\overline{TO}$ bit.

28.4.7 Effects of a Device RESET

When $\overline{MCLR}$ is asserted, $\overline{TO}$ is set and $\overline{PD}$ is clear. All other bits in RCON are unchanged. The controller will resume code execution at the RESET vector address.

28.5 Initialization

No initialization code at this time.

28.6 Design Tips

Question 1: ***My system voltage drops and then returns to the specified device voltage range. The device is not operating correctly and the WDT does not reset and return the device to proper operation.***

Answer 1:

The WDT was not designed to be a recovery from a brown-out condition. It was designed to recover from errant software operation (the device remaining in the specified operating ranges). If your system can be subjected to brown-outs, either the on-chip brown-out circuitry should be enabled or an external brown-out circuit should be implemented.

Question 2: ***Device RESETS even though I do the CLRWDT instruction in my loop.***

Answer 2:

Make sure that the loop with the CLRWDT instruction meets the minimum specification of the WDT (not the typical).

Question 3: ***Device never gets out of RESETS.***

Answer 3:

On power-up, you must take into account the Oscillator Start-up time (Tost). Sometimes it helps to put the CLRWDT instruction at the beginning of the loop, since this start-up time may be variable.

28.7 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is, they may be written for the Base-Line, the Mid-Range or High-End families), but the concepts are pertinent and could be used (with modification and possible limitations). The current application notes related to the WDT and SLEEP Mode are:

Title	Application Note #
Power-up Trouble Shooting	AN607

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

28.8 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Watchdog Timer and SLEEP mode description.

Section 29. Device Configuration Bits

HIGHLIGHTS

This section of the manual contains the following major topics:

29.1 Introduction

The device configuration bits allow each user to customize certain aspects of the device to the needs of the application. When the device powers up, the state of these bits determines the modes that the device uses. Section **29.2 "Configuration Word Bits"** discusses the configuration bits and the modes to which they can be configured. These bits are mapped in program memory locations starting at address 300000h. These locations are accessible during normal device operation.

The configuration bits can be programmed (read as '0') or left unprogrammed (read as '1') to select various device configurations. The ability to change these settings once they have been programmed depends on the memory technology and the package type. This is discussed below:

- Read Only Memory (ROM) devices: These bits are specified at time of ROM code submittal; once the device is masked, these bits can not be changed (would require a new mask code).
- One Time Programmable (OTP) devices: Once the bit is programmed ('0'), it may not be changed.
- Windowed EPROM devices: Once these bits are programmed ('0'), the device must be UV erased to return the configuration word to the erased state. UV erasing the device also erases the program memory. Window devices are for debugging purposes.

Note:	Microchip does not recommend code-protecting windowed devices.

- FLASH devices: These bits may be erased and reprogrammed.

29.2 Configuration Word Bits

These configuration bits specify some of the device modes and are programmed by a device programmer, or by using the In-Circuit Serial Programming™ (ICSP) feature of the Enhanced Architecture devices. The placement of these configuration bits is automatically handled when you select the device in your device programmer. The desired state of the configuration bits may be specified in the source code (dependent on the language tool used), or through the programming interface. After the device has been programmed, the application software may read the configuration bit values through the Table read instructions. For additional information, please refer to the Programming Specification of the device.

Note 1: Always ensure that your device programmer has the same device selected as you are programming.

2: Microchip recommends that the desired configuration bit states be embedded into the application source code. This is easily done in the MPASM assembler by the use of the CONFIG directive. See Subsection **29.2.1 "MPASM's CONFIG Directive."**

Register 29-1: Configuration Bits

P/R-u
Bit placement is device dependent

A19DIS: Disable A19:A16 Drivers bit

This bit disables the A19:A16 address lines of the system bus so that these pins may be used as general purpose I/O.

1 = Drivers enabled
0 = Drivers disabled

A15DIS: Disable AD15:AD12 Drivers bit

This bit disables the AD15:AD12 address lines of the system bus so that these pins may be used as general purpose I/O. The AD15:AD12 address lines may only be disabled when the system bus is configured for 8-bit data (BW16 = '0').

When BW16 = '1'
1 = Drivers enabled (state of A15DIS is ignored)
0 = Drivers enabled (state of A15DIS is ignored)

When BW16 = '0'
1 = Drivers enabled
0 = Drivers disabled

A11DIS: Disable AD11:AD8 Drivers bit

This bit disables the AD11:AD8 address lines of the system bus so that these pins may be used as general purpose I/O. The AD11:AD8 address lines may only be disabled when the system bus is configured for 8-bit data (BW16 = '0').

When BW16 = '1'
1 = Drivers enabled (state of A11DIS is ignored)
0 = Drivers enabled (state of A11DIS is ignored)

When BW16 = '0'
1 = Drivers enabled
0 = Drivers disabled

BADIS: Byte Address BA0 Disable bit

1 = Drivers enabled
0 = Drivers disabled

BSDIS: Byte Select $\overline{UB}$,$\overline{LB}$ Disable bit

1 = Drivers enabled
0 = Drivers disabled

BOREN: Brown-out Reset Enable bit

1 = Brown-out Reset enabled
0 = Brown-out Reset disabled

BORV1:BORV0: Brown-out Reset Voltage bits

These bits specify the trip point for the Brown-out Reset circuitry. The values shown below are for a typical device. Please refer to the device data sheet **"Electrical Specifications"** section for the tested range.

11 = V_{BOR} set to 2.5V or 1.8V (device dependent)
10 = V_{BOR} set to 2.7V
01 = V_{BOR} set to 4.2V
00 = V_{BOR} set to 4.5V

BW16: 16-bit Bus Width bit

This bit specifies the data width of the system bus.

1 = System Bus has 16-bit data bus
0 = System Bus has 8-bit data bus

CCP2MX: CCP2 Mux bit

1 = CCP2 input/output is multiplexed with an I/O
0 = CCP2 input/output is multiplexed with a different I/O

CP: Code Protection bits (apply when in Code Protected Microcontroller Mode)

1 = Program memory code protection off
0 = All of program memory code protected

DP: Data EEPROM Memory Code Protection bit

1 = Code protection off
0 = Data EEPROM memory is code protected

Note: This bit is used when a ROM program memory device or a ROMless device has Data EEPROM memory.

FOSC2:FOSC0: Oscillator Selection bits

111 = RC oscillator w/ OSC2 configured as I/O
110 = HS oscillator with PLL enabled/clock frequency = (4 x FOSC1)
101 = EC oscillator w/ OSC2 configured as I/O
100 = EC oscillator w/ OSC2 configured as divide by 4 clock output
011 = RC oscillator
010 = HS oscillator
001 = XT oscillator
000 = LP oscillator

$\overline{\text{OSCSEN}}$: Oscillator System Clock Switch Enable bit

1 = Oscillator system clock switch option is disabled (main oscillator is source)
0 = Oscillator system clock switch option is enabled (oscillator switching is enabled)

PM1:PM0: Processor Mode Select bits

These bits select the processor operating mode for the device. The processor operating mode specifies how the program memory is mapped (internal/external) and the default configuration of the system bus pins.

11 = Microprocessor mode
10 = Microcontroller mode
01 = Reserved
00 = Extended microcontroller mode

$\overline{\text{PWRTEN}}$: Power-up Timer Enable bit

1 = PWRT disabled
0 = PWRT enabled

STVREN: Stack Full/Underflow Reset Enable bit

1 = Stack Full/Underflow will cause RESET
0 = Stack Full/Underflow will not cause RESET

WAIT: System Bus Wait bit

This bit is used to enable wait states for table reads and table writes to external memory on the system bus
1 = Wait selections unavailable, device will not wait
0 = Wait programmed by WAIT1 and WAIT0 bit in MEMCON register

WDIS: Write Select $\overline{WRH}$, $\overline{WRL}$ Disable bit

1 = Drivers enabled
0 = Drivers disabled

WDTPS2:WDTPS0: Watchdog Timer Postscale Select bits

111 = 1:1
110 = 1:2
101 = 1:4
100 = 1:8
011 = 1:16
010 = 1:32
001 = 1:64
000 = 1:128

WDTEN: Watchdog Timer Enable bit

1 = WDT enabled
0 = WDT disabled (control is placed on the SWDTEN bit, in register WDTCON)

Legend		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as '0'
- n = Value at POR reset	'1' = bit is set	'0' = bit is cleared x = bit is unknown

Note 1: The position of the configuration bits is device dependent. Please refer to the device programming specification for bit placement. You are not required to know the configuration bit positions when using a Microchip device programmer. This is addressed by either the configuration direction of the software tool or MPLAB's user interface.

2: In ROMless devices, some of the system bus configuration bits are hardwired into a set configuration. Other bits may be placed in protected SFR locations which can only be modified after properly writing to the CMLK1:CMLK0 bits.

29.2.1 MPASM's CONFIG Directive

Microchip's assembler, MPASM, has a nice feature (directives) that allows you to specify the device configuration in the source code file. This ensures that when programming a device for an application, the required configuration is also programmed. This minimizes the risk of programming the wrong device configuration and then wondering why it no longer works in the application (unexpected operation).

Example 29-1 shows a template for using the CONFIG directive.

Example 29-1: Using the CONFIG Directive, a Source File Template

```
     LIST   p = p18C452      ; List Directive,
;    Revision History
;
     #INCLUDE    <P18C452.INC>    ; Microchip Device Header File
;
     #INCLUDE    <MY_STD.MAC>    ; File which includes my standard macros
     #INCLUDE    <APP.MAC>       ; File which includes macros specific
                                 ;   to this application
;
; Specify Device Configuration Bits for
; Program Configuration Registers 0 through 6
     __CONFIG    _CONFIG0, CP_OFF_0
     __CONFIG    _CONFIG1, LPSCEN_OFF_1 & RCRA6_OSC_1
     __CONFIG    _CONFIG2, BORV_25_2 & BOREN_ON_2 & PWRTEN_OFF_2
     __CONFIG    _CONFIG3, WDPS_128_3 & WDT_ON_3
     __CONFIG    _CONFIG5, CCP2MX_ON_5
     __CONFIG    _CONFIG6, SVTREN_ON_6
;
     org   0x00            ; Start of Program Memory
RESET_ADDR   :             ; First instruction to execute after a reset


     end
```

The symbols currently in the Microchip Device Header files make using the CONFIG directive straight forward. These are shown in Table 29-1. The symbols available for your device are listed in the Microchip Include file for that device.

Note:	As long as the correct device is specified (in the LIST and INCLUDE file directives), the correct polarity of all bits is ensured.

Table 29-1: _CONFIG Directive Symbols (from Microchip Header Files) (1)

Feature	Configuration Byte	Symbols	Comment
Oscillators	_CONFIG1H	_LP_OSC_1	LP mode for device
		_XT_OSC_1	XT mode for device
		_HS_OSC_1	HS mode for device oscillator
		_RC_OSC_1	RC mode for device oscillator
		_RCIO_OSC_1	RC mode for device oscillator with clockout configured as I/O pin
		_EC_OSC_1	EC (external clock) mode for device oscillator
		_ECIO_OSC_1	EC (external clock) mode for device oscillator with clockout configured as I/O pin
		_HS4_OSC_1	HS mode with 4 x PLL for device oscillator
		_HSPLL_OSC_1	HS mode with 4 x PLL for device oscillator (4)
Oscillator Switch	_CONFIG1H	_LPSCEN_ON_1	Oscillator switch enabled
		_LPSCEN_OFF_1	Oscillator switch disabled
Code Protect	_CONFIG1L	_CP_ON_0	Code Protect enabled
		_CP_OFF_0	Code Protect disabled
Watchdog Timer	_CONFIG2H	_WDT_ON_3	Watchdog Timer enabled
		_WDT_OFF_3	Watchdog Timer disabled
Watchdog Timer Postscale Assignment	_CONFIG2H	_WDTPS_128_3	WDT prescaler set to 1:128
		_WDTPS_64_3	WDT prescaler set to 1:64
		_WDTPS_32_3	WDT prescaler set to 1:32
		_WDTPS_16_3	WDT prescaler set to 1:16
		_WDTPS_8_3	WDT prescaler set to 1:8
		_WDTPS_4_3	WDT prescaler set to 1:4
		_WDTPS_2_3	WDT prescaler set to 1:2
		_WDTPS_1_3	WDT prescaler set to 1:1
Power-up Timer	_CONFIG2L	_PWRTEN_ON_2	Power-up Timer enabled
		_PWRTEN_OFF_2	Power-up Timer disabled
Brown-out Reset	_CONFIG2L	_BOREN_ON_2	Brown-out Reset enabled
		_BOREN_OFF_2	Brown-out Reset disabled
BOR Trip-Point Voltage	_CONFIG2L	_BORV_18_2	BOR trip point = 1.8V min (2, 3)
		_BORV_25_2	BOR trip point = 2.5V min (2, 3)
		_BORV_27_2	BOR trip point = 2.7V min (3)
		_BORV_42_2	BOR trip point = 4.2V min (3)
		_BORV_45_2	BOR trip point = 4.5V min (3)

Note 1: Not all configuration bit symbols may be available on any one device. Please refer to the Microchip include file of that device for available symbols.

2: The option for a 1.8V or 2.5V BOR trip point is device dependent. Only one of these symbols will be available in the Microchip supplied header file.

3: These are the trip points for a typical device. Other trip points (symbols) may be specified for the device.

4: Symbol obsoleted and may not be available in header file.

Table 29-1: _CONFIG Directive Symbols (from Microchip Header Files) (1) (Continued)

Feature	Configuration Byte	Symbols	Comment
CCP2 pin Multiplex	_CONFIG3H	_CCP2MX_ON_5	CCP2 out multiplexer enabled
		_CCP2MX_OFF_5	CCP2 out multiplexer disabled
Processor Mode	_CONFIG3L	_MP_MODE_4	Microprocessor mode
		_MC_MODE_4	Microcontroller mode
		_XMC_MODE_4	Extended Microcontroller mode
External Data Bus Wait	_CONFIG3L	_WAIT_ON_4	Wait on external data bus enabled
		_WAIT_OFF_4	Wait on external data bus disabled
System Bus A19 Disable	_CONFIG4H	_A19DIS_ON_7	System bus A19 disabled
		_A19DIS_OFF_7	System bus A19 enabled
System Bus A15 Disable	_CONFIG4H	_A15DIS_ON_7	System bus A15 disabled
		_A15DIS_OFF_7	System bus A15 enabled
System Bus A11 Disable	_CONFIG4H	_A11DIS_ON_7	System bus A11 disabled
		_A11DIS_OFF_7	System bus A11 enabled
Byte Address BA0 Disable	_CONFIG4H	_BADIS_ON_7	Byte Address BA0 disabled
		_BADIS_OFF_7	Byte Address BA0 enabled
Byte Select Disable	_CONFIG4H	_BSDIS_ON_7	Byte Select disabled
		_BSDIS_OFF_7	Byte Select enabled
Write Select Disable	_CONFIG4H	_WDIS_ON_7	Write Select disabled
		_WDIS_OFF_7	Write Select enabled
Stack Full/Overflow	_CONFIG4L	_STVREN_ON_6	Stack full/overflow rest enabled
		_STVREN_OFF_6	Stack full/overflow rest disabled
Code Protect Data EEPROM	TBD	_DP_ON	Data EEPROM protect enabled
		_DP_OFF	Data EEPROM protect disabled

Note 1: Not all configuration bit symbols may be available on any one device. Please refer to the Microchip include file of that device for available symbols.

2: The option for a 1.8V or 2.5V BOR trip point is device dependent. Only one of these symbols will be available in the Microchip supplied header file.

29.3 Program Verification/Code Protection

If the code protection bit(s) have not been programmed, the on-chip program memory can be read out for verification purposes.

Note:	Microchip does not recommend code protecting windowed devices.

29.3.1 ROM Devices

When a ROM device also has Data EEPROM memory, an additional code protect configuration bit may be implemented. The program memory configuration bit is submitted as part of the ROM code submittal. The Data EEPROM memory code protect configuration bit will be an EEPROM bit. When ROM devices complete testing, the EEPROM data memory code protect bit will be programmed to the same state as the program memory code protect bit. That is, Data EEPROM code protect is off when program memory code protect is off and Data EEPROM code protect is on for all other selections.

In applications where the device is code protected and the Data EEPROM needs to be programmed before the application can be released, the Data EEPROM memory must have the entire Data EEPROM memory erased. The device programming specification details the steps to do this. Microchip device programmers implement the specified sequence. Once this sequence is complete, the Data EEPROM memory code protect is disabled. This allows the desired data to be programmed into the device. After programming the Data EEPROM memory array, the Data EEPROM memory code protect configuration bit should be programmed as desired.

29.4 ID Locations

Five memory locations (200000h - 200004h) are designated as ID locations where the user can store checksum or other code-identification numbers. These locations are accessible during normal execution through the TBLRD instruction, or during program/verify. The ID locations can be read when the device is code protected.

29.5 Device ID

One memory location (two bytes) (3FFFFEh-3FFFFFh) is designated as the Device ID location. The value at this location is specified by Microchip and is useful in determining the device.

Device ID bits can be used by a device programmer to retrieve information about what device is being programmed and what the revision of the device is.

The Device ID can be accessed by a TBLRD instruction or via serial program/verify. The Device ID can be read when the part is code protected.

The 5 LSbs are the device revision information, and the remaining 11 bits contain the device ID number. This is shown in Table 29-2.

Table 29-2: Device ID Registers

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DEVID2	DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3
DEVID1	DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0

29.6 Design Tips

Question 1: ***I have a JW device and I can no longer program it (reads scrambled data or all '0's). What's wrong with the device?***

Answer 1:

Nothing. You probably code protected the device. If this is the case, the device is no longer usable. See Section **29.3 "Program Verification/Code Protection"** for more details.

Question 2: ***In converting from a PIC16C74 to a PIC18C452, my application no longer works.***

Answer 2:

1. Did you re-assemble the source file specifying the PIC18C452 in the INCLUDE file and LIST directives? The use of the CONFIG directive is highly recommended.
2. On the device programmer, did you specify the PIC18C452, and were all the configuration bits as desired?

Question 3: ***When I erase the device, the program memory is blank but the configuration word is not yet erased.***

Answer 3:

That is by design. Also remember that Microchip does not recommend code protecting windowed devices.

29.7 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced MCU family (that is, they may be written for the Base-Line, Mid-Range or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to Configuration Word are:

Title	Application Note #
No related Application Notes at this time.	

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

29.8 Revision History

Revision A

This is the initial released revision of the Configuration Word description.

Section 30. In-Circuit Serial Programming™ (ICSP™)

HIGHLIGHTS

This section of the manual contains the following major topics:

30.1 Introduction

All Enhanced MCU devices can be In-Circuit Serial Programmed (ICSP™) while in the end application circuit. This is simply done with two lines for clock and data, and three other lines for power, ground and the programming voltage.

In-Circuit Serial Programming (ICSP™) is a great way to reduce your inventory overhead and time-to-market for your product. By assembling your product with a blank Microchip microcontroller (MCU), you can stock one design. When an order has been placed, these units can be programmed with the latest revision of firmware, tested and shipped in a very short time. This method also reduces scrapped inventory due to old firmware revisions. This type of manufacturing system can also facilitate quick turnarounds on custom orders for your product.

Most people would think to use ICSP with PICmicro® OTP MCUs only on an assembly line where the device is programmed once. However, there is a method by which an OTP device can be programmed several times depending on the size of the firmware. This method, explained in **Section 30.7**, provides a way to field upgrade your firmware in a way similar to EEPROM- or FLASH-based devices.

30.2 Entering In-Circuit Serial Programming Mode

The device is placed into a program/verify mode by holding the CLOCK (typically on the RB6 pin) and DATA (typically on the RB7 pin) pins low, while raising the MCLR (V_{PP}) pin from V_{IL} to V_{IHH} (see programming specification) and having V_{DD} at the programming voltage. Both the CLOCK and DATA pins are Schmitt Trigger inputs in this mode. When in I/O mode and RB7 is driving data, it is a CMOS output driver.

After RESET, to place the device into programming/verify mode, the program counter (PC) is at location 00h. A command is then supplied to the device. Some commands then specify that 16-bits of program data are then supplied to or read from the device, depending on whether the command was a load or a read. For complete details of serial programming, please refer to the device specific Programming Specifications.

During the In-Circuit Serial Programming Mode, the WDT circuitry is disabled from generating a device RESET.

30.3 Application Circuit

The application circuit must be designed to allow all the programming signals to be directly connected to the PICmicro MCU. Figure 30-1 shows a typical circuit that is a starting point when designing with ICSP. The application must compensate for the following:

1. Isolation of the $\overline{\text{MCLR}}$/VPP pin from the rest of the circuit
2. Loading of pins CLOCK and DATA
3. Capacitance on each of the VDD, $\overline{\text{MCLR}}$/VPP, CLOCK and DATA pins
4. Minimum and maximum operating voltage for VDD
5. PICmicro oscillator

30.3.1 Isolation of the $\overline{\text{MCLR}}$/VPP Pin from the Rest of the Circuit

The $\overline{\text{MCLR}}$/VPP pin is normally connected to an RC circuit. The pull-up resistor is tied to VDD and a capacitor is tied to ground. This circuit can affect the operation of ICSP depending on the size of the capacitor, since the VPP voltage must be isolated from the rest of the circuit . The resistor (R1) should be greater than 10kΩ to provide isolation between VDD and VPP. It is, therefore, recommended that the circuit in Figure 30-1 be used when an RC is connected to $\overline{\text{MCLR}}$/VPP. Another consideration with $\overline{\text{MCLR}}$/VPP is that when the PICmicro device is programmed, this pin is driven to approximately 13V and also to ground. Therefore, the application circuit must be isolated from this voltage provided by the programmer.

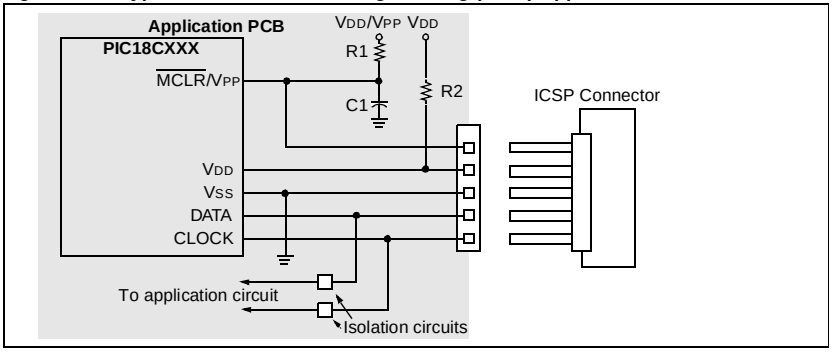
30.3.2 Loading of Pins CLOCK and DATA

The CLOCK and DATA pins are used by the PICmicro MCU for serial programming. CLOCK is driven by the programmer. DATA is a bi-directional pin that is driven by the programmer when programming, and driven by the MCU when verifying. These pins must be isolated from the rest of the application circuit so as not to affect the signals during programming. You must take into consideration the output impedance of the programmer when isolating CLOCK and DATA from the rest of the circuit. This isolation circuit must account for CLOCK being an input on the MCU and for DATA being bi-directional (can be driven by both the MCU and the programmer). For instance, PRO MATE® II has an output impedance of 1kΩ. If the design permits, these pins should not be used by the application. This is not the case with most applications, so it is recommended that the designer evaluate whether these signals need to be buffered. As a designer, you must consider what type of circuitry is connected to CLOCK and DATA and then make a decision on how to isolate these pins. Figure 30-1 does not show any circuitry to isolate CLOCK and DATA on the application circuit, because this is very application dependent.

To simplify this interface, the optimal usage of these I/O pins in the application are (in order):

1. Dedicate the CLOCK/DATA pins for the ICSP interface (not connected to other circuitry).
2. Use these pins as outputs with minimal loading on the signal line.
3. Use isolation circuitry so these signals can be driven to the ICSP specifications.

Figure 30-1: Typical In-Circuit Serial Programming (ICSP) Application Circuit



30.3.3 Capacitance on Each of the VDD, MCLR/VPP, CLOCK and DATA Pins

The total capacitance on the programming pins affects the rise rates of these signals as they are driven out of the programmer. Typical circuits use several hundred microfarads of capacitance on VDD, which helps to dampen noise and ripple. However, this capacitance requires a fairly strong driver in the programmer to meet the rise rate timings for VDD. Most programmers are designed to simply program the MCU itself and don't have strong enough drivers to power the application circuit. One solution is to use a driver board between the programmer and the application circuit. The driver board requires a separate power supply that is capable of driving the VPP and VDD pins with the correct rise rates and should also provide enough current to power the application circuit. CLOCK and DATA are not buffered on this schematic, but may require buffering depending upon the application. A sample driver board schematic is shown in Figure 30-2.

Note: The driver board design MUST be tested in the user's application to determine the effects of the application circuit on the programming signals timing. Changes may be required if the application places a significant load on the VDD, VPP, CLOCK or DATA pins.

30.3.4 Minimum and Maximum Operating Voltage for VDD

The Microchip programming specification states that the device should be programmed at 5V. Special considerations must be made if your application circuit operates at 3V only. These considerations may include totally isolating the MCU during programming. Another consideration is the device must be verified at the minimum and maximum voltages at which the application circuit will be operating. For instance, a battery operated system may operate from three 1.5V cells giving an operating voltage range of 2.7V to 4.5V. The programmer must program the device at 5V and must verify the program memory contents at both 2.7V and 4.5V to ensure that proper programming margins have been achieved. This ensures the PICmicro MCU operation over the voltage range of the system.

30.3.5 PICmicro Oscillator

The final consideration deals with the oscillator circuit on the application board. The voltage on MCLR/VPP must rise to the specified program mode entry voltage before the device executes any code. The crystal modes available on the device are not affected by this, because the Oscillator Start-up Timer waits for 1024 oscillations before any code is executed. However, RC or EC oscillators do not require any start-up time; therefore, the Oscillator Start-up Timer is not used. The programmer must drive MCLR/VPP to the program mode entry voltage before the RC or EC oscillator toggles four times. If the RC or EC oscillator toggles four or more times, the program counter will be incremented to some value X. When the device enters programming mode, the program counter will not be zero and the programmer will start programming your code at an offset of X. There are several alternatives that can compensate for a slow rise rate on MCLR/VPP. The first method is to not populate the resistor (R1) in Figure 30-1, program the device, and then insert the resistor (R1). The other method is to have the programming interface drive the OSC1 pin of the PICmicro MCU to ground while programming. This will prevent any oscillations from occurring during programming.

Connecting the application circuit to the programmer is dependent on the programming environment. Refer to **Section 30.5 "Programming Environment"** for more details.

30.4 Programmer

PIC18CXXX MCUs only use serial programming and, therefore, all programmers supporting these devices will support ICSP. One area of consideration with the programmer is the drive capability. As discussed before, it must be able to provide the specified rise rates on the ICSP signals and also provide enough current to power the application circuit. Figure 30-2 shows an example driver board. This driver schematic does not show any buffer circuitry for CLOCK and DATA. It is recommended that an evaluation be performed to determine if buffering is required. Another consideration with the programmer is what VDD levels are used to verify the memory contents of the device. For instance, the PRO MATE II verifies program memory at the minimum and maximum VDD levels for the specified device and is, therefore, considered a production quality programmer. On the other hand, the PICSTART® Plus only verifies at 5V and is for prototyping use only. The Microchip programming specifications state that the program memory contents should be verified at both the minimum and maximum VDD levels that the application circuit will be operating. This implies that the application circuit must be able to handle the varying VDD voltages.

There are also several third party programmers that are available. You should select a programmer based on the features it has and how it fits into your programming environment. The *Microchip Development Systems Ordering Guide* (DS30177) provides detailed information on all our development tools. The *Microchip Third Party Guide* (DS00104) provides information on all of our third party tool developers. Please consult these two references when selecting a programmer. Many options exist, including serial or parallel PC host connection, stand-alone operation, and single or gang programmers. Some of the third party developers include Advanced Transdata Corporation, BP Microsystems, Data I/O, Emulation Technology, and Logical Devices.

30.5 Programming Environment

The programming environment affects the type of programmer used, the programmer cable length and the application circuit interface. Some programmers are well suited for a manual assembly line, while others are desirable for an automated assembly line. You may want to choose a gang programmer to program multiple systems at a time.

The physical distance between the programmer and the application circuit affects the load capacitance on each of the programming signals. This directly affects the drive strength needed to provide the correct signal rise rates and current. This programming cable must also be as short as possible and properly terminated and shielded, or the programming signals may be corrupted by ringing or noise.

Finally, the application circuit interface to the programmer depends on the size constraints of the application circuit itself and the assembly line. A simple header can be used to interface the application circuit to the programmer. This might be more desirable for a manual assembly line where a technician plugs the programmer cable into the board. A different method uses spring loaded test pins (commonly referred to as pogo pins). The application circuit has pads on the board for each of the programming signals, and there is a fixture that has pogo pins in the corresponding configuration. The application circuit or fixture is moved into position, such that the pogo pins come into contact with the board. This method might be more suitable for an automated assembly line.

After taking into consideration the various points with the application circuit, the programmer and the programming environment, anyone can build a high quality, reliable manufacturing line based on ICSP.

30.6 Other Benefits

ICSP provides other benefits, such as calibration and serialization.

If program memory permits, it would be cheaper and more reliable, to store calibration constants in program memory instead of using an external serial EEPROM. For example, if your system has a thermistor that can vary from one system to another, storing some calibration information in a table format allows the microcontroller to compensate (in software) for external component tolerances. System cost can be reduced without affecting the required performance of the system by using software calibration techniques. But how does this relate to ICSP? The PICmicro MCU has already been programmed with firmware that performs a calibration cycle. The calibration data is transferred to a calibration fixture. When all calibration data has been transferred, the fixture places the PICmicro MCU in programming mode and programs the PICmicro MCU with the calibration data. Application note *AN656, "In-Circuit Serial Programming of Calibration Parameters Using a PICmicro Microcontroller*," shows exactly how to implement this type of calibration data programming.

The other benefit of ICSP is serialization. Each individual system can be programmed with a unique or random serial number. One such application of a unique serial number would be for security systems. A typical system might use DIP switches to set the serial number. Instead, this number can be burned into program memory, thus reducing the overall system cost and lowering the risk of tampering.

30.7 Field Programming of PICmicro OTP MCUs

An OTP device is not normally capable of being reprogrammed, but the PICmicro architecture gives you this flexibility, provided the size of your firmware is at least half that of the desired device, and the device is not code protected. If your target device does not have enough program memory, Microchip provides a wide spectrum of devices from 0.5K to 16K word program memory with the same set of peripheral features that will help meet the criteria.

The Enhanced MCU devices have three vectors; RESET and two interrupt vector addresses. When the PICmicro device encounters a RESET or interrupt condition, the code located at one of these locations in program memory is executed.

For an example of reprogramming an OTP device, we will use an example from our Mid-Range family. This technology is applicable to all EPROM based PICmicro devices. The first listing of Example 30-1 shows the code that is first programmed into the PICmicro device. The second listing of Example 30-1 shows the code that is programmed into the PICmicro device for the second time.

Example 30-1 shows that to program the device a second time, the memory location 0x0000 (originally `goto Main` (0x2808)), is reprogrammed to all 0's. This happens to be a `NOP` instruction. This location cannot be reprogrammed to the new opcode (0x2860), because the bits that are 0's cannot be reprogrammed to 1's. Only bits that are 1's can be reprogrammed to 0's. The next memory location, 0x0001, was originally blank (all 1's) and now becomes a `goto Main` (0x2860). When a RESET condition occurs, the MCU executes the instruction at location 0x0000, which is the `NOP` (a completely benign instruction), and then executes the `goto Main` to start the execution of code. The example also shows that all program memory locations after 0x005A are blank in the original program, so that the second time the PICmicro device is programmed, the revised code can be programmed at these locations. The same descriptions can be given for the interrupt vector locations.

Now your one-time programmable Enhanced MCU is exhibiting EEPROM- or FLASH-like qualities.

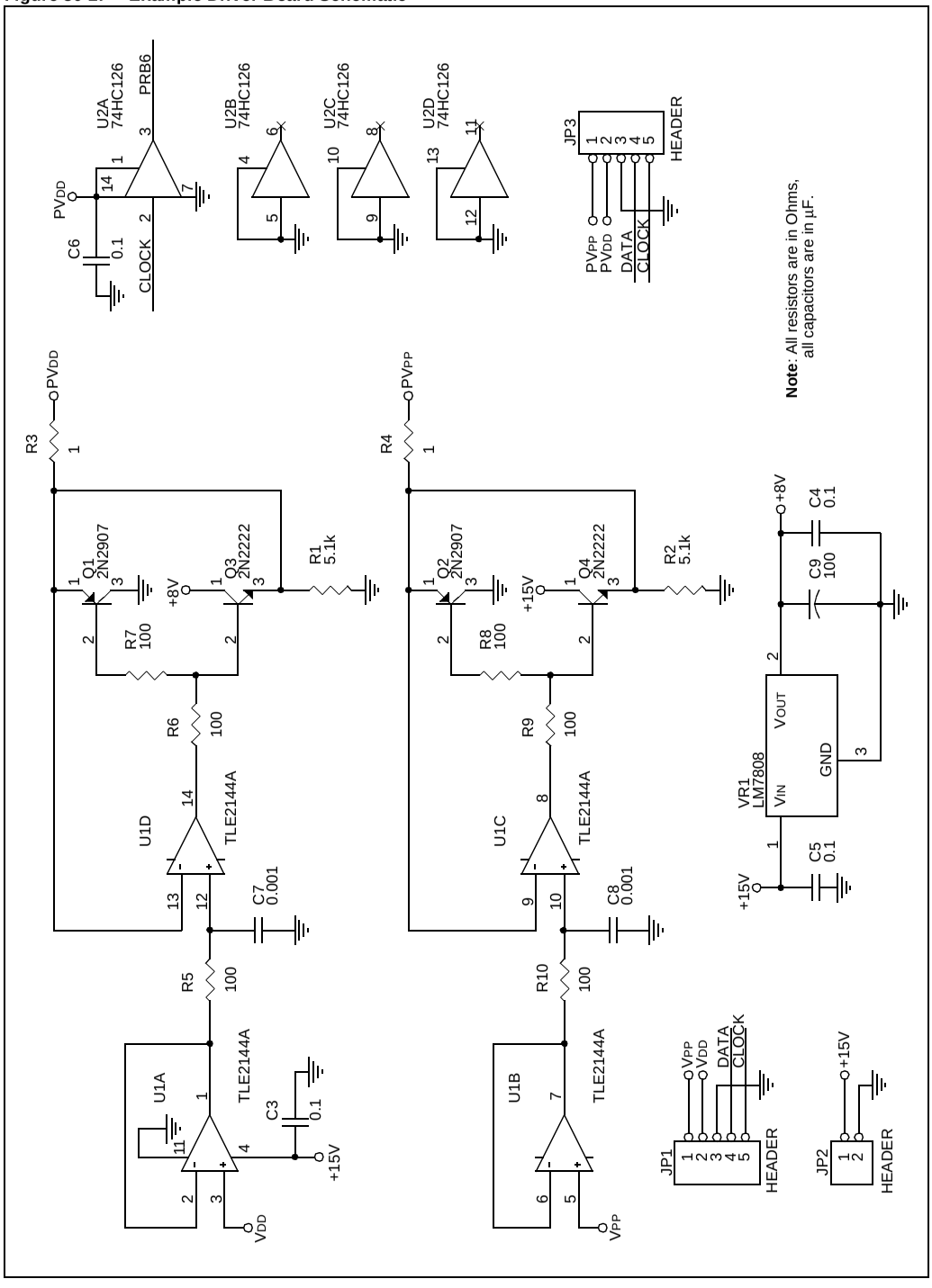
Example 30-1: Programming Cycle Listing Files

```
        First Program Cycle                                  Second Program Cycle
_____________________________________________________________________________________________
Prog    Opcode    Assembly                        |  Prog   Opcode  Assembly
Mem               Instruction                     |  Mem            Instruction
-------------------------------------------------------------------------------------------
0000    2808      goto  Main        ;Main loop    |  0000   0000    nop
0001    3FFF      <blank>           ; at 0x0008   |  0001   2860    goto  Main; Main now
0002    3FFF      <blank>                         |  0002   3FFF    <blank>  ;   at 0x0060
0003    3FFF      <blank>                         |  0003   3FFF    <blank>
0004    2848      goto  ISR         ; ISR at      |  0004   0000    nop
0005    3FFF      <blank>           ;   0x0048    |  0005   28A8    goto  ISR ; ISR now at
0006    3FFF      <blank>                         |  0006   3FFF    <blank>  ;   0x00A8
0007    3FFF      <blank>                         |  0007   3FFF    <blank>
0008    1683      bsf   STATUS,RP0                |  0008   1683    bsf   STATUS,RP0
0009    3007      movlw 0x07                      |  0009   3007    movlw 0x07
000A    009F      movwf ADCON1                    |  000A   009F    movwf ADCON1
 .                                                |   .
 .                                                |   .
 .                                                |   .
0048    1C0C      btfss PIR1,RBIF                 |  0048   1C0C    btfss PIR1,RBIF
0049    284E      goto  EndISR                    |  0049   284E    goto  EndISR
004A    1806      btfsc PORTB,0                   |  004A   1806    btfsc PORTB,0
 .                                                |   .
 .                                                |   .
 .                                                |   .
0060    3FFF      <blank>                         |  0060   1683    bsf   STATUS,RP0
0061    3FFF      <blank>                         |  0061   3005    movlw 0x05
0062    3FFF      <blank>                         |  0062   009F    movwf ADCON1
 .                                                |   .
 .                                                |   .
 .                                                |   .
00A8    3FFF      <blank>                         |  00A8   1C0C    btfss PIR1,RBIF
00A9    3FFF      <blank>                         |  00A9   28AE    goto  EndISR
00AA    3FFF      <blank>                         |  00AA   1806    btfsc PORTB,0
 .                                                |   .
 .                                                |   .
 .                                                |   .
-------------------------------------------------------------------------------------------
```

30.8 Field Programming of FLASH PICmicro MCUs

With the ICSP interface circuitry already in place, FLASH-based PICmicro MCUs can be easily reprogrammed in the field. These FLASH devices allow you to reprogram them even if they are code protected. A portable ICSP programming station might consist of a laptop computer and programmer. The technician plugs the ICSP interface cable into the application circuit and downloads the new firmware into the device. The next thing you know, the system is up and running without those annoying “bugs.” Another instance would be that you want to add an additional feature to your system. All of your current inventory can be converted to the new firmware, and field upgrades can be performed to bring your installed base of systems up to the latest revision of firmware.

Figure 30-2: Example Driver Board Schematic



30.9 Design Tips

Question 1: ***When I try to do ICSP, the entire program is shifted (offset) in the device program memory.***

Answer 1:

If the $\overline{\text{MCLR}}$ pin does not rise fast enough, while the device's voltage is in the valid operating range, the internal Program Counter (PC) can increment. This means that the PC is no longer pointing to the address that you expected. The exact location depends on the number of device clocks that occurred in the valid operating region of the device.

Question 2: ***I am using a PRO MATE II with a socket that I designed to bring the programming signal to my application board. Sometimes when I try to do ICSP, the program memory is programmed wrong.***

Answer 2:

The voltages / timings may be violated at the device. This could be due to the:

- Application board circuitry
- Cable length from programmer to target
- Large capacitance on V_{DD} that affects levels / timings

30.10 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is, they may be written for the Base-Line, the Mid-Range or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to In-Circuit Serial Programming are:

Title	Application Note #
In-Circuit Serial Programming of Calibration Parameters using a PICmicro® Microcontroller	AN656
In-Circuit Serial Programming Guide	DS30277

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

30.11 Revision History

Revision A

This is the initial released revision of the Enhanced MCU In-Circuit Serial Programming module description.

Section 31. Instruction Set

HIGHLIGHTS

This section of the manual contains the following major topics:

31.1 Introduction

The PIC18CXXX instruction set adds many enhancements to the previous PICmicro instruction sets, while maintaining an easy migration from these PICmicro instruction sets. Most instructions are a single program memory word (16-bits), but to address customer requests, some new instructions have been added that require two program memory locations. The Instruction Set Summary, shown in Table 31-1, lists the instructions recognized by the Microchip assembler (MPASM). Table 31-2 gives the instruction description conventions.

Each instruction is divided into an OPCODE that specifies the instruction type and one or more operands which further specify the operation of the instruction.

The instruction set is highly orthogonal and is grouped into four basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal** operations
- **Control** operations

Most **byte-oriented** instructions have three operands:

1. The File Register (specified by the value of 'f')
2. The destination of the result (specified by the value of 'd')
3. The accessed memory (specified by the value of 'a')

'f' represents a File Register Designator and 'd' represents a Destination Designator. The File Register Designator specifies which File Register is to be used by the instruction. The access indicator 'a' specifies if the BSR selects the bank or if the access bank is used.

The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the WREG Register. If 'd' is one, the result is placed in the File Register specified in the instruction.

All **bit-oriented** instructions have three operands:

1. The File Register (specified by the value of 'f')
2. The bit in the File Register (specified by the value of 'b')
3. The accessed memory (specified by the value of 'a')

'b' represents a bit field designator that selects the number of the bit affected by the operation, while 'f' represents the number of the file in which the bit is located. The access indicator 'a' specifies if the BSR selects the bank or if the access bank is used.

The **literal** instructions may use some of the following operands:

- A literal value to be loaded into a File Register (specified by the value of 'k')
- The desired FSR Register to load the literal value into (specified by the value of 'f')
- No operand required (specified by the value of '—')

The **control** instructions may use some of the following operands:

- A program memory address (specified by the value of 'n')
- The mode of the CALL or RETURN instructions (specified by the value of 's')
- The mode of the `Table Read` and `Table Write` instructions (specified by the value of 'm')
- No operand required (specified by the value of '—')

All instructions are a single word except for three double word instructions. These three instructions were made double word instructions so that all the required information is available in these 32 bits. In the second word, the 4-MSb's, are '1's. If this second word is executed as an instruction (by itself), it will execute as a NOP.

All single word instructions are executed in a single instruction cycle, unless a conditional test is true or the program counter is changed as a result of the instruction. In these cases, the execution takes two instruction cycles with the additional instruction cycle(s) executed as a NOP.

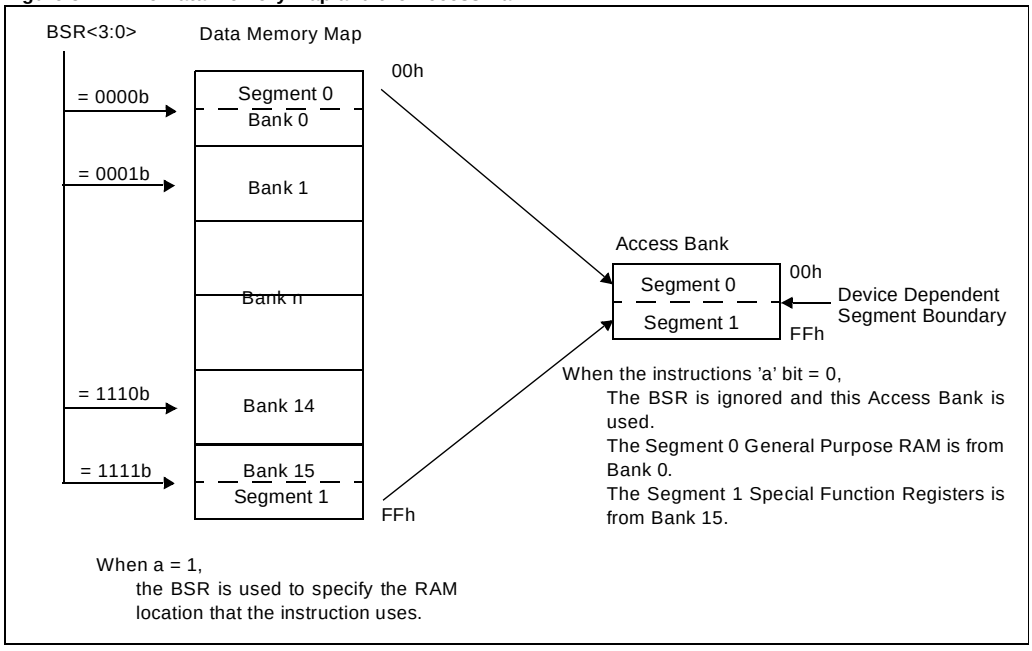
The double word instructions (that do not modify the PC) execute in two instruction cycles.

One instruction cycle consists of four oscillator periods. Thus, for an oscillator frequency of 4 MHz, the normal instruction execution time is 1 µs. If a conditional test is true or the program counter is changed as a result of an instruction, the instruction execution time is 2 µs. Two word branch instructions (if true) would take 3 µs.

31.2 Data Memory Map

The Data Memory Map has 16 banks of 256 bytes. The Instruction Set and architecture allows operations across all banks (such as MOVFF). A Segment of Bank 0 and a segment of Bank 15 comprise the access bank. See Section **31.2.1** for description of the access bank.

Figure 31-1: The Data Memory Map and the Access Bank



31.2.1 Access Bank

The access bank is an architectural enhancement that is very useful for C compiler code optimization. The techniques used by the C compiler may also be useful for programs written in assembly.

This data memory region can be used for

- Intermediate computational values
- Local variables of subroutine
- Faster context saving/switching of variables
- Common variables
- Faster evaluation/control of SFRs (no banking)

The access bank is comprised of 2 segments: Segment 0 and Segment 1. Segment 0 is the RAM that is mapped in Bank 0. Segment 1 is the SFRs that are mapped in Bank 15. Each Segment can be of different sizes. The sum of RAM mapped by Segment 0 and Segment 1 is 256 bytes.

When forced in the access bank (a = '0'), the last address in Segment 0 is followed by the first address in Segment 1. Segment 1 maps the Special Function Registers so that these registers can be accessed without any software overhead. This is useful for testing status flags and modifying control bits.

Example 31-1 shows how registers are affected depending on the value of the access bit. Register MYREG has an 8-bit address. This address specifies the location in the specified bank to perform the operation. The specified bank is either the bank specified by the Bank Select Register (BSR) (when a = 1), or the access bank (when a = 0).

Example 31-1:Operation of Destination and Access Bits

```
;
; The following symbols are defined in the Microchip supplied
; device header file:
;
; For destination bit:
;    F = 1    ; Result is placed in File Register
;    W = 0    ; Result is placed in WREG Register
;
; For access bit:
;    B = 1    ; Register used specified by BSR Bank Register
;    A = 0    ; Register used is in Access Bank
;
; MYREG is a register with an 8-bit address value between 0h and FFh.
; For this example we will assign MYREG to bank 5, though it could
; be in any (or all) banks.
;
   MOVLB    5              ; BSR points to RAM bank 5
                           ;
                           ;          Contents of
                           ;         Addr(MYREG) in
                           ; MYREG    access bank    WREG
;           Starting Value   0x7F        0x5A          x
                           ;
   DECF     MYREG, F, B    ; 0x7E        ---           ---
   DECF     MYREG, F, A    ;  ---        0x59          ---
                           ;
   DECF     MYREG, W, B    ;  ---         ---         0x7D
   DECF     MYREG, W, A    ;  ---         ---         0x58
```

Note:	If the register is specified with the full 12-bit address, the assembler will automatically force the access bit to a '0' (when the address is in the access RAM area) or a '1' (for all other addresses).

The common assembler usage should be that all RAM and SFR addresses are 12-bit. This means that the assembler can determine from the address of the register, whether the access bit needs to be set or cleared. Example 31-2 shows this, as well as forcing the use of the access bank.

Example 31-2:Code

```
;
; The following symbols are defined in the Microchip supplied
; device header file:
;
; For destination bit:
;    F = 1    ; Result is placed in File Register
;    W = 0    ; Result is placed in WREG Register
;
; For access bit:
;    B = 1    ; Register used specified by BSR Bank Register
;    A = 0    ; Register used is in Access Bank
;
;  Register Name   Address
;  Loop_CNTR        0x000
;  MYREG            0x524
;  SFR1             0xA9F
;
    MOVLB    5            ; BSR points to RAM bank 5
                          ;                                    Addr 24
                          ;  a-bit  MYREG Loop_CNTR SFR1  WREG  Bank0
;          Starting Value     ---   0x7F    0x7F    0x7F   x    0xA9
                          ;
    DECF     Loop_CNTR, F ;    0     ---    0x7E     ---   ---   ---
    DECF     MYREG, F     ;    1    0x7E    ---      ---   ---   ---
    DECF     SFR1, F      ;    0     ---    ---     0x7E   ---   ---
                          ;
                          ;
    DECF     Loop_CNTR, W ;    0     ---    ---      ---   0x7D  ---
    DECF     MYREG, W     ;    1     ---    ---      ---   0x7D  ---
    DECF     SFR1, W      ;    0     ---    ---      ---   0x7D  ---

    INCF     MYREG, F, A  ;    0     ---    ---      ---   ---  0xAA
    INCF     MYREG, W, A  ;    0     ---    ---      ---   0x7F  ---
```


Table 31-1: PIC18CXXX Instruction Set Summary

Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF f, d, a	Add WREG and f	1	0010 01da	ffff ffff	C, DC, Z, OV, N	1, 2, 3
ADDWFC f, d, a	Add WREG and Carry bit to f	1	0010 00da	ffff ffff	C, DC, Z, OV, N	1, 2, 3
ANDWF f, d, a	AND WREG with f	1	0001 01da	ffff ffff	Z, N	1,2, 3
CLRF f, a	Clear f	1	0110 101a	ffff ffff	Z	2, 3
COMF f, d, a	Complement f	1	0001 11da	ffff ffff	Z, N	1, 2, 3
CPFSEQ f, a	Compare f with WREG, skip =	1 (2 or 3)	0110 001a	ffff ffff	None	4
CPFSGT f, a	Compare f with WREG, skip >	1 (2 or 3)	0110 010a	ffff ffff	None	4
CPFSLT f, a	Compare f with WREG, skip <	1 (2 or 3)	0110 000a	ffff ffff	None	2
DECF f, d, a	Decrement f	1	0000 01da	ffff ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010 11da	ffff ffff	None	1, 2, 3, 4
DCFSNZ f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100 11da	ffff ffff	None	1, 2
INCF f, d, a	Increment f	1	0010 10da	ffff ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011 11da	ffff ffff	None	1, 2, 4
INFSNZ f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100 10da	ffff ffff	None	1, 2
IORWF f, d, a	Inclusive OR WREG with f	1	0001 00da	ffff ffff	Z, N	1, 2
MOVF f, d, a	Move f	1	0101 00da	ffff ffff	Z, N	1, 2
MOVFF f_s, f_d	Move f_s (source) to 1st word f_d (destination) 2nd word	2	1100 ffff 1111 ffff	ffff ffff ffff ffff	None	2
MOVWF f, a	Move WREG to f	1	0110 111a	ffff ffff	None	2
MULWF f, a	Multiply WREG with f	1	0000 001a	ffff ffff	None	
NEGF f, a	Negate f	1	0110 110a	ffff ffff	C, DC, Z, OV, N	1, 2
RLCF f, d, a	Rotate Left f through Carry	1	0011 01da	ffff ffff	C, Z, N	1, 2
RLNCF f, d, a	Rotate Left f (No Carry)	1	0100 01da	ffff ffff	Z, N	1, 2
RRCF f, d, a	Rotate Right f through Carry	1	0011 00da	ffff ffff	C, Z, N	1, 2
RRNCF f, d, a	Rotate Right f (No Carry)	1	0100 00da	ffff ffff	Z, N	1, 2
SETF f, a	Set f	1	0110 100a	ffff ffff	None	2
SUBFWB f, d, a	Subtract f from WREG with borrow	1	0101 01da	ffff ffff	C, DC, Z, OV, N	1, 2
SUBWF f, d, a	Subtract WREG from f	1	0101 11da	ffff ffff	C, DC, Z, OV, N	1, 2
SUBWFB f, d, a	Subtract WREG from f with borrow	1	0101 10da	ffff ffff	C, DC, Z, OV, N	1, 2
SWAPF f, d, a	Swap nibbles in f	1	0011 10da	ffff ffff	None	1, 2, 4
TSTFSZ f, a	Test f, skip if 0	1 (2 or 3)	0110 011a	ffff ffff	None	2
XORWF f, d, a	Exclusive OR WREG with f	1	0001 10da	ffff ffff	Z, N	
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF f, b, a	Bit Clear f	1	1001 bbba	ffff ffff	None	1, 2
BSF f, b, a	Bit Set f	1	1000 bbba	ffff ffff	None	1, 2
BTFSC f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011 bbba	ffff ffff	None	3, 4
BTFSS f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010 bbba	ffff ffff	None	3, 4
BTG f, d, a	Bit Toggle f	1	0111 bbba	ffff ffff	None	1, 2

Note 1: When a PORT Register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 Register (and, where applicable, d = 1), the prescaler will be cleared if assigned.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

4: Some instructions are 2 word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the table write starts the write cycle to internal program memory, the write continues until terminated.

Table 31-1: PIC18CXXX Instruction Set Summary (Continued)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb LSb	Status Affected	Notes
CONTROL OPERATIONS						
BC	n	Branch if Carry	1 (2)	1110 0010 nnnn nnnn	None	
BN	n	Branch if Negative	1 (2)	1110 0110 nnnn nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110 0011 nnnn nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110 0111 nnnn nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110 0101 nnnn nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110 0001 nnnn nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110 0100 nnnn nnnn	None	
BRA	n	Branch Unconditionally	2	1101 0nnn nnnn nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110 0000 nnnn nnnn	None	
CALL	n, s	Call subroutine 1st word	2	1110 110s kkkk kkkk	None	
		2nd word		1111 kkkk kkkk kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000 0000 0000 0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000 0000 0000 0111	C	
GOTO	n	Go to address 1st word	2	1110 1111 kkkk kkkk	None	
		2nd word		1111 kkkk kkkk kkkk		
NOP	—	No Operation	1	0000 0000 0000 0000	None	
NOP	—	No Operation	1	1111 xxxx xxxx xxxx	None	4
POP	—	Pop top of return stack (TOS)	1	0000 0000 0000 0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000 0000 0000 0101	None	
RCALL	n	Relative Call	2	1101 1nnn nnnn nnnn	None	
RESET		Software device RESET	1	0000 0000 1111 1111	All	
RETFIE	s	Return from interrupt enable	2	0000 0000 0001 000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000 1100 kkkk kkkk	None	
RETURN	s	Return from Subroutine	2	0000 0000 0001 001s	None	
SLEEP	—	Go into standby mode	1	0000 0000 0000 0011	$\overline{TO}$, $\overline{PD}$	

Note 1: When a PORT Register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 Register (and, where applicable, d = 1), the prescaler will be cleared if assigned.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

4: Some instructions are 2 word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the table write starts the write cycle to internal program memory, the write continues until terminated.

Table 31-1: PIC18CXXX Instruction Set Summary (Continued)

Mnemonic, Operands	Description	Cycles	16-Bit Instruction Word MSb LSb	Status Affected	Notes
LITERAL OPERATIONS					
ADDLW k	Add literal and WREG	1	0000 1111 kkkk kkkk	C, DC, Z, OV, N	
ANDLW k	AND literal with WREG	1	0000 1011 kkkk kkkk	Z, N	
IORLW k	Inclusive OR literal with WREG	1	0000 1001 kkkk kkkk	Z, N	
LFSR f, k	Move literal (12-bit) 1st word to FSRx 2nd word	2	1110 1110 00ff kkkk 1111 0000 kkkk kkkk	None	
MOVLB k	Move literal to BSR<3:0>	1	0000 0001 0000 kkkk	None	
MOVLW k	Move literal to WREG	1	0000 1110 kkkk kkkk	None	
MULLW k	Multiply literal with WREG	1	0000 1101 kkkk kkkk	None	
RETLW k	Return with literal in WREG	2	0000 1100 kkkk kkkk	None	
SUBLW k	Subtract WREG from literal	1	0000 1000 kkkk kkkk	C, DC, Z, OV, N	
XORLW k	Exclusive OR literal with WREG	1	0000 1010 kkkk kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS					
TBLRD*	Table Read	2	0000 0000 0000 1000	None	
TBLRD*+	Table Read with post-increment	2	0000 0000 0000 1001	None	
TBLRD*-	Table Read with post-decrement	2	0000 0000 0000 1010	None	
TBLRD+*	Table Read with pre-increment	2	0000 0000 0000 1011	None	
TBLWT*	Table Write	2 (5)	0000 0000 0000 1100	None	
TBLWT*+	Table Write with post-increment	2 (5)	0000 0000 0000 1101	None	
TBLWT*-	Table Write with post-decrement	2 (5)	0000 0000 0000 1110	None	
TBLWT+*	Table Write with pre-increment	2 (5)	0000 0000 0000 1111	None	

Note 1: When a PORT Register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 Register (and, where applicable, d = 1), the prescaler will be cleared if assigned.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

4: Some instructions are 2 word instructions. The second word of these instructions will be executed as a `NOP`, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the table write starts the write cycle to internal program memory, the write continues until terminated.

31.3 Instruction Formats

Figure 31-2 shows the three general formats that the instructions can have. As can be seen from the general format of the instructions, the opcode portion of the instruction word varies from 3-bits to 6-bits of information. This is what allows the Enhanced Instruction Set to have 75 instructions.

Note:	Any unused opcode is Reserved. Use of any reserved opcode may cause unexpected operation.

All instruction examples use the following format to represent a hexadecimal number:

```
0xhh
```

where h signifies a hexadecimal digit.

To represent a binary number:

```
00000100b
```

where b is a binary string identifier.

Figure 31-2: General Format for Instructions

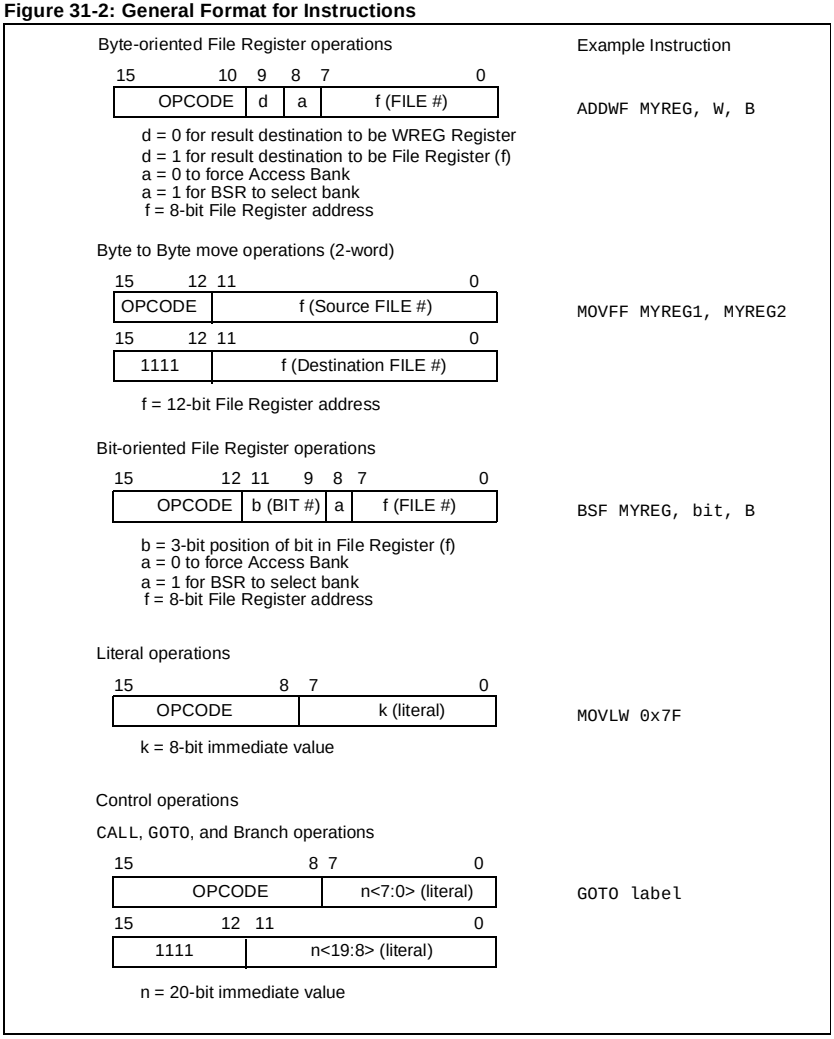


Table 31-2: Instruction Description Conventions

Field	Description
a	RAM access bit a = 0: RAM location in access bank (BSR Register is ignored) a = 1: RAM bank is specified by BSR Register
bbb	Bit address within an 8-bit File Register (0 to 7)
BSR	Bank Select Register. Used to select the current RAM bank.
d	Destination select bit; d = 0: store result in WREG, d = 1: store result in File Register f.
dest	Destination either the WREG Register or the specified register file location
f	8-bit Register file address (0x00 to 0xFF)
fs	12-bit Register file address (0x000 to 0xFFF). This is the source address.
fd	12-bit Register file address (0x000 to 0xFFF). This is the destination address.
k	Literal field, constant data or label (may be either an 8-bit, 12-bit or a 20-bit value)
label	Label name
mm * *+ *- +*	The mode of the TBLPTR Register for the Table Read and Table Write instructions Only used with Table Read and Table Write instructions: No Change to Register (such as TBLPTR with Table reads and writes) Post-Increment Register (such as TBLPTR with Table reads and writes) Post-Decrement Register (such as TBLPTR with Table reads and writes) Pre-Increment Register (such as TBLPTR with Table reads and writes)
n	The relative address (2's complement number) for relative branch instructions, or the direct address for Call/Branch and Return instructions
PRODH	Product of Multiply high byte
PRODL	Product of Multiply low byte
s	Fast Call / Return mode select bit. s = 0: do not update into/from Shadow Registers s = 1: certain registers loaded into/from Shadow Registers
u	Unused or Unchanged
WREG	Working Register (accumulator)
x	Don't care (0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
TBLPTR	21-bit Table Pointer (points to a Program Memory location)
TABLAT	8-bit Table Latch
TOS	Top of Stack
INDF	Any one of the indirect addressing registers, such as INDF0, INDF1, or INDF2
FSR	Any one of the file select register pairs, such as FSR0H:FSR0L, FSR1H:FSR1L, or FSR2H:FSR2L

Table 31-2: Instruction Description Conventions (Continued)

Field	Description
PC	Program Counter
PCL	Program Counter Low Byte
PCH	Program Counter High Byte
PCLATH	Program Counter High Byte Latch
PCLATU	Program Counter Upper Byte Latch
GIE	Global Interrupt Enable bit
WDT	Watchdog Timer
TO	Time-out bit
PD	Power-down bit
C,DC, Z,OV,N	ALU status bits Carry, Digit Carry, Zero, Overflow, Negative
[ ]	Optional
( )	Contents of
→	Assigned to
< >	Register bit field
∈	In the set of
italics	User defined term (font is courier)

Table 31-3: Indirect Addressing Symbols

Field	Description
*FSRn	Selects INDFn Register
*FSRn++	Selects POSTINCn Register
*FSRn--	Selects POSTDECn Register
*(++FSRn)	Selects PREINCn Register
*(FSRn+W)	Selects PLUSWn Register

31.4 Special Function Registers as Source/Destination

The Section 31. Instruction Set's Orthogonal Instruction Set allows read and write of all File Registers, including Special Function Registers. The user should be aware of some special situations which are explained in the following subsections.

31.4.1 STATUS Register as Destination

If an instruction writes to the STATUS Register, the Z, C, DC, OV, and N bits may be set or cleared as a result of the instruction and overwrite the original data bits. For example, executing CLRF STATUS will clear Register STATUS, and then set the Z bit leaving 0000 0100b in the register.

31.4.2 Bit Manipulation

All bit manipulation instructions will first read the entire register, operate on the selected bit and then write the result back to (read-modify-write (R-M-W)) the specified register. The user should keep this in mind when operating on some Special Function Registers, such as the Port Pin Register.

Note:	Status bits that are manipulated by the device (including the interrupt flag bits) are set or cleared in the Q1 cycle, so there is no issue with executing R-M-W instructions on registers that contain these bits.

31.4.3 PCL as Source or Destination

Read, write or read-modify-write (R-M-W) on PCL may have the following results:

Read PCL:	PCL → destination PCH → PCLATH PCU → PCLATU	; Reading PCL causes the following
Write PCL:	8-bit destination value → PCL PCLATH → PCH PCLATU → PCU	; Writing PCL causes the following
Read-Modify-Write:	PCL→ ALU operand PCH → PCLATH PCU → PCLATU 8-bit result → PCL PCLATH → PCH PCLATU → PCU	; R-M-W of PCL causes the following ; PCL data is modified ; result is written back to PCL

Where PCH = program counter high byte (not an addressable register),
PCLATH = Program counter high holding latch,
PCU = program counter upper byte (not an addressable register),
PCLATU = Program counter upper holding latch,
destination = Register file 'f'.

31.5 Fast Register Stack

At times it is desirable to be able to quickly access and return from a function. This function may be called as a subroutine, or an interrupt routine of the device. To reduce the overhead for accessing/returning from these functions, the architecture has the ability to save three key registers in a one deep Register Stack. These registers are:

- WREG Register
- BSR (Bank Select Register) Register
- STATUS Register

The two events that cause these registers to be loaded onto the Fast Register Stack are:

- A fast call (`CALL K, fast`)(where the `fast` bit is set ('1'))
- Any interrupt occurs

These Fast Stack Registers are not accessible for reading or writing. When doing the return from these subroutine, the values can be restored into their registers executing the fast return:

- `RETFIE fast` (where the `fast` bit is set ('1'))
- `RETURN fast` (where the `fast` bit is set ('1'))

When s (`fast`) = '0', the Fast Register Stack is not used, when s (`fast`) = '1', the Fast Register Stack is used.

31.6 Q Cycle Activity

Each instruction cycle (Tcy) is comprised of four Q clocks (also called Q cycles). These are referred to as Q1, Q2, Q3, or Q4. The Q cycles provide the timing/designation for the Decode, Read, Process Data, Write etc., of each instruction cycle. The Figure 31-3 shows the relationship of the Q cycles to the instruction cycle.

The four Q cycles that make up an instruction cycle (Tcy) can be generalized as:

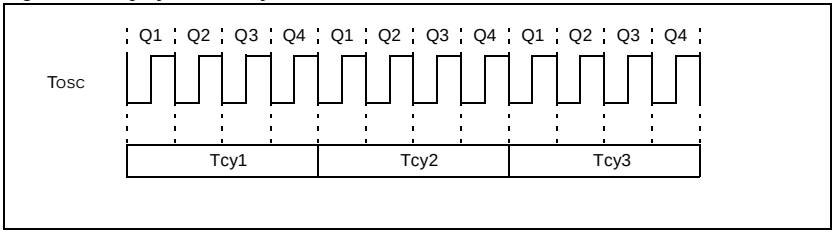
Q1: Instruction Decode Cycle or forced No Operation
Q2: Instruction Read Cycle or No Operation
Q3: Process the Data
Q4: Instruction Write Cycle or No Operation

Some actions occur on the edge between the end of one Q cycle and the start of the next Q cycle. An example would be a Q2-Q3 action. This action occurs on the clock edge between the end of Q2 cycle and the beginning of the Q3 cycle.

The clock source for the Q cycle is normally the device oscillator clock (TOSC). But the clock source is software selectable. So the Q cycle may be independent of the device oscillator cycle (TOSC).

In the full description of each instruction, the detailed Q cycle operation for the instruction will be shown.

Figure 31-3: Q Cycle Activity



31.7 Instruction Descriptions

ADDLW Add Literal to WREG

Syntax: [*label*] ADDLW k

Operands: $0 \le k \le 255$

Operation: (WREG) + k → WREG

Status Affected: C, DC, Z, OV, N

Encoding:

0000	1111	kkkk	kkkk

Description: The eight bit literal 'k' is added to the contents of the WREG and the result is placed in the WREG.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to WREG Register

Example1

```
ADDLW  0x19          ; Add 19h to value in WREG
```

Before Instruction
WREG = 0x18
C, DC, Z, OV, N = x
After Instruction
WREG = 0x31
C = 0
DC = 1
Z = 0
OV = 0
N = 0

Example 2

```
ADDLW  MYREG         ; Add the value of the
                     ; address for MYREG Register
                     ; to WREG
```

Before Instruction
WREG = 0x60
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
C, DC, Z, OV, N = x
After Instruction
WREG = 0x97
C = 0
DC = 0
Z = 0
OV = 1
N = 1

Example 3

```
ADDLW   HIGH (LU_TABLE)  ; Add high byte of address
                         ; LU_TABLE to WREG
```

Before Instruction
WREG = 0x10
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
C, DC, Z, OV, N = x
After Instruction
WREG = 0xA3
C = 0
DC = 0
Z = 0
OV = 0
N = 1

Example 4

```
ADDLW  PCL           ; Add value of the address
                     ; of Program Counter Low
                     ; byte (PCL) to WREG
```

Before Instruction
WREG = 0x02
Address of PCL † = 0xFF8 (only low 8-bits are used)
† PCL is the symbol for the Program Counter low byte location
C, DC, Z, OV, N = x
After Instruction
WREG = 0xFA
C = 0
DC = 0
Z = 0
OV = 0
N = 0

Example 5

```
ADDLW  Offset        ; Add the value of symbol
                     ; Offset to WREG
```

Before Instruction
WREG = 0x10
Offset = 0x02
C, DC, Z, OV, N = x
After Instruction
WREG = 0x12
Offset = 0x02
C = 0
DC = 0
Z = 0
OV = 0
N = 0

ADDWF Add WREG and f

Syntax: [*label*] ADDWF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (WREG) + (f) → destination

Status Affected: C, DC, Z, OV, N

Encoding:

0010	01da	ffff	ffff

Description: Add the contents of the WREG Register to the contents of Register 'f'.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

```
Example 1               ADDWF   FSR0L, 1, 1 ; Add value in WREG to
                                            ;    value in the
                                            ;    FSR0H:FSR0L Register
```

Case 1:
Before Instruction
WREG = 0x17
FSR0H:FSR0L = 0x2C2
C, DC, Z, OV, N = x
After Instruction
WREG = 0x17
FSR0H:FSR0L = 0x2D9
C = 0
DC = 0
Z = 0
OV = 0
N = 1

Case 2:
Before Instruction
WREG = 0x17
FSR0H:FSR0L = 0x2FF
C, DC, Z, OV, N = x
After Instruction
WREG = 0x17
FSR0H:FSR0L = 0x316
C = 0
DC = 0
Z = 0
OV = 0
N = 0

Example 2

```
ADDWF   INDF0, 1, 1 ; Add value in WREG to
                    ; value in the register
                    ; pointed to (addressed)
                    ; by the FSR0H:FSR0L
                    ; Register
```

Before Instruction
WREG = 0x17
FSR0H:FSR0L = 0x6C2
Contents of Address (FSR0) = 0x20
C, DC, Z, OV, N = x
After Instruction
WREG = 0x17
FSR0H:FSR0L = 0x6C2
Contents of Address (FSR0) = 0x37
C = 0
DC = 0
Z = 0
OV = 0
N = 0

Example 3

```
ADDWF   INDF0, 1, 0 ; Add value in WREG to
                    ; value in the register
                    ; pointed to (addressed)
                    ; by the FSR0H:FSR0L
                    ; Register
```

Before Instruction
WREG = 0x17
FSR0H:FSR0L = 0x0C2
Contents of Address (FSR0) = 0x20
C, DC, Z, OV, N = x
After Instruction
WREG = 0x17
FSR0H:FSR0L = 0x0C2
Contents of Address (FSR0) = 0x37
C = 0
DC = 0
Z = 0
OV = 0
N = 0

Example 4

```
ADDWF    PCL, 1, 1  ; Add the value in WREG to
                    ; the current value in the
                    ; low byte of the program
                    ; counter (PCL)
```

Case 1:
Before Instruction
WREG = 0x10
PCL = 0x37
C, DC, Z, OV, N = x
After Instruction
WREG = 0x10
PCL = 0x47
C = 0
DC = 0
Z = 0
OV = 0
N = 0

Case 2:
Before Instruction
WREG = 0x10
PCL = 0xF7
PCH = 0x08
C, DC, Z, OV, N = x
After Instruction
WREG = 0x10
PCL = 0x07
PCH = 0x08
C = 1
DC = 0
Z = 0
OV = 0
N = 0

```
Example 5               ADDWF    MYREG, 1    ; Add the value in WREG to
                                             ; the current value in
                                             ; MYREG
                                             ; (assembler determines
                                             ;  that MYREG requires
                                             ;  access bit to be set)
```

Case 1: Before Instruction

```
        BSR       =     0x01
        WREG      =     0x10
        MYREG     =     0x37     ; In Bank 1
        C, DC, Z, OV, N  =   x
```

After Instruction

```
        BSR       =     0x01
        WREG      =     0x10
        MYREG     =     0x47     ; In Bank 1
        C         =     0
        DC        =     0
        Z         =     0
        OV        =     0
        N         =     0
```

Case 2: Before Instruction

```
        BSR       =     0x01
        WREG      =     0x10
        MYREG     =     0xF7     ; In Bank 1
        C, DC, Z, OV, N  =   x
```

After Instruction

```
        BSR       =     0x01
        WREG      =     0x10
        MYREG     =     0x07     ; In Bank 1
        C         =     1
        DC        =     0
        Z         =     0
        OV        =     0
        N         =     0
```

ADDWFC Add WREG and Carry bit to f

Syntax: [*label*] ADDWF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (WREG) + (f) + (C) → destination

Status Affected: C, DC, Z, OV, N

Encoding:

0010	00da	ffff	ffff

Description: Add the contents of the WREG Register and the Carry bit to the contents of Register 'f'.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

```
Example 1               ADDWFC  FSR0L, 0, 1 ; Add WREG, C bit, and FSR0L
                                             ;   value (Destination WREG)
```

Case 1: Before Instruction
WREG = 0x17
FSR0H:FSR0L = 0x9C2
C = 0
DC, Z, OV, N = x
After Instruction
WREG = 0xD9
FSR0H:FSR0L = 0x9C2
C = 0
DC = 0
Z = 0
OV = 0
N = 1

Case 2: Before Instruction
WREG = 0x17
FSR0H:FSR0L = 0x7C2
C = 1
DC, Z, OV, N = x
After Instruction
WREG = 0xDA
FSR0H:FSR0L = 0x7C2
C = 0
DC = 0
Z = 0
OV = 0
N = 1

Example 2

```
ADDWFC  INDF0, 1, 1 ; Add WREG and the Carry
                    ; bit to the value pointed
                    ; to by the FSR0H:FSR0L
                    ; (Destination: File
                    ; Register)
```

Before Instruction
- WREG = 0x17
- FSR0H:FSR0L = 0x0C2
- Contents of Address (FSR0H:FSR0L) = 0x20
- C = 0
- DC, Z, OV, N = x

After Instruction
- WREG = 0x17
- FSR0H:FSR0L = 0x0C2
- Contents of Address (FSR0H:FSR0L) = 0x37
- C = 0
- DC = 0
- Z = 0
- OV = 0
- N = 0

Example 3

```
ADDWFC  PCL, 1, 1   ; Add WREG and the Carry
                    ; bit to the PCL Register
```

Case 1:

Before Instruction
- WREG = 0x10
- PCL = 0x38
- C = 0
- DC, Z, OV, N = x

After Instruction
- WREG = 0x10
- PCL = 0x48
- C = 0
- DC = 0
- Z = 0
- OV = 0
- N = 0

Case 2:

Before Instruction
- WREG = 0x10
- PCL = 0xF8
- PCH = 0x08
- C = 0
- DC, Z, OV, N = x

After Instruction
- WREG = 0x10
- PCL = 0x08
- PCH = 0x08
- C = 1
- DC = 0
- Z = 0
- OV = 0
- N = 0

ANDLW AND Literal with WREG

Syntax: [*label*] ANDLW k

Operands: $0 \le k \le 255$

Operation: (WREG).AND. (k) → W

Status Affected: Z, N

Encoding:

0000	1011	kkkk	kkkk

Description: The contents of WREG Register are AND'd with the eight bit literal 'k'. The result is placed in the WREG Register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to WREG Register

Example 1

```
ANDLW   0x5F          ; And constant to WREG
```

Before Instruction

```
WREG  =   0xA3        ; 0101 1111  (0x5F)
Z, N  =   x           ; 1010 0011  (0xA3)
```

After Instruction

```
                      ;----------  ------
WREG  =   0x03        ; 0000 0011  (0x03)
Z     =   0
N     =   0
```

Example 2

```
ANDLW   MYREG         ; And address of MYREG
                      ; to WREG
```

Before Instruction

```
WREG    =   0xA3      ; 0011 0111  (0x37)
Address of MYREG † = 0x37 ; 1010 0011  (0xA3)
Z, N    =   x         ;----------  ------
                      ; 0010 0011  (0x23)
```

† MYREG is a symbol for a data memory location

After Instruction

```
WREG    =   0x23
Z       =   0
N       =   0
```

Example 3

```
ANDLW     HIGH          ; And the high byte of
          (LU_TABLE)    ; address LU_TABLE
                        ; with WREG
```

Before Instruction

```
WREG  =  0xA3           ; 1010 0011  (0xA3)
Address of LU_TABLE †
      =  0x9375         ; 1001 0011  (0x93)
Z, N  =  x              ;----------  ------
```

† LU_TABLE is a label for an address in program memory

After Instruction

```
WREG  =  0x83           ; 1000 0011  (0x83)
Z     =  0
N     =  1
```

Example 4

```
ANDLW     LOW (LU_TABLE); And the low byte of
                        ; address LU_TABLE
                        ; with WREG
```

Before Instruction

```
WREG  =  0xA3           ; 1010 0011  (0xA3)
Address of LU_TABLE †
      =  0x9375         ; 0111 0101  (0x75)
Z, N  =  x              ;----------  ------
```

† LU_TABLE is a label for an address in program memory

After Instruction

```
WREG  =  0x21           ; 0010 0001  (0x21)
Z     =  0
N     =  0
```

ANDWF AND WREG with f

Syntax: [*label*] ANDWF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (WREG).AND. (f) → destination

Status Affected: Z, N

Encoding:

0001	01da	ffff	ffff

Description: The contents of the WREG Register are AND'd with the contents of Register 'f'.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
        ANDWF  REG1, 1, 1   ; And WREG with REG1
Before Instruction
        WREG    =  0x17     ; 0001 0111  (0x17)
        REG1    =  0xC2     ; 1100 0010  (0xC2)
        Z, N    =  x        ;----------  ------
After Instruction
        WREG    =  0x17
        REG1    =  0x02     ; 0000 0010  (0x02)
        Z       =  0
        N       =  0
```

Example 2

```
        ANDWF REG1, 0, 1    ; And WREG with REG1
                            ; (destination WREG)
Before Instruction
        WREG    =  0x17     ; 0001 0111  (0x17)
        REG1    =  0xC2     ; 1100 0010  (0xC2)
        Z, N    =  x        ;----------  ------
After Instruction
        WREG    =  0x02     ; 0000 0010  (0x02)
        REG1    =  0xC2
        Z       =  0
        N       =  0
```

```
Example 3               ANDWF INDF0, 1, 1  ; And WREG with value pointed
                                           ; by FSR0H:FSR0L (FSR0)
     Case 1:    Before Instruction
                     WREG   =   0x17       ; 0001 0111  (0x17)
                     FSR0H:FSR0L
                            =   0xFC2
                     Contents of Address   ; 0101 1010  (0x5A)
                          (FSR0) = 0x5A    ;----------  ------
                     Z, N   =   x
                After Instruction
                     WREG   =   0x17
                     FSR0H:FSR0L
                            =   0xFC2      ; 0001 0010  (0x12)
                     Contents of Address
                          (FSR0) = 0x12
                     Z      =   0
                     N      =   0
     Case 2:    Before Instruction
                     WREG   =   0x00       ; 0000 0000  (0x00)
                     FSR0H:FSR0L
                            =   0x4C2      ; 0101 1010  (0x5A)
                     Contents of Address   ;----------  ------
                          (FSR0) = 0x5A
                     Z, N   =   x
                After Instruction
                     WREG   =   0x00
                     FSR0H:FSR0L
                            =   0x4C2      ; 0000 0000  (0x00)
                     Contents of Address
                          (FSR0) = 0x00
                     Z      =   1
                     N      =   0
```

BC Branch if Carry

Syntax: [*label*] BC n

Operands: -128 ≤ f ≤ 127

Operation: If carry bit is '1'
(PC + 2) + 2n → PC

Status Affected: None

Encoding:

1110	0010	nnnn	nnnn

Description: If the Carry bit is '1', then the program will branch.
The 2's complement number '2n' (the offset) is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be (PC+2)+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1 (2)

Q Cycle Activity:

If Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	Write to PC
No operation	No operation	No operation	No operation

If No Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	No operation

Example 1

```
HERE    BC      C_CODE          ;
NOT_C   •                       ; If C bit is not set
        •                       ; execute this code.
        GOTO    MORE_CODE       ;
C_CODE  •                       ; else if C bit is set
        •                       ; this code will execute
```

Case 1: Before Instruction
PC = address HERE
C = 0
After Instruction
PC = address NOT_C

Case 2: Before Instruction
PC = address HERE
C = 1
After Instruction
PC = address C_CODE

Example 2

```
HERE    BC     $ + OFFSET    ; If carry bit is set,
NO_C    GOTO   PROCESS_CODE  ; branch to HERE+OFFSET
PLUS0   •
PLUS1   •
PLUS2   •
PLUS3   •
PLUS4   •
PLUS5   •
PLUS6   •
```

Case 1:
Before Instruction
PC = address HERE
C = 0
After Instruction
PC = address NO_C

Case 2:
Before Instruction
PC = address HERE
C = 1
After Instruction
PC = address HERE + OFFSET

Example 3

```
MIN6    •
MIN5    •
MIN4    •
MIN3    •
MIN2    •
MIN1    •
MIN0    •                    ; If carry bit is set,
HERE    BC     $ - OFFSET    ; branch to HERE-OFFSET
NO_C    GOTO   PROCESS_CODE
```

Case 1:
Before Instruction
PC = address HERE
C = 0
After Instruction
PC = address NO_C

Case 2:
Before Instruction
PC = address HERE
C = 1
After Instruction
PC = address HERE - OFFSET

Note: Assembler will convert the specified address label into the offset to be used.

BCF Bit Clear f

Syntax: [*label*] BCF f, b, a

Operands: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

Operation: $0 \rightarrow$ f<b>

Status Affected: None

Encoding:

1001	bbba	ffff	ffff

Description: Bit 'b' in Register 'f' of the specified bank is cleared.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write Register 'f'

Example 1

```
BCF   MYREG, 7, 1  ; Clear bit 7 in Register
                   ; MYREG
```

Before Instruction

```
MYREG  =  0xC7     ; 1100 0111
```

After Instruction

```
MYREG  =  0x47     ; 0100 0111
```

Example 2

```
BCF  INDF0, 3, 0  ; Clear bit 7 in the register
                  ; pointed to by the FSR0
                  ; (FSR0H:FSR0L) Register
```

Before Instruction
FSR0 = 0x3C2
Contents of Address
(FSR0) = 0x2F ; 0010 **1**111
After Instruction
FSR0 = 0x3C2
Contents of Address
(FSR0) = 0x27 ; 0010 **0**111

BN Branch if Negative

Syntax: [*label*] BN n

Operands: -128 ≤ f ≤ 127

Operation: If negative bit is '1'
(PC + 2) + 2n → PC

Status Affected: None

Encoding:

1110	0110	nnnn	nnnn

Description: If the Negative bit is '1', then the program will branch.
The 2's complement number '2n' (the offset) is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be (PC+2)+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1 (2)

Q Cycle Activity:

If Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	Write to PC
No operation	No operation	No operation	No operation

If No Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	No operation

Example 1

```
HERE    BN      N_CODE        ; If N bit is not set
NOT_N   •                     ; execute this code.
        •                     ;
        GOTO    MORE_CODE     ;
N_CODE  •                     ; else if N bit is set
        •                     ; this code will execute
```

Case 1: Before Instruction
PC = address HERE
N = 0
After Instruction
PC = address NOT_N

Case 2: Before Instruction
PC = address HERE
N = 1
After Instruction
PC = address N_CODE

Example 2

```
HERE    BN     $ + OFFSET    ; If negative bit is set,
NOT_N   GOTO   PROCESS_CODE  ; branch to HERE + OFFSET
PLUS0   •
PLUS1   •
PLUS2   •
PLUS3   •
PLUS4   •
PLUS5   •
PLUS6   •
```

Case 1: Before Instruction
PC = address HERE
N = 0
After Instruction
PC = address NOT_N

Case 2: Before Instruction
PC = address HERE
N = 1
After Instruction
PC = address HERE + OFFSET

Example 3

```
MIN6    •
MIN5    •
MIN4    •
MIN3    •
MIN2    •
MIN1    •
MIN0    •                        ; If negative bit is set,
HERE    BN      $ - OFFSET       ; branch to HERE - OFFSET
NO_N    GOTO    PROCESS_CODE
```

Case 1: Before Instruction
PC = address HERE
N = 0
After Instruction
PC = address NO_N

Case 2: Before Instruction
PC = address HERE
N = 1
After Instruction
PC = address HERE - OFFSET

Note: Assembler will convert the specified address label into the offset to be used.

BNC Branch if Not Carry

Syntax: [*label*] BNC n

Operands: -128 ≤ f ≤ 127

Operation: If carry bit is '0'
(PC + 2) + 2n → PC

Status Affected: None

Encoding:

1110	0011	nnnn	nnnn

Description: If the Carry bit is '0', then the program will branch.
The 2's complement number '2n' (the offset) is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be (PC+2)+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1 (2)

Q Cycle Activity:

If Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	Write to PC
No operation	No operation	No operation	No operation

If No Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	No operation

Example 1

```
HERE     BNC     NC_CODE      ; If C bit is set
CARRY    •                    ; execute this code.
         •                    ;
         GOTO    MORE_CODE    ;
NC_CODE  •                    ; else if C bit is clear
         •                    ; this code will execute
```

Case 1: Before Instruction
PC = address HERE
C = 0
After Instruction
PC = address NC_CODE

Case 2: Before Instruction
PC = address HERE
C = 1
After Instruction
PC = address CARRY

Example 2

```
HERE    BNC    $ + OFFSET    ; If carry bit is clear,
CARRY   GOTO   PROCESS_CODE  ; branch to HERE + OFFSET
PLUS0   •
PLUS1   •
PLUS2   •
PLUS3   •
PLUS4   •
PLUS5   •
PLUS6   •
```

Case 1:
Before Instruction
PC = address HERE
C = 0
After Instruction
PC = address HERE + OFFSET

Case 2:
Before Instruction
PC = address HERE
C = 1
After Instruction
PC = address CARRY

Example 3

```
MIN6    •
MIN5    •
MIN4    •
MIN3    •
MIN2    •
MIN1    •
MIN0    •                    ; If carry bit is clear,
HERE    BNC    $ - OFFSET    ; branch to HERE - OFFSET
CARRY   GOTO   PROCESS_CODE
```

Case 1:
Before Instruction
PC = address HERE
C = 0
After Instruction
PC = address HERE - OFFSET

Case 2:
Before Instruction
PC = address HERE
C = 1
After Instruction
PC = address CARRY

Note: Assembler will convert the specified address label into the offset to be used.

BNN Branch if Not Negative

Syntax: [*label*] BNN n

Operands: -128 ≤ f ≤ 127

Operation: If negative bit is '0'
(PC + 2) + 2n → PC

Status Affected: None

Encoding:

1110	0111	nnnn	nnnn

Description: If the Negative bit is '0', then the program will branch.
The 2's complement number '2n' (the offset) is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be (PC+2)+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1 (2)

Q Cycle Activity:

If Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	Write to PC
No operation	No operation	No operation	No operation

If No Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	No operation

Example 1

```
HERE     BNN     POS_CODE     ; If N bit is set
NEG      •                    ; execute this code.
         •                    ;
         GOTO    MORE_CODE    ;
POS_CODE •                    ; else if N bit is clear
         •                    ; this code will execute
```

Case 1: Before Instruction
PC = address HERE
N = 0
After Instruction
PC = address POS_CODE

Case 2: Before Instruction
PC = address HERE
N = 1
After Instruction
PC = address NEG

Example 2

```
HERE    BNN    $ + OFFSET    ; If negative bit is clear,
NEG     GOTO   PROCESS_CODE  ; branch to HERE + OFFSET
PLUS0   •
PLUS1   •
PLUS2   •
PLUS3   •
PLUS4   •
PLUS5   •
PLUS6   •
```

Case 1:
Before Instruction
PC = address HERE
N = 0
After Instruction
PC = address HERE + OFFSET

Case 2:
Before Instruction
PC = address HERE
N = 1
After Instruction
PC = address NEG

Example 3

```
MIN6    •
MIN5    •
MIN4    •
MIN3    •
MIN2    •
MIN1    •
MIN0    •                    ; If negative bit is clear,
HERE    BNN    $ - OFFSET    ; branch to HERE - OFFSET
NEG     GOTO   PROCESS_CODE
```

Case 1:
Before Instruction
PC = address HERE
N = 0
After Instruction
PC = address HERE - OFFSET

Case 2:
Before Instruction
PC = address HERE
N = 1
After Instruction
PC = address NEG

Note: Assembler will convert the specified address label into the offset to be used.

BNOV

Branch if Not Overflow

Syntax: [*label*] BNOV n

Operands: -128 ≤ f ≤ 127

Operation: If overflow bit is '0'
(PC + 2) + 2n → PC

Status Affected: None

Encoding:

1110	0101	nnnn	nnnn

Description: If the Overflow bit is '0', then the program will branch.
The 2's complement number '2n' (the offset) is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be (PC+2)+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1 (2)

Q Cycle Activity:

If Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	Write to PC
No operation	No operation	No operation	No operation

If No Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	No operation

Example 1

```
HERE     BNOV    NOV_CODE     ; If overflow bit is set
OVFL     •                    ; execute this code.
         •                    ;
         GOTO    MORE_CODE    ;
NOV_CODE•                     ; else if overflow bit is
         •                    ; clear this code will
                              ; execute
```

Case 1: Before Instruction
PC = address HERE
OV = 0
After Instruction
PC = address NOV_CODE

Case 2: Before Instruction
PC = address HERE
OV = 1
After Instruction
PC = address OVFL

Example 2

```
HERE    BNOV  $ + OFFSET   ; If overflow bit is clear,
OVFL    GOTO  PROCESS_CODE ; branch to HERE + OFFSET
PLUS0   •
PLUS1   •
PLUS2   •
PLUS3   •
PLUS4   •
PLUS5   •
PLUS6   •
```

Case 1: Before Instruction
PC = address HERE
OV = 0
After Instruction
PC = address HERE + OFFSET

Case 2: Before Instruction
PC = address HERE
OV = 1
After Instruction
PC = address OVFL

Example 3

```
MIN6    •
MIN5    •
MIN4    •
MIN3    •
MIN2    •
MIN1    •
MIN0    •                  ; If overflow bit is clear,
HERE    BNOV  $ - OFFSET   ; branch to HERE - OFFSET
OVFL    GOTO  PROCESS_CODE
```

Case 1: Before Instruction
PC = address HERE
OV = 0
After Instruction
PC = address HERE - OFFSET

Case 2: Before Instruction
PC = address HERE
OV = 1
After Instruction
PC = address OVFL

Note: Assembler will convert the specified address label into the offset to be used.

BNZ Branch if Not Zero

Syntax: [*label*] BNZ n

Operands: -128 ≤ f ≤ 127

Operation: If zero bit is '0'
(PC + 2) + 2n → PC

Status Affected: None

Encoding:

1110	0001	nnnn	nnnn

Description: If the Zero bit is '0', then the program will branch.
The 2's complement number '2n' (the offset) is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be (PC+2)+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1 (2)

Q Cycle Activity:

If Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	Write to PC
No operation	No operation	No operation	No operation

If No Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	No operation

Example 1

```
HERE    BNZ     Z_CODE        ; If Z bit is set
ZERO    •                     ; execute this code.
        •                     ;
        GOTO    MORE_CODE     ;
Z_CODE  •                     ; else if Z bit is clear
        •                     ; this code will execute
```

Case 1: Before Instruction
PC = address HERE
Z = 0
After Instruction
PC = address Z_CODE

Case 2: Before Instruction
PC = address HERE
Z = 1
After Instruction
PC = address ZERO

Example 2

```
HERE    BNZ   $ + OFFSET   ; If zero bit is clear,
ZERO    GOTO  PROCESS_CODE ; branch to HERE + OFFSET
PLUS0   •
PLUS1   •
PLUS2   •
PLUS3   •
PLUS4   •
PLUS5   •
PLUS6   •
```

Case 1:
Before Instruction
PC = address HERE
Z = 0
After Instruction
PC = address HERE + OFFSET

Case 2:
Before Instruction
PC = address HERE
Z = 1
After Instruction
PC = address ZERO

Example 3

```
MIN6    •
MIN5    •
MIN4    •
MIN3    •
MIN2    •
MIN1    •
MIN0    •                  ; If zero bit is clear,
HERE    BNZ   $ - OFFSET   ; branch to HERE - OFFSET
ZERO    GOTO  PROCESS_CODE
```

Case 1:
Before Instruction
PC = address HERE
Z = 0
After Instruction
PC = address HERE - OFFSET

Case 2:
Before Instruction
PC = address HERE
Z = 1
After Instruction
PC = address ZERO

Note: Assembler will convert the specified address label into the offset to be used.

BOV

Branch if Overflow

Syntax: [*label*] BOV n

Operands: -128 ≤ f ≤ 127

Operation: If overflow bit is '1'
(PC + 2) + 2n → PC

Status Affected: None

Encoding:

1110	0100	nnnn	nnnn

Description: If the Overflow bit is '1', then the program will branch.
The 2's complement number '2n' (the offset) is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be (PC+2)+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1 (2)

Q Cycle Activity:

If Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	Write to PC
No operation	No operation	No operation	No operation

If No Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	No operation

Example 1

```
HERE     BOV     OV_CODE      ; If OV bit is clear
OVFL     •                    ; execute this code.
         •                    ;
         GOTO    MORE_CODE    ;
OV_CODE  •                    ; else if OV bit is set
         •                    ; this code will execute
```

Case 1: Before Instruction
PC = address HERE
OV = 0
After Instruction
PC = address OVFL

Case 2: Before Instruction
PC = address HERE
OV = 1
After Instruction
PC = address OV_CODE

Example 2

```
HERE    BOV   $ + OFFSET   ; If overflow bit is set,
OVFL    GOTO  PROCESS_CODE ; branch to HERE + OFFSET
PLUS0   •
PLUS1   •
PLUS2   •
PLUS3   •
PLUS4   •
PLUS5   •
PLUS6   •
```

Case 1: Before Instruction
PC = address HERE
OV = 0
After Instruction
PC = address OVFL

Case 2: Before Instruction
PC = address HERE
OV = 1
After Instruction
PC = address HERE + OFFSET

Example 3

```
MIN6    •
MIN5    •
MIN4    •
MIN3    •
MIN2    •
MIN1    •
MIN0    •                  ; If OV bit is set,
HERE    BOV   $ - OFFSET   ; branch to HERE - OFFSET
OVFL    GOTO  PROCESS_CODE
```

Case 1: Before Instruction
PC = address HERE
OV = 0
After Instruction
PC = address OVFL

Case 2: Before Instruction
PC = address HERE
OV = 1
After Instruction
PC = address HERE - OFFSET

Note: Assembler will convert the specified address label into the offset to be used.

BRA Branch Unconditional

Syntax: [*label*] BRA n

Operands: -1024 ≤ f ≤ 1023

Operation: (PC + 2) + 2n → PC

Status Affected: None

Encoding:

1101	0nnn	nnnn	nnnn

Description: The 2's complement number '2n' (the offset) is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be (PC+2)+2n. This instruction is a two-cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

If Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	Write to PC
No operation	No operation	No operation	No operation

If No Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	No operation

Example 1

```
HERE    BRA     THERE           ; Branch to a program memory
        •                       ; location (THERE)
        •                       ; this location must be
        •                       ; < 1023 locations forward
THERE   •
```

Before Instruction
PC = address HERE
After Instruction
PC = address THERE

Example 2

```
THERE   •                       ; Branch to a program memory
        •                       ; location (THERE)
        •                       ; this location must be
        •                       ; < 1024 locations backward
HERE    BRA     THERE
```

Before Instruction
PC = address HERE
After Instruction
PC = address THERE

Example 3

```
HERE    BRA     $               ; Branch to program memory
                                ; location (HERE).
                                ; Infinite Loop
```

Before Instruction
PC = address HERE
After Instruction
PC = address HERE

Note: Assembler will convert the specified address label into the offset to be used.

BSF Bit Set f

Syntax: [*label*] BSF f, b, a

Operands: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

Operation: $1 \rightarrow f<b>$

Status Affected: None

Encoding:

1000	bbba	ffff	ffff

Description: Bit 'b' in Register 'f' is set.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write Register 'f'

Example 1

```
BSF   FLAG_REG, 7  ; Set bit 7 in Register
                   ; FLAG_REG
                   ; (assembler determines
                   ; that FLAG_REG requires
                   ; access bit to be set)
```

Before Instruction
FLAG_REG = 0x0A ; **0**000 1010
After Instruction
FLAG_REG = 0x8A ; **1**000 1010

Example 2

```
BSF  INDF0, 3, 0  ; Set bit 3 in the register
                  ; pointed to by the FSR0
                  ; (FSR0H:FSR0L) Register
```

Before Instruction
WREG = 0x17
FSR0 = 0x0C2
Contents of Address ; 0010 **0**000
(FSR0)= 0x20
After Instruction
WREG = 0x17
FSR0 = 0x0C2
Contents of Address ; 0010 **1**000
(FSR0) = 0x28

BTFSC Bit Test File, Skip if Clear

Syntax: [*label*] BTFSC f, b, a

Operands: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

Operation: Skip if (f<b>) = 0

Status Affected: None

Encoding:

1011	bbba	ffff	ffff

Description: If bit 'b' in Register 'f' is '0' then the next instruction is skipped.
If bit 'b' is '0' then the next instruction (fetched during the current instruction execution) is discarded, and a NOP is executed instead, making this a 2-cycle instruction.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1 (2 or 3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	No operation

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by a two word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example 1

```
HERE    BTFSC  FLAG, 4, 1    ; Test bit 4 of Register
FALSE   GOTO   PROCESS_CODE  ; FLAG, and skip if
TRUE    •                    ; clear
        •
        •
```

Case 1: Before Instruction
PC = address HERE
FLAG = xxx0 xxxx
After Instruction
Since FLAG<4> = 0
PC = address TRUE

Case 2: Before Instruction
PC = address HERE
FLAG = xxx1 xxxx
After Instruction
Since FLAG<4> = 1
PC = address FALSE

BTFSS Bit Test File, Skip if Set

Syntax: [*label*] BTFSS f, b, a

Operands: $0 \le f \le 255$
$0 \le b < 7$
$a \in [0,1]$

Operation: Skip if (f<b>) = 1

Status Affected: None

Encoding:

1010	bbba	ffff	ffff

Description: If bit 'b' in Register 'f' is '1' then the next instruction is skipped.
If bit 'b' is '1', then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2-cycle instruction.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1 (2 or 3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	No operation

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by a two word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example 1

```
HERE   BTFSS  FLAG, 4, 0   ; Test bit 4 of Register
FALSE  GOTO   PROCESS_CODE ; FLAG, and skip if set
TRUE   •
       •
       •
```

Case 1:
Before Instruction
PC = address HERE
FLAG = xxx0 xxxx
After Instruction
Since FLAG<4> = 0
PC = address FALSE

Case 2:
Before Instruction
PC = address HERE
FLAG = xxx1 xxxx
After Instruction
Since FLAG<4> = 1
PC = address TRUE

BTG Bit Toggle f

Syntax: [*label*] BTG f, b, a

Operands: $0 \le f \le 255$
$0 \le b \le 7$
$a \in [0,1]$

Operation: $(\overline{f<b>}) \rightarrow f<b>$

Status Affected: None

Encoding:

0111	bbba	ffff	ffff

Description: Bit 'b' in Register 'f' is toggled.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write Register 'f'

Example 1

```
BTG  LATC, 7, 1   ; Toggle the value of bit 7
                  ; in the LATC Register
```

Before Instruction
LATC = 0x0A ; **0**000 1010

After Instruction
LATC = 0x8A ; **1**000 1010

Example 2

```
BTG  INDF0, 3, 1  ; Toggle the value of bit 3
                  ; in the register pointed to
                  ; by the value in the FSR0
                  ; (FSR0H:FSR0L) Register
```

Before Instruction
FSR0 = 0xAC2
Contents of Address
(FSR0)= 0x20 ; 0010 **0**000

After Instruction
FSR0 = 0xAC2
Contents of Address
(FSR0)= 0x28 ; 0010 **1**000

BZ — Branch if Zero

Syntax: [*label*] BZ n

Operands: -128 ≤ f ≤ 127

Operation: If zero bit is '1'
(PC + 2) + 2n → PC

Status Affected: None

Encoding:

1110	0000	nnnn	nnnn

Description: If the Zero bit is '1', then the program will branch.
The 2's complement number '2n' (the offset) is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.

Words: 1

Cycles: 1 (2)

Q Cycle Activity:

If Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	Write to PC
No operation	No operation	No operation	No operation

If No Branch

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process data	No operation

Example 1

```
HERE    BZ      Z_CODE          ; If zero bit is clear
ZERO    •                       ; execute this code.
        •                       ;
        GOTO    MORE_CODE       ;
Z_CODE  •                       ; else if zero bit is set
        •                       ; this code will execute
```

Case 1: Before Instruction
PC = address HERE
Z = 0
After Instruction
PC = address ZERO

Case 2: Before Instruction
PC = address HERE
Z = 1
After Instruction
PC = address Z_CODE

Example 2

```
HERE    BZ     $ + OFFSET    ; If zero bit is set,
NZERO   GOTO   PROCESS_CODE  ; branch to HERE + OFFSET
PLUS0   •
PLUS1   •
PLUS2   •
PLUS3   •
PLUS4   •
PLUS5   •
PLUS6   •
```

Case 1: Before Instruction
PC = address HERE
Z = 0
After Instruction
PC = address NZERO

Case 2: Before Instruction
PC = address HERE
Z = 1
After Instruction
PC = address HERE + OFFSET

Example 3

```
MIN6    •
MIN5    •
MIN4    •
MIN3    •
MIN2    •
MIN1    •
MIN0    •                  ; If zero bit is set,
HERE    BZ     $ - OFFSET  ; branch to HERE - OFFSET
NZERO   GOTO   PROCESS_CODE
```

Case 1: Before Instruction
PC = address HERE
Z = 0
After Instruction
PC = address NZERO

Case 2: Before Instruction
PC = address HERE
Z = 1
After Instruction
PC = address HERE - OFFSET

Note: Assembler will convert the specified address label into the offset to be used.

CALL — Call Subroutine

Syntax: [*label*] CALL k, s

Operands: $0 \le k \le 1048575$
$s \in [0,1]$

Operation: (PC)+ 4 → TOS,
k → PC<20:1>,
0 → PC<0>,
if s = 1
(WREG) → WREGS,
(STATUS) → STATUSS,
(BSR) → BSRS

Status Affected: None

Encoding:

1st word (k<7:0>)	1110	110s	$k_7$kkk	kkkk_0
2nd word (k<19:8>)	1111	k_{19}kkk	kkkk	kkkk_8

Description: Subroutine call of entire 2M byte memory range. First, return address (PC+ 4) is pushed onto the return stack (20-bits wide).

If 's' = 1, the WREG, STATUS and BSR Registers are also pushed into their respective Shadow Registers, WREGS, STATUSS and BSRS.

If 's' = 0, no update occurs.

Then the 20-bit value 'k' is loaded into PC<20:1>. CALL is a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1

```
HERE   CALL  THERE, 1     ; Call subroutine THERE.
                          ; This is a fast call so
                          ; the BSR, WREG, and STATUS
                          ; Registers are forced onto
                          ; the Fast Register Stack
```

Before Instruction
PC = Address HERE
After Instruction
TOS = Address HERE+4
PC = Address THERE
WREGS = WREG
BSRS = BSR
STATUSS = STATUS

Example 2

```
HERE   CALL  THERE, 0     ; Call subroutine THERE.
                          ; This is NOT a fast call
```

Before Instruction
PC = Address HERE
WREGS = 0x45
BSRS = 0x29
STATUSS = 0x01
After Instruction
TOS = Address HERE+4
PC = Address THERE
WREGS = 0x45 (unchanged)
BSRS = 0x29 (unchanged)
STATUSS = 0x01 (unchanged)

CLRF Clear f

Syntax: [*label*] CLRF f, a

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: $00h \rightarrow f$
$1 \rightarrow Z$

Status Affected: Z

Encoding:

0110	101a	ffff	ffff

Description: The contents of Register 'f' are cleared and the Z bit is set.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write Register 'f'

Example 1

```
CLRF  FLAG_REG, 1  ; Clear Register FLAG_REG
```

Before Instruction
FLAG_REG = 0x5A
Z = x
After Instruction
FLAG_REG = 0x00
Z = 1

Example 2

```
CLRF  INDF0, 1     ; Clear the register pointed
                   ; to by the FSR0
                   ; (FSR0H:FSR0L) Register
```

Before Instruction
FSR0 = 0x0C2
Contents of Address
(FSR0) = 0xAA
Z = x
After Instruction
FSR0 = 0x0C2
Contents of Address
(FSR0) = 0x00
Z = 1

CLRWDT — Clear Watchdog Timer

Syntax: [*label*] CLRWDT

Operands: None

Operation: 00h → WDT
0 → WDT prescaler count,
1 → $\overline{TO}$
1 → $\overline{PD}$

Status Affected: $\overline{TO}$, $\overline{PD}$

Encoding:

0000	0000	0000	0100

Description: CLRWDT instruction clears the Watchdog Timer. It also clears the postscaler count of the WDT. Status bits $\overline{TO}$ and $\overline{PD}$ are set.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	Clear WDT Counter

Example

```
CLRWDT          ; Clear the Watchdog
                ; Timer count value
```

Before Instruction
- WDT counter = x
- WDT postscaler count = 0
- WDT postscaler = 1:128
- $\overline{TO}$ = x
- $\overline{PD}$ = x

After Instruction
- WDT counter = 0x00
- WDT postscaler count = 0
- WDT postscaler = 1:128
- $\overline{TO}$ = 1
- $\overline{PD}$ = 1

Note: The CLRWDT instruction does not affect the assignment of the WDT postscaler.

COMF Complement f

Syntax: [*label*] COMF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: $(\bar{f}) \rightarrow$ destination

Status Affected: Z, N

Encoding:

0001	11da	ffff	ffff

Description: The contents of Register 'f' are 1's complemented.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

```
Example 1               COMF  REG1, 0, 1   ; Complement the value in
                                            ; Register REG1 and place the
                                            ; result in the WREG Register
      Case 1:   Before Instruction
                      REG1   =  0x13        ;  0001 0011
                      Z, N   =  x
                After Instruction
                      REG1   =  0x13
                      WREG   =  0xEC        ;  1110 1100
                      Z      =  0
                      N      =  1
      Case 2:   Before Instruction
                      REG1   =  0xFF        ;  1111 1111
                      Z, N   =  x
                After Instruction
                      REG1   =  0xFF
                      WREG   =  0x00        ;  0000 0000
                      Z      =  1
                      N      =  0
      Case 3:   Before Instruction
                      REG1   =  0x00        ;  0000 0000
                      Z, N   =  x
                After Instruction
                      REG1   =  0x00
                      WREG   =  0xFF        ;  1111 1111
                      Z      =  0
                      N      =  1
```

Example 2

```
COMF  INDF0, 1, 1  ; Complement the value in the
                   ; register pointed to by the
                   ; FSR0 (FSR0H:FSR0L)
                   ; Register, placing the
                   ; result in that register
```

Before Instruction
FSR0 = 0xFC2
Contents of Address
(FSR0) = 0xAA ; 1010 1010
Z, N = x
After Instruction
FSR0 = 0xFC2
Contents of Address
(FSR0) = 0x55 ; 0101 0101
Z = 0
N = 0

Example 3

```
COMF  REG1, 1, 1   ; Complement the value in
                   ; Register REG1 and place the
                   ; result in Register REG1
```

Before Instruction
REG1 = 0xFF ; 1111 1111
Z, N = x
After Instruction
REG1 = 0x00 ; 0000 0000
Z = 1
N = 0

CPFSEQ — Compare f with WREG, Skip if Equal (f = WREG)

Syntax: [*label*] CPFSEQ f, a

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: (f) - (WREG)
skip if (f) = (WREG)

Status Affected: None

Encoding:

0110	001a	ffff	ffff

Description: Compares the contents of Register 'f' to the contents of WREG Register by performing an unsigned subtraction.
If 'f' = WREG then the fetched instruction is discarded and a NOP is executed instead, making this a two-cycle instruction.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1 (2 or 3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	No operation

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by a two word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

```
Example         HERE    CPFSEQ   FLAG, 1       ; Compare the value in
                NEQUAL  GOTO     PROCESS_CODE  ; Register FLAG to the
                EQUAL   •                      ; WREG Register and skip
                        •                      ; the next program memory
                        •                      ; location if they are
                                               ; equal
```

Case 1:

```
Before Instruction
        PC     =  address  HERE
        FLAG   =  0x5A
        WREG   =  0x5A           ; FLAG - WREG = 0x00
After Instruction
        PC     =  address  EQUAL ; The two values were
                                 ; Equal
```

Case 2:

```
Before Instruction
        PC     =  address  HERE
        FLAG   =  0xA5
        WREG   =  0x5A           ; FLAG - WREG = 0x4B
After Instruction
        PC     =  address  NEQUAL; The two values were
                                 ; Not Equal
```

CPFSGT — Compare f with WREG, Skip if Greater Than (f > WREG)

Syntax: [*label*] CPFSGT f, a

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: (f) - (WREG)
skip if (f) > (WREG); (unsigned comparison)

Status Affected: None

Encoding:

0110	010a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of WREG Register by performing an unsigned subtraction.

If 'f' > WREG then the fetched instruction is discarded and a NOP is executed instead, making this a two-cycle instruction.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1 (2 or 3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	No operation

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by a two word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

```
Example         HERE    CPFSGT   FLAG, 1       ; Compare the value in
                NGT     GOTO     PROCESS_CODE  ; Register FLAG to the
                GT      •                      ; WREG Register and skip
                        •                      ; the next program memory
                        •                      ; location if
                                               ; FLAG > WREG
```

Case 1:

```
Before Instruction
        PC     =   address  HERE
        FLAG   =   0x5A
        WREG   =   0x5A           ; FLAG - WREG = 0x00
After Instruction
        PC     =   address  NGT    ; The two values were
                                   ; Equal (Not Greater Than)
```

Case 2:

```
Before Instruction
        PC     =   address  HERE
        FLAG   =   0xA5
        WREG   =   0x5A           ; FLAG - WREG = 0x4B
After Instruction
        PC     =   address  GT     ; FLAG > WREG, Skip
                                   ; the next instruction
```

CPFSLT — Compare f with WREG, Skip if Less Than (f < WREG)

Syntax: [*label*] CPFSLT f, a

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: (f) - (WREG); (unsigned comparison)
skip if (f) < (WREG)

Status Affected: None

Encoding:

0110	000a	ffff	ffff

Description: Compares the contents of data memory location 'f' to the contents of WREG Register by performing an unsigned subtraction.

If 'f' < WREG then the fetched instruction is discarded and an NOP is executed instead making this a two-cycle instruction.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1 (2 or 3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	No operation

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by a two word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

```
Example         HERE    CPFSLT   FLAG, 1       ; Compare the value in
                NLT     GOTO     PROCESS_CODE  ; Register FLAG to the
                LT      •                      ; WREG Register and skip
                        •                      ; the next program memory
                        •                      ; location if
                                               ; FLAG < WREG
```

Case 1:

```
Before Instruction
        PC     =  address  HERE
        FLAG   =  0x5A
        WREG   =  0x5A            ; FLAG - WREG = 0x00
After Instruction
        PC     =  address  NLT    ; the two values were
                                  ; Equal (Not less than)
```

Case 2:

```
Before Instruction
        PC     =  address  HERE
        FLAG   =  0x5A
        WREG   =  0xA5            ; FLAG - WREG = 0x4B
After Instruction
        PC     =  address  LT     ; FLAG < WREG, Skip
                                  ; the next instruction
```

DAW

Decimal Adjust WREG Register

Syntax: [*label*] DAW

Operands: None

Operation:
If [WREG<3:0> >9] or [DC = 1] then
(WREG<3:0>) + 6 → WREG<3:0>;
else
(WREG<3:0>) → WREG<3:0>;

If [WREG<7:4> >9] or [C = 1] then
(WREG<7:4>) + 6 → WREG<7:4>;
else
(WREG<7:4>) → WREG<7:4>;

Status Affected: C

Encoding:

0000	0000	0000	0111

Description: DAW adjusts the eight bit value in WREG resulting from the earlier addition of two variables (each in packed BCD format) and produces a correct packed BCD result.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register REG	Process data	Write to REG

```
Example         HERE     DAW                    ; Decimal Adjust WREG
    Case 1:     Before Instruction
                    WREG  =  0x0F              ; 0x0F is 15 decimal
                    C     =  x
                After Instruction
                    WREG  =  0x15
                    C     =  0
    Case 2:     Before Instruction
                    WREG  =  0x68              ; 0x68 is 104 decimal
                    C     =  x
                After Instruction
                    PC    =  0x04
                    C     =  1                 ; Carry to indicate
                                               ; decimal rollover
    Case 3:     Before Instruction             ;
                    WREG  =  C6                ; 0xC6 is 198 decimal
                    C     =  x
                After Instruction
                    PC    =  98
                    C     =  1                 ; Carry to indicate
                                               ; decimal rollover
```

DECF Decrement f

Syntax: [*label*] DECF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f) - 1 → destination

Status Affected: C, DC, Z, OV, N

Encoding:

0000	01da	ffff	ffff

Description: Decrement the contents of Register 'f'.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
DECF  CNT, 1, 1    ; Decrement Register CNT
```

Before Instruction
CNT = 0x01
C, DC, OV, N = x
Z = 0

After Instruction
CNT = 0x00
C = 0
DC = 0
Z = 1
OV = 0
N = 0

Example 2

```
DECF  INDF0, 1, 1  ; Decrement the register
                   ; pointed to by the FSR
                   ; (FSR0H:FSR0L) Register
```

Before Instruction
- FSR0 = 0x1C2
- Contents of Address (FSR0) = 0x01
- C, DC, OV, N = x
- Z = 0

After Instruction
- FSR0 = 0x1C2
- Contents of Address (FSR0) = 0x00
- C = 0
- DC = 0
- Z = 1
- OV = 0
- N = 0

Example 3

```
DECF  CNT, 0, 1    ; Decrement Register CNT
                   ; WREG is destination
```

Before Instruction
- CNT = 0x10
- WREG = x
- Z = 0

After Instruction
- CNT = 0x10
- WREG = 0x0F
- C = 0
- DC = 1
- Z = 0
- OV = 0
- N = 0

DECFSZ Decrement f, Skip if 0

Syntax: [*label*] DECFSZ f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f) - 1 → destination; skip if result = 0

Status Affected: None

Encoding:

0010	11da	ffff	ffff

Description: The contents of Register 'f' are decremented. If the result is 0, then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2 cycle instruction.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1 (2 or 3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2 word instruction

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

```
Example         HERE    DECFSZ  CNT, 1, 1    ; Decrement Register CNT,
                        GOTO    LOOP         ; if CNT then equals 0
                CONTINUE •                   ; skip the next
                         •                   ; instruction
```

Case 1: Before Instruction
PC = address HERE
CNT = 0x01
After Instruction
CNT = 0x00
PC = address CONTINUE

Case 2: Before Instruction
PC = address HERE
CNT = 0x02
After Instruction
CNT = 0x01
PC = address HERE + 2

DCFSNZ Decrement f, Skip if Not 0

Syntax: [*label*] DCFSNZ f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f) - 1 → destination; skip if result ≠ 0

Status Affected: None

Encoding:

0100	11da	ffff	ffff

Description: The contents of Register 'f' are decremented. If the result is not 0, then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2-cycle instruction.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1 (2 or 3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2 word instruction

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example

```
HERE     DCFSNZ  CNT, 1, 1    ; Decrement Register CNT,
         GOTO    LOOP         ; if CNT does not equal 0
CONTINUE •                    ; skip the next instruction
         •
```

Case 1:

Before Instruction
PC = address HERE
CNT = 0x01
After Instruction
CNT = 0x00
PC = address HERE + 2

Case 2:

Before Instruction
PC = address HERE
CNT = 0x02
After Instruction
CNT = 0x01
PC = address CONTINUE

GOTO — Unconditional Branch

Syntax: [*label*] GOTO k

Operands: $0 \le k \le 1048575$

Operation: $k \rightarrow PC<20:1>$
$0 \rightarrow PC<0>$,

Status Affected: None

Encoding:

1st word (k<7:0>)	1110	1111	$k_7$kkk	kkkk_0
2nd word (k<19:8>)	1111	k_{19}kkk	kkkk	kkkk_8

Description: GOTO allows an unconditional branch anywhere within the entire 2M byte memory range. The 20-bit immediate value 'k' is loaded into PC<20:1>. GOTO is always a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>	Process data	No operation

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1
```
HERE   GOTO   THERE        ; Goto address THERE
```
After Instruction
PC = Address THERE

Example 2
```
HERE   GOTO   $-2          ; GOTO address HERE - 2
```
After Instruction
PC = Address HERE -2

Example 3
```
HERE   GOTO   $            ; GOTO address HERE
                           ; (infinite loop)
```
After Instruction
PC = Address HERE

Example 4
```
HERE   GOTO   HERE         ; GOTO address HERE
                           ; (infinite loop)
```
After Instruction
PC = Address HERE

INCF Increment f

Syntax: [*label*] INCF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f) + 1 → destination

Status Affected: C, DC, Z, OV, N

Encoding:

0010	10da	ffff	ffff

Description: The contents of Register 'f' are incremented.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
        INCF  CNT,1, 1     ; Increment Register CNT
Before Instruction
        CNT         =  0xFF  ;  1111 1111
        C, DC, Z, OV, N  =  x
After Instruction
        CNT         =  0x00  ;  0000 0000
        C           =  1
        DC          =  1
        Z           =  1
        OV          =  0
        N           =  0
```

Example 2

```
        INCF  INDF0, 1, 1  ; Increment Register
                           ; indirectly
                           ; (FSR0 (FSR0H:FSR0L) points
                           ; to address to increment)
```

Before Instruction

```
        FSR0          = 0x0C2
        Contents of Address
           (FSR0)     = 0xFF ;  1111 1111
        C, DC, Z, OV, N = x
```

After Instruction

```
        FSR0       =  0x0C2
        Contents of Address
           (FSR0)  =  0x00   ;  0000 0000
        C          =  1
        DC         =  1
        Z          =  1
        OV         =  0
        N          =  0
```

Example 3

```
        INCF  CNT, 0, 1    ; Increment Register CNT
                           ; place result in WREG
```

Before Instruction

```
        CNT        =  0x10  ;  0001 0000
        WREG       =  x
        C, DC, OV, N =  x
        Z          =  0
```

After Instruction

```
        CNT        =  0x10
        WREG       =  0x11  ;  0001 0001
        C          =  0
        DC         =  0
        Z          =  0
        OV         =  0
        N          =  0
```

INCFSZ Increment f, Skip if 0

Syntax: [*label*] INCFSZ f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f) + 1 → destination, skip if result = 0

Status Affected: None

Encoding:

0011	11da	ffff	ffff

Description: The contents of Register 'f' are incremented. If the result is 0, then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2-cycle instruction.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1 (2 or 3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2 word instruction

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example 1

```
HERE    INCFSZ  CNT, 1, 1     ; Increment Register CNT,
NZERO   GOTO    LOOP          ; if CNT then equals 0
ZERO    •                     ; skip the next
        •                     ; instruction
```

Case 1: Before Instruction
PC = address HERE
CNT = 0xFF
After Instruction
CNT = 0x00
PC = address ZERO

Case 2: Before Instruction
PC = address HERE
CNT = 0x00
After Instruction
CNT = 0x01
PC = address NZERO

Example 2

```
HERE    INCFSZ  CNT, 1, 0     ; Increment Register CNT,
NZERO   GOTO    LOOP          ; if CNT equals 0
ZERO    •                     ; skip the next
        •                     ; instruction
        •
```

Case 1: Before Instruction
PC = address HERE
CNT = 0xFF ; In access bank
After Instruction
CNT = 0x00
PC = address ZERO

Case 2: Before Instruction
PC = address HERE
CNT = 0x00 ; In access bank
After Instruction
CNT = 0x01
PC = address NZERO

INFSNZ Increment f, Skip if Not 0

Syntax: [*label*] INFSNZ f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f) + 1 → destination, skip if result ≠ 0

Status Affected: None

Encoding:

0100	10da	ffff	ffff

Description: The contents of Register 'f' are incremented. If the result is not 0, then the next instruction (fetched during the current instruction execution) is discarded and a NOP is executed instead, making this a 2-cycle instruction.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1 (2 or 3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by 2 word instruction

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example 1

```
HERE    INFSNZ  CNT, 1, 1  ; Increment Register CNT,
ZERO    GOTO    LOOP       ; if CNT does not equal 0
NZERO   •                  ; skip the next instruction
        •
```

Case 1: Before Instruction
PC = address HERE
CNT = 0xFF
After Instruction
CNT = 0x00
PC = address ZERO

Case 2: Before Instruction
PC = address HERE
CNT = 0x00
After Instruction
CNT = 0x01
PC = address NZERO

Example 2

```
HERE    INFSNZ  CNT, 1, 0  ; Increment Register CNT,
ZERO    GOTO    LOOP       ; if CNT does not equal 0
NZERO   •                  ; skip the next instruction
        •
        •
```

Case 1: Before Instruction
PC = address HERE
CNT = 0xFF ; In access bank
After Instruction
CNT = 0x00
PC = address ZERO

Case 2: Before Instruction
PC = address HERE
CNT = 0x00 ; In access bank
After Instruction
CNT = 0x01
PC = address NZERO

IORLW Inclusive OR Literal with WREG

Syntax: [*label*] IORLW k

Operands: $0 \le k \le 255$

Operation: (WREG).OR. k → WREG

Status Affected: Z, N

Encoding:

0000	1001	kkkk	kkkk

Description: The contents of the WREG Register is OR'd with the eight bit literal 'k'. The result is placed in the WREG Register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to WREG Register

Example 1

```
        IORLW  0x35          ; bit wise OR 35h with the
                             ; WREG Register
Before Instruction           ;  0011 0101   (35h)
    WREG  =  0x9A            ;  1001 1010
    Z, N  =  x
After Instruction
    WREG  =  0xBF            ;  1011 1111
    Z     =  0
    N     =  1
```

Example 2

```
        IORLW  MYREG         ; bit wise OR the value of
                             ; the address of Register
                             ; MYREG with the WREG
                             ; Register
Before Instruction           ;
    WREG  =  0x9A            ;  1001 1010
    Address of MYREG †       ;
          =  0x37            ;  0011 0111
    † MYREG is a symbol for a
    data memory location
    Z, N  =  x
After Instruction
    WREG  =  0xBF            ;  1011 1111
    Z     =  0
    N     =  1
```

Example 3

```
        IORLW HIGH (LU_TABLE) ; bit wise OR the value of
                              ; the high byte of address
                              ; LU_TABLE with the WREG
                              ; Register
Before Instruction
        WREG   =   0x9A       ;  1001 1010
        Address of LU_TABLE †
               =   0x9375     ;  1001 0011    (93h)
        † LU_TABLE is a label for an
        address in program memory
        Z, N   =   x
After Instruction
        WREG   =   0x9B
        Z      =   0          ;  1001 1011
        N      =   1
```

Example 4

```
        IORLW 0x00            ; bit wise OR 00h with the
                              ; WREG Register
Before Instruction            ;  0000 0000   (literal)
        WREG   =   0x00       ;  0000 0000
        Z, N   =   x
After Instruction
        WREG   =   0x00       ;  0000 0000
        Z      =   1
        N      =   0
```


IORWF Inclusive OR WREG with f

Syntax: [*label*] IORWF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (WREG).OR. (f) → destination

Status Affected: Z, N

Encoding:

0001	00da	ffff	ffff

Description: Inclusive OR the WREG Register with the contents of Register 'f'.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
IORWF  INDF0, 1, 1  ; Comment
```

Before Instruction
WREG = 0x17 ; 0001 0111
FSR0 = 0xDC2
Contents of Address
(FSR0) = 0x30 ; 0011 0000
Z, N = x

After Instruction
WREG = 0x17
FSR0 = 0xDC2
Contents of Address
(FSR0) = 0x37 ; 0011 0111
Z = 0
N = 0

```
Example 2               IORWF  RESULT, 1, 1 ; bit wise OR the WREG
                                            ; Register with the
                                            ; Register RESULT.
     Case 1:   Before Instruction
                     RESULT  =  0x13        ;  0001 0011
                     WREG    =  0x91        ;  1001 0001
                     Z, N    =  x
               After Instruction
                     RESULT  =  0x93        ;  1001 0011
                     WREG    =  0x91
                     Z       =  0
                     N       =  1
     Case 2:   Before Instruction
                     RESULT  =  0x00        ;  0000 0000
                     WREG    =  0x00        ;  0000 0000
                     Z, N    =  x
               After Instruction
                     RESULT  =  0x00        ;  0000 0000
                     WREG    =  0x00
                     Z       =  1
                     N       =  0

Example 3               IORWF  RESULT, 1, 0 ; bit wise OR the WREG
                                            ; Register with the
                                            ; register in the Access
                                            ; bank at address of RESULT
                                            ; Register.
     Case 1:   Before Instruction
                     RESULT  =  0x13
                     (RESULT) in access bank ;  1100 1000
                              = 0xC8         ;  1001 0001
                     WREG    =  0x91
                     Z, N    =  x
               After Instruction
                     RESULT  =  0x13        ;  1101 1001
                     (RESULT) in access bank
                              = 0xD9
                     WREG    =  0x91
                     Z       =  0
                     N       =  1
     Case 2:   Before Instruction
                     RESULT  =  0x00
                     (RESULT) in access bank ;  0001 0001
                              = 0x11         ;  0000 0000
                     WREG    =  0x00
                     Z, N    =  x
               After Instruction
                     RESULT  =  0x00        ;  0001 0001
                     (RESULT) in access bank
                              = 0x11
                     WREG    =  0x00
                     Z       =  0
                     N       =  0
```

LFSR Load 12-bit Literal to FSR

Syntax: [*label*] LFSR f, k

Operands: $0 \le f \le 2$
$0 \le k \le 4095$

Operation: $k \rightarrow FSRx$

Status Affected: None

Encoding:

1st word	1110	1110	00ff	$k_{11}kkk_8$
2nd word	1111	0000	k_7kkk	kkk_0

Description: The 12-bit literal 'k' is loaded into the File Select Register (FSR Register) pointed to by 'f':
f = 00 → FSR0
f = 01 → FSR1
f = 10 → FSR2
f = 11 → Reserved

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal k_{11}:k_8	Process data	Write to FSRxH Register k_{11}:k_8
Decode	Read literal k_7:k_0	Process data	Write to FSRxL Register k_7:k_0

```
Example 1               LFSR   2, 0x123     ; Load the 12-bit FSR2 with
                                            ; 123h
```

Before Instruction
FSR0H = 0x05
FSR0L = 0xA5
FSR1H = 0x05
FSR1L = 0xA5
FSR2H = 0x05
FSR2L = 0xA5

After Instruction
FSR0H = 0x05
FSR0L = 0xA5
FSR1H = 0x05
FSR1L = 0xA5
FSR2H = **0x01**
FSR2L = **0x23**

```
Example 2               LFSR    0, 0xFE3     ; Load the 12-bit FSR0 with
                                             ; FE3h
```

Before Instruction

FSR0H = 0x05
FSR0L = 0xA5
FSR1H = 0x05
FSR1L = 0xA5
FSR2H = 0x05
FSR2L = 0xA5

After Instruction

FSR0H = **0x0F**
FSR0L = **0xE3**
FSR1H = 0x05
FSR1L = 0xA5
FSR2H = 0x05
FSR2L = 0xA5

```
Example 3               LFSR    1, 0xFE3     ; Load the 12-bit FSR1 with
                                             ; FE3h
```

Before Instruction

FSR0H = 0x05
FSR0L = 0xA5
FSR1H = 0x05
FSR1L = 0xA5
FSR2H = 0x05
FSR2L = 0xA5

After Instruction

FSR0H = 0x05
FSR0L = 0xA5
FSR1H = **0x0F**
FSR1L = **0xE3**
FSR2H = 0x05
FSR2L = 0xA5

MOVF Move f

Syntax: [*label*] MOVF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: f → destination

Status Affected: Z, N

Encoding:

0101	00da	ffff	ffff

Description: The contents of Register 'f' is moved to a destination dependent upon the status of the 'd' and 'a' bits.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to WREG Register

Example 1

```
MOVF  MYREG, 0, 1  ; Copy the contents of
                   ; Register MYREG to the WREG
                   ; Register
```

Before Instruction
MYREG = 0x22
WREG = 0xFF
Z, N = x
After Instruction
MYREG = 0x22
WREG = 0x22
Z = 0
N = 0

Example 2

```
MOVF  MYREG, 1, 1  ; Copy the contents of
                   ; Register MYREG to itself
                   ; (affects the status bits)
```

Before Instruction
MYREG = 0x00
WREG = 0x10
Z, N = x
After Instruction
MYREG = 0x00
WREG = 0x10
Z = 1
N = 0

```
Example 3               MOVF  MYREG, 1, 0  ; Copy the contents of
                                           ; Register MYREG in the
                                           ; access bank to itself
                                           ; (affects the status bits)
```

Case 1: Before Instruction

```
        MYREG  =  0x00      ; In access bank
        WREG   =  0x10
        Z, N   =  x
```

After Instruction

```
        MYREG  =  0x00      ; In access bank
        WREG   =  0x10
        Z      =  1
        N      =  0
```

Case 2: Before Instruction

```
        MYREG  =  0x80      ; In access bank
        WREG   =  0x10
        Z, N   =  x
```

After Instruction

```
        MYREG  =  0x80      ; In access bank
        WREG   =  0x10
        Z      =  0
        N      =  1
```


MOVFF Move f to f

Syntax: [*label*] MOVFF f_s, f_d

Operands: $0 \le f_s \le 4095$
$0 \le f_d \le 4095$

Operation: $(f_s) \rightarrow f_d$

Status Affected: None

Encoding:

1st word (source)	1100	ffff	ffff	ffff
2nd word (destination)	1111	ffff	ffff	ffff

Description: The contents of Source Register 'f_s' are moved to Destination Register 'f_d'. Location of source 'f_s' can be anywhere in the 4096 byte data space (000h to FFFh), and location of destination 'f_d' can also be anywhere from 000h to FFFh.

MOVFF is particularly useful for transferring a data memory location to a Peripheral Register (such as the transmit buffer or an I/O port) without affecting the WREG Register.

Note: The MOVFF instruction cannot use the PCL, TOSU, TOSH, and TOSL as the Destination Register

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Source Register f	Process data	No operation
Decode	No operation	No operation	Write to destination Register f

Example 1

```
MOVFF   REG1, REG2  ; Copy the contents of
                    ; Register REG1 to Register
                    ; REG2
```

Before Instruction
REG1 = 0x33
REG2 = 0x11
After Instruction
REG1 = 0x33
REG2 = 0x33

Example 2

```
MOVFF   REG2, REG1  ; Copy the contents of
                    ; Register REG2 to Register
                    ; REG1
```

Before Instruction
REG1 = 0x33
REG2 = 0x11
After Instruction
REG1 = 0x11
REG2 = 0x11

MOVLB Move Literal to low nibble in BSR

Syntax: [*label*] MOVLB k

Operands: $0 \le k \le 15$

Operation: $k \rightarrow BSR<3:0>$

Status Affected: None

Encoding:

0000	0001	0000	kkkk

Description: The 4-bit literal 'k' is loaded into the Bank Select Register (BSR).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write literal 'k' to BSR

```
Example 1                MOVLB  5          ; Modify Least Significant
                                           ; nibble of BSR Register
                                           ; to value 5
        Before Instruction
                BSR  =   0x02
        After Instruction
                BSR  =   0x05

Example 2                MOVLB  9          ; Modify Least Significant
                                           ; nibble of BSR Register
                                           ; to value 9
        Before Instruction
                BSR  =   0x0F
        After Instruction
                BSR  =   0x09
```

MOVLW Move Literal to WREG

Syntax: [*label*] MOVLW k

Operands: $0 \le k \le 255$

Operation: $k \rightarrow$ WREG

Status Affected: None

Encoding:

0000	1110	kkkk	kkkk

Description: The eight bit literal 'k' is loaded into WREG Register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to WREG Register

Example 1

```
MOVLW  0x5A          ; Load the WREG Register
                     ; with the value 5Ah
```

Before Instruction
WREG = x
After Instruction
WREG = 0x5A

Example 2

```
MOVLW  MYREG         ; Load the WREG Register
                     ; with the value of the
                     ; address of MYREG
```

Before Instruction
WREG = 0x10
Address of MYREG † = 0x37
† MYREG is a symbol for a data memory location
After Instruction
WREG = 0x37

Example 3

```
MOVLW HIGH (LU_TABLE) ; Load the WREG Register
                      ; with the value of the high
                      ; byte of address LU_TABLE
```

Before Instruction
WREG = 0x10
Address of LU_TABLE † = 0x9375
† LU_TABLE is a label for an address in program memory
After Instruction
WREG = 0x93

MOVWF Move WREG to f

Syntax: [*label*] MOVWF f, a

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: (WREG) → f

Status Affected: None

Encoding:

0110	111a	ffff	ffff

Description: Move data from WREG Register to Register 'f'.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read WREG Register	Process data	Write Register 'f'

Example 1

```
MOVWF   OPTION_REG, 1; Copy the value in the WREG
                     ; Register to the
                     ; OPTION_REG Register
```

Before Instruction
OPTION_REG = 0xFF
WREG = 0x4F
After Instruction
OPTION_REG = 0x4F
WREG = 0x4F

Example 2

```
MOVWF    INDF0, 1     ; Copy the value in the WREG
                      ; Register to the
                      ; FSR0 (FSR0H:FSR0L)
                      ; Register
```

Before Instruction
WREG = 0x17
FSR0 = 0x5C2
Contents of Address (FSR0) = 0x00
After Instruction
WREG = 0x17
FSR0 = 0x5C2
Contents of Address (FSR0) = 0x17

MULLW Multiply Literal with WREG

Syntax: [*label*] MULLW k

Operands: $0 \le k \le 255$

Operation: (WREG) x k → PRODH:PRODL

Status Affected: None

Encoding:

0000	1101	kkkk	kkkk

Description: An unsigned multiplication is carried out between the contents of WREG and the 8-bit literal 'k'. The 16-bit result is placed in PRODH:PRODL Register Pair. PRODH contains the high byte.

WREG is unchanged.

None of the status flags are affected.

Neither an overflow nor carry is possible in this operation. A zero result is possible but not detected.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write Registers PRODH: PRODL

Example 1

```
MULLW   0xC4        ; Multiply the WREG Register
                    ; with the constant value
                    ; C4h
```

Before Instruction
WREG = 0xE2
PRODH = x
PRODL = x
After Instruction
WREG = 0xE2
PRODH = 0xAD
PRODL = 0x08

Example 2

```
MULLW   FACTOR      ; Multiply the WREG Register
                    ; with the constant value
                    ; FACTOR
```

Before Instruction
FACTOR = 0xC4
WREG = 0xE2
PRODH = x
PRODL = x
After Instruction
WREG = 0xE2
PRODH = 0xAD
PRODL = 0x08

MULWF Multiply WREG with f

Syntax: [*label*] MULWF f, a

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: (WREG) x (f) → PRODH:PRODL

Status Affected: None

Encoding:

0000	001a	ffff	ffff

Description: An unsigned multiplication is carried out between the contents of WREG and the value in Register File Location 'f'. The 16-bit result is placed in the PRODH:PRODL Register Pair. PRODH contains the high byte.

Both WREG and 'f' are unchanged.

None of the status flags are affected.

Neither an overflow nor carry is possible in this operation. A zero result is possible but not detected.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write Registers PRODH: PRODL

Example

```
MULWF   MYREG, 1     ; Multiple the WREG
                     ; Register with the value
                     ; in MYREG Register
```

Before Instruction
WREG = 0xE2
MYREG = 0xB5
PRODH = x
PRODL = x

After Instruction
WREG = 0xE2
MYREG = 0xB5
PRODH = 0x9F
PRODL = 0xCA

NEGF Negate f

Syntax: [*label*] NEGF f, a

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: $(\overline{f}) + 1 \rightarrow (f)$

Status Affected: C, DC, Z, OV, N

Encoding:

0110	110a	ffff	ffff

Description: Location 'f' is negated using two's complement. The result is placed in the data memory location 'f'.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write Register 'f'

```
Example 1               NEGF  MYREG, 1     ; 2's complement the value in
                                           ; MYREG
```

Case 1:
```
Before Instruction
        MYREG       = 0x3A  ;  0011 1010
        C, DC, Z, OV, N = x
After Instruction
        MYREG       = 0xC6  ;  1100 0110
        C           = 0
        DC          = 0
        Z           = 0
        OV          = 0
        N           = 1
```

Case 2:
```
Before Instruction
        MYREG       = 0xB0  ;  1011 0000
        C, DC, Z, OV, N = x
After Instruction
        MYREG       = 0x50  ;  0101 0000
        C           = 0
        DC          = 1
        Z           = 0
        OV          = 0
        N           = 0
```

Case 3:
```
Before Instruction
        MYREG       = 0x00  ;  0000 0000
        C, DC, Z, OV, N = x
After Instruction
        MYREG       = 0x00  ;  0000 0000
        C           = 1
        DC          = 1
        Z           = 1
        OV          = 0
        N           = 0
```

NOP No Operation

Syntax: [*label*] NOP

Operands: None

Operation: No operation

Status Affected: None

Encoding:

Default	0000	0000	0000	0000
Used with 2 word instructions	1111	xxxx	xxxx	xxxx

Description: No operation.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation

Example

```
HERE    NOP             ; This instruction cycle
                        ; does nothing
                        ; (No Operation)
```

Before Instruction
PC = address HERE
After Instruction
PC = address HERE + 2

POP

POP Top of Return Stack

Syntax: [*label*] POP

Operands: None

Operation: (TOS) → bit bucket

Status Affected: None

Encoding:

0000	0000	0000	0110

Description: The Top of Stack (TOS) value is pulled off the return stack and is discarded. The TOS value then becomes the previous value that was pushed onto the return stack.

This instruction is provided to enable the user to manage the return stack to incorporate a software stack.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	POP TOS value

Example

```
HERE    POP                 ; Modify the Top of Stack
                            ; (TOS). The TOS points to
                            ; what was one level down
```

Before Instruction
- TOS = 0x0031A2
- Stack (1 level down) = 0x014332

After Instruction
- TOS = 0x014332
- PC = HERE + 2

PUSH PUSH Top of Return Stack

Syntax: [*label*] PUSH

Operands: None

Operation: PC → (TOS)

Status Affected: None

Encoding:

0000	0000	0000	0101

Description: The previous Top of Stack (TOS) value is pushed down on the stack. The PC is pushed onto the top of the return stack.

This instruction is provided to enable the user to manage the return stack to incorporate a software stack.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	PUSH PC onto return stack	No operation	No operation

Example

```
HERE    PUSH                    ; PUSH current Program
                                ; Counter value onto the
                                ; hardware stack
```

Before Instruction
PC = 0x000124
TOS = 0x00345A
After Instruction
PC = 0x000126
TOS = 0x000124
Stack (1 level down)
= 0x00345A

RCALL Relative Call

Syntax: [*label*] RCALL n

Operands: -1024 ≤ n ≤ 1023

Operation: (PC + 2) → TOS,
(PC + 2) + 2n → PC

Status Affected: None

Encoding:

1101	1nnn	nnnn	nnnn

Description: Subroutine call with a jump up to 1K from the current location. First, the return address (PC+2) is pushed onto the stack. Then the 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is a two-cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'n', Push PC to stack	Process Data	Write to PC
No operation	No operation	No operation	No operation

Example 1

```
HERE    RCALL   Sub1        ; Call a program memory
                            ; location (Sub1)
                            ; this location must be
                            ; < 1024 locations forward or
                            ; > 1025 locations backward
```

Before Instruction
PC = Address (HERE)
TOS = 0x0031A2
After Instruction
PC = Address (Sub1)
TOS = Address (HERE + 2)
Stack (1 level down)
= 0x0031A2

Example 2

```
HERE    RCALL $ + OFFSET    ; Call to HERE+OFFSET
PLUS0   •
PLUS1   •
PLUS2   •
PLUS3   •
PLUS4   •
PLUS5   •
PLUS6   •
```

Before Instruction
PC = address HERE
After Instruction
PC = address HERE + OFFSET

Example 3

```
MIN6   •
MIN5   •
MIN4   •
MIN3   •
MIN2   •
MIN1   •
MIN0   •                    ; Call to HERE-OFFSET
HERE   RCALL $ - OFFSET
NEXT
```

Before Instruction
PC = address HERE
After Instruction
PC = address HERE - OFFSET

RESET **Reset Device**

Syntax: [*label*] RESET

Operands: None

Operation: Force all registers and flag bits that are affected by a $\overline{\text{MCLR}}$ reset to their reset condition.

Status Affected: All

Encoding:

0000	0000	1111	1111

Description: This instruction provides a way to execute a software reset.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	Start reset

Example

```
HERE    RESET               ; Do a software reset
```

Before Instruction
- PC = address HERE
- C, DC, Z, OV, N = x

After Instruction
- PC = 0x000000
- SFRs = See reset section
- GPRs = u (unchanged)

RETFIE Return from Interrupt

Syntax: [*label*] RETFIE s

Operands: s ∈ [0,1]

Operation:
(TOS) → PC,
if IPEN = 0 (compatibility mode)
1 → GIE
if IPEN = 1

GIEH	GEIL	
1	1	1 → Invalid
1	0	1 → GIEL
0	1	1 → GIEH
0	0	1 → GIEH

if s = 1
(WREGS) → WREG
(STATUSS) → STATUS
(BSRS) → BSR
if s = 0
(WREGS) → unchanged
(STATUSS) → unchanged
(BSRS) → unchanged

In both cases PCLATU, PCLATH are unchanged.

Status Affected: GIE/GIEH,PEIE/GIEL

Encoding:

0000	0000	0001	000s

Description: Return from Interrupt. Stack is popped and Top of Stack (TOS) is loaded into the PC. Interrupts are enabled by setting either the high or low priority global interrupt enable bits (GIEH or GIEL).

If 's' = 1, the contents of the Shadow Registers WREGS, STATUSS and BSRS are loaded into their corresponding registers, WREG, STATUS and BSR.

If 's' = 0, no update of these registers occurs (default).

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	No operation	Set GIEH or GIEL	POP PC from stack

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1

```
HERE    RETFIE   0          ; Return from interrupt,
                            ; enable interrupts
```

Before Instruction
- PC = address HERE
- GIE/GIEH, PEIE/GIEL = x
- WREG = x
- BSR = x
- STATUS = x

After Instruction
- PC = TOS
- GIE/GIEH, PEIE/GIEL = 1
- WREG = unchanged
- BSR = unchanged
- STATUS = unchanged

Example 2

```
HERE    RETFIE   1          ; Return from interrupt,
                            ; enable interrupts.
                            ; This is a fast return so
                            ; the BSR, WREG, and STATUS
                            ; Registers are restored
                            ; with the values in the
                            ; Fast Register Stack
```

Before Instruction
- PC = address HERE
- GIE/GIEH, PEIE/GIEL = x
- WREG = x
- BSR = x
- STATUS = x

After Instruction
- PC = TOS
- GIE/GIEH, PEIE/GIEL = 1
- WREG = WREGS
- BSR = BSRS
- STATUS = STATUSS

RETLW Return with Literal in W

Syntax: [*label*] RETLW k

Operands: $0 \le k \le 255$

Operation: $k \rightarrow$ WREG;
TOS $\rightarrow$ PC
PCLATU and PCLATH are unchanged

Status Affected: None

Encoding:

0000	1100	kkkk	kkkk

Description: The WREG Register is loaded with the eight bit literal 'k'. The program counter is loaded from the Top of Stack (the return address). The upper and high address latches (PCLATU:PCLATH) remain unchanged. This is a two-cycle instruction.

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	POP PC from stack, write to WREG

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example

```
HERE   CALL  TABLE      ; WREG contains table offset
       •                ; value WREG now has table
       •                ; value
       •
TABLE  ADDWF PC         ; WREG = offset
       RETLW k1         ; Begin table,
       RETLW k2         ; Return with constant in WREG
       •
       •
       •
       RETLW kn         ; End of table
```

Before Instruction
WREG = x
After Instruction
WREG = value of kx
PC = TOS = Address HERE + 2

RETURN Return from Subroutine

Syntax: [*label*] RETURN s

Operands: s ∈ [0,1]

Operation: (TOS) → PC
if s = 1
(WREGS) → WREG
(STATUSS) → STATUS
(BSRS) → BSR
if s = 0
(WREGS) → unchanged
(STATUSS) → unchanged
(BSRS) → unchanged

In both cases PCLATU and PCLATH are unchanged

Status Affected: None

Encoding:

0000	0000	0001	001s

Description: Return from subroutine. The stack is popped and the Top of Stack (TOS) is loaded into the program counter.
If 's' = 1, the contents of the Shadow Registers WREGS, STATUSS and BSRS are loaded into their corresponding registers, WREG, STATUS and BSR.
If 's' = 0, no update of these registers occurs (default).

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	No operation	Process data	POP PC from stack

2nd cycle:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

Example 1

```
HERE    RETURN   0        ; Return from subroutine.
```

Before Instruction
PC = address HERE
WREG = x
BSR = x
STATUS = x
After Instruction
PC = TOS
WREG = unchanged
BSR = unchanged
STATUS = unchanged

Example 2

```
HERE    RETURN   1        ; Return from subroutine.
                          ; This is a fast return so
                          ; the BSR, WREG, and STATUS
                          ; Registers are restored
                          ; with the values in the
                          ; Fast Register Stack
```

Before Instruction
- PC = address HERE
- WREG = x
- BSR = x
- STATUS = x

After Instruction
- PC = TOS
- WREG = WREGS
- BSR = BSRS
- STATUS = STATUSS

RLCF Rotate Left f through Carry

Syntax: [*label*] RLCF f, d, a

Operands: $0 \le f \le 127$
$d \in [0,1]$
$a \in [0,1]$

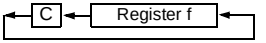
Operation: See description below

Status Affected: C, Z, N

Encoding:

0011	01da	ffff	ffff

Description: The contents of Register 'f' are rotated one bit to the left through the Carry Flag.



The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
RLCF  REG1, 0, 1   ; Rotate the value in REG1
                   ; 1 bit position left and
                   ; the carry bit loads into
                   ; bit 0. Then place the
                   ; result in the WREG
                   ; Register
```

Before Instruction
REG1 = 1110 0110
C = 0
Z, N = x
After Instruction
REG1 = 1110 0110
WREG = 1100 1100
C = 1
Z = 0
N = 1

```
Example 2               RLF  INDF0, 1, 1  ; Rotate the value in the
                                          ; register pointed by the
                                          ; FSR0 (FSR0H:FSR0L)
                                          ; Register 1 bit position
                                          ; left and place the result
                                          ; back into that register
                                          ; Carry loads into bit 0
```

Case 1: Before Instruction

FSR0 = 0x0C2
Contents of Address (FSR0) = 0011 1010
C = 1
Z, N = x

After Instruction

FSR0 = 0x0C2
Contents of Address (FSR0) = 0111 0101
C = 0
Z = 0
N = 0

Case 2: Before Instruction

FSR0 = 0x0C2
Contents of Address (FSR0) = 1011 1001
C = 0
Z, N = x

After Instruction

FSR0 = 0x0C2
Contents of Address (FSR0) = 0111 0010
C = 1
Z = 0
N = 0

RLNCF Rotate Left f (No Carry)

Syntax: [*label*] RLNCF f, d, a

Operands: $0 \le f \le 127$
$d \in [0,1]$
$a \in [0,1]$

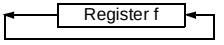
Operation: See description below

Status Affected: Z, N

Encoding:

0100	01da	ffff	ffff

Description: The contents of Register 'f' are rotated one bit to the left. The Carry Flag bit is not affected.



The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
RLNCF REG1, 0, 1   ; Rotate the value in REG1
                   ; 1 bit position left and
                   ; bit 7 loads into bit 0.
                   ; Then place the result in
                   ; the WREG Register
```

Before Instruction
```
REG1 =  1110 0110
Z, N =  x
```
After Instruction
```
REG1 =  1110 0110
WREG =  1100 1101
Z    =  0
N    =  1
```

```
Example 2           RLNCF INDF0, 1, 1  ; Rotate the value in the
                                       ; register pointed by the
                                       ; FSR0 (FSR0H:FSR0L)
                                       ; Register 1 bit position left
                                       ; and place the result in the
                                       ; back into that register.
                                       ; bit 7 loads into bit 0.
```

Case 1:
Before Instruction
FSR0 = 0x1C2
Contents of Address (FSR0) = 0011 1010
Z, N = x
After Instruction
FSR0 = 0x1C2
Contents of Address (FSR0) = 0111 0100
Z = 0
N = 0

Case 2:
Before Instruction
FSR0 = 0x1C2
Contents of Address (FSR0) = 1011 1001
Z, N = x
After Instruction
FSR0 = 0x1C2
Contents of Address (FSR0) = 0111 0011
Z = 0
N = 0

RRCF Rotate Right f through Carry

Syntax: [*label*] RRCF f, d, a

Operands: $0 \le f \le 127$
$d \in [0,1]$
$a \in [0,1]$

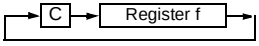
Operation: See description below

Status Affected: C, Z, N

Encoding:

0011	00da	ffff	ffff

Description: The contents of Register 'f' are rotated one bit to the right through the Carry Flag.



The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
RRCF  REG1, 0, 1   ; Rotate the value in REG1
                   ; 1 bit position right and
                   ; the carry bit loads into
                   ; bit 7. Then place the
                   ; result in the WREG
                   ; Register
```

Before Instruction
```
REG1  =   1110 0110
WREG  =   xxxx xxxx
C     =   0
Z, N  =   x
```
After Instruction
```
REG1  =   1110 0110
WREG  =   0111 0011
C     =   0
Z     =   0
N     =   0
```

```
Example 2               RRCF  INDF0, 1, 1  ; Rotate the value in the
                                           ; register pointed by the
                                           ; FSR0 (FSR0H:FSR0L)
                                           ; Register 1 bit position
                                           ; right and place the result
                                           ; in the back into that
                                           ; register.
                                           ; Carry loads into bit 7.
```

Case 1: Before Instruction
- FSR0 = 0x2C2
- Contents of Address (FSR0) = 0011 1010
- C = 1
- Z, N = x

After Instruction
- FSR0 = 0x2C2
- Contents of Address (FSR0) = 1001 1101
- C = 0
- Z = 0
- N = 1

Case 2: Before Instruction
- FSR0 = 0x2C2
- Contents of Address (FSR0) = 0011 1001
- C = 0
- Z, N = x

After Instruction
- FSR0 = 0x2C2
- Contents of Address (FSR0) = 0001 1100
- C = 1
- Z = 0
- N = 0

RRNCF Rotate Right f (No Carry)

Syntax: [*label*] RRNCF f, d, a

Operands: $0 \le f \le 127$
$d \in [0,1]$
$a \in [0,1]$

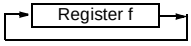
Operation: See description below

Status Affected: Z, N

Encoding:

0100	00da	ffff	ffff

Description: The contents of Register 'f' are rotated one bit to the right. The Carry Flag bit is not affected.



The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
RRNCF REG1, 0, 1   ; Rotate the value in REG1
                   ; 1 bit position right and
                   ; bit 0 loads into bit 7.
                   ; Then place the result in
                   ; the WREG Register
```

Before Instruction
```
REG1  =  1110 0110
WREG  =  x
Z, N  =  1
```
After Instruction
```
REG1  =  1110 0110
WREG  =  0111 0011
Z     =  0
N     =  0
```

```
Example 2          RRNCF INDF0, 1, 1  ; Rotate the value in the
                                      ; register pointed by the
                                      ; FSR (FSR0H:FSR0L)
                                      ; Register 1 bit
                                      ; position right and place
                                      ; the result back into
                                      ; that register.
                                      ; bit 0 loads into bit 7.
```

Case 1: Before Instruction
- FSR0 = 0x3C2
- Contents of Address (FSR0) = 0011 1010
- Z, N = x

After Instruction
- FSR0 = 0x3C2
- Contents of Address (FSR0) = 0001 1101
- Z = 0
- N = 0

Case 2: Before Instruction
- FSR0 = 0x3C2
- Contents of Address (FSR0) = 0011 1001
- Z, N = x

After Instruction
- FSR0 = 0x3C2
- Contents of Address (FSR0) = 1001 1100
- Z = 0
- N = 1

SETF Set f

Syntax: [*label*] SETF f, a

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: FFh → f

Status Affected: None

Encoding:

0110	100a	ffff	ffff

Description: The contents of the specified register are set.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write Register 'f'

Example 1

```
SETF  FLAG_REG, 1    ; Set all the bits in
                     ; Register FLAG_REG
```

Before Instruction
FLAG_REG = 0x5A
After Instruction
FLAG_REG = 0xFF

Example 2

```
SETF  INDF0, 1     ; Set all the bits in the
                   ; register pointed to by the
                   ; FSR (FSR0H:FSR0L) Register
```

Before Instruction
FSR0 = 0x4C2
Contents of Address
(FSR0) = 0xAA
After Instruction
FSR0 = 0x4C2
Contents of Address
(FSR0) = 0xFF

SLEEP Enter SLEEP mode

Syntax: [*label*] SLEEP

Operands: None

Operation: 00h → WDT,
0 → WDT prescaler count,
1 → $\overline{TO}$,
0 → $\overline{PD}$

Status Affected: $\overline{TO}$, $\overline{PD}$

Encoding:

0000	0000	0000	0011

Description: The power-down status bit, $\overline{PD}$ is cleared. Time-out status bit, $\overline{TO}$ is set. Watchdog Timer and its prescaler count are cleared.
The processor is put into SLEEP mode with the oscillator stopped.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	Go to sleep

Example:

```
SLEEP              ; Turn off the device
                   ; oscillator. This is the
                   ; lowest power mode
```

Before Instruction
$\overline{TO}$ = ?
$\overline{PD}$ = ?
After Instruction
$\overline{TO}$ = 1 †
PD = 0
† If WDT causes wake-up, this bit is cleared

Note: The SLEEP instruction does not affect the assignment of the WDT prescaler.

SUBFWB Subtract f from WREG with borrow

Syntax: [*label*] SUBFWB f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (WREG) – (f) – ($\overline{C}$) → destination

Status Affected: C, DC, Z, OV, N

Encoding:

0101	01da	ffff	ffff

Description: Subtract Register 'f' and carry flag (borrow) from W (2's complement method).

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
SUBFWB  MYREG, 1, 1 ; WREG - MYREG - borrow bit
```

Before Instruction
MYREG = 0x37
WREG = 0x10
C, DC, Z, OV, N = x
C = 0

After Instruction
MYREG = 0xA8
WREG = 0x10 ; result is negative
C = 0
DC = 0
Z = 0
OV = 0
N = 1

```
Example 2:          SUBFWB    MYREG, 1, 1  ; WREG - MYREG - borrow bit
    Case 1:  Before Instruction
                 MYREG     =  0x03
                 WREG      =  0x02
                 C         =  1
                 DC, Z, OV, N =  x
             After Instruction
                 MYREG     =  0xFF
                 WREG      =  0x02
                 C         =  0
                 DC        =  0
                 Z         =  0
                 OV        =  0
                 N         =  1         ; result is negative
    Case 2:  Before Instruction
                 MYREG        =  0x02
                 WREG         =  0x02
                 C            =  1
                 DC, Z, OV, N =  x
             After Instruction
                 MYREG     =  0x00
                 WREG      =  0x02      ; result is zero
                 C         =  1
                 DC        =  1
                 Z         =  1
                 OV        =  0
                 N         =  0
    Case 3:  Before Instruction
                 MYREG     =  0x01
                 WREG      =  0x03
                 C         =  1
                 DC, Z, OV, N =  x
             After Instruction
                 MYREG     =  0x02
                 WREG      =  0x03      ; result is positive
                 C         =  1
                 DC        =  1
                 Z         =  0
                 OV        =  0
                 N         =  0
```

SUBLW Subtract W from Literal

Syntax: [*label*] SUBLW k

Operands: $0 \le k \le 255$

Operation: k - (WREG) → WREG

Status Affected: C, DC, Z, OV, N

Encoding:

0000	1000	kkkk	kkkk

Description: The WREG Register is subtracted (2's complement method) from the eight bit literal 'k'. The result is placed in the WREG Register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to WREG Register

Example 1

```
SUBLW  OFFSET       ; Subtract the value in
                    ; WREG from the constant
                    ; OFFSET
```

Before Instruction
- WREG = 0x37
- OFFSET = 0x10
- C, DC, Z, OV, N = x

After Instruction
- WREG = 0xD9
- C = 0 ; result is negative
- DC = 0
- Z = 0
- OV = 0
- N = 1

```
Example 2:          SUBLW   0x02        ; Subtract WREG Register
                                        ; from 2h
    Case 1:  Before Instruction
                 WREG          =  0x01
                 C, DC, Z, OV, N =  x
             After Instruction
                 WREG          =  0x01
                 C             =  1     ; result is positive
                 DC            =  1
                 Z             =  0
                 OV            =  0
                 N             =  0
    Case 2:  Before Instruction
                 WREG          =  0x02
                 C, DC, Z, OV, N =  x
             After Instruction
                 WREG          =  0x00
                 C             =  1    ; result is zero
                 DC            =  1
                 Z             =  1
                 OV            =  0
                 N             =  0
    Case 3:  Before Instruction
                 WREG          =  0x03
                 C, DC, Z, OV, N =  x
             After Instruction
                 WREG          =
                 0xFF                  ; result is negative
                 C             =  0
                 DC            =  0
                 Z             =  0
                 OV            =  0
                 N             =  1
```

SUBWF Subtract W from f

Syntax: [*label*] SUBWF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f) - (WREG) → destination

Status Affected: C, DC, Z, OV, N

Encoding:

0101	11da	ffff	ffff

Description: Subtract (2's complement method) WREG Register from Register 'f'.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1:

```
SUBWF  REG1, 1, 1    ; Subtract the value in the
                     ; WREG Register from REG1,
                     ; placing the result in REG1
```

Case 1: Before Instruction

```
REG1          =  3
WREG          =  2
C, DC, Z, OV, N =  x
```

After Instruction

```
REG1          =  1
WREG          =  2
C             =  1    ; result is positive
DC            =  1
Z             =  0
OV            =  0
N             =  0
```

Case 2: Before Instruction

```
REG1            =  2
WREG            =  2
C, DC, Z, OV, N =  x
```

After Instruction

```
REG1            =  0
WREG            =  2
C               =  1   ; result is zero
DC              =  1
Z               =  1
OV              =  0
N               =  0
```

Case 3: Before Instruction

```
REG1            =  1
WREG            =  2
C, DC, Z, OV, N =  x
```

After Instruction

```
REG1        =  0xFF
WREG        =  2
C           =  0      ; result is negative
DC          =  0
Z           =  0
OV          =  0
N           =  1
```


SUBWFB Subtract W from f with Borrow

Syntax: [*label*] SUBWFB f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f) - (WREG) - $(\overline{C}) \rightarrow$ destination

Status Affected: C, DC, Z, OV, N

Encoding:

0101	10da	ffff	ffff

Description: Subtract (2's complement method) WREG Register from Register 'f' with borrow.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1:

```
SUBWF   REG1, 1, 1  ; Subtract the value in the
                    ; WREG Register from REG1,
                    ; placing the result in REG1
```

Case 1: Before Instruction

```
REG1        =  3
WREG        =  2
C           =  1
DC, Z, OV, N =  x
```

After Instruction

```
REG1        =  1
WREG        =  2
C           =  1    ; result is positive
DC          =  1
Z           =  0
OV          =  0
N           =  0
```

Case 2: Before Instruction

```
REG1          =  2
WREG          =  2
C             =  1
DC, Z, OV, N  =  x
```

After Instruction

```
REG1          =  0
WREG          =  2
C             =  1     ; result is zero
DC            =  1
Z             =  1
OV            =  0
N             =  0
```

Case 3: Before Instruction

```
REG1          =  1
WREG          =  2
C             =  1
DC, Z, OV, N  =  x
```

After Instruction

```
REG1          =  0xFF
WREG          =  2
C             =  0     ; result is negative
DC            =  0
Z             =  0
OV            =  0
N             =  1
```


SWAPF Swap Nibbles in f

Syntax: [*label*] SWAPF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (f<3:0>) → destination<7:4>,
(f<7:4>) → destination<3:0>

Status Affected: None

Encoding:

0011	10da	ffff	ffff

Description: The upper and lower nibbles of Register 'f' are exchanged.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
SWAPF  REG1, 0, 1   ; Swap the high and low
                    ; nibble of Register REG1
                    ; and place the result in
                    ; the WREG Register
```

Before Instruction

REG1 = 0xA5
WREG = x

After Instruction

REG1 = 0xA5
WREG = 0x5A

Example 2

```
SWAPF  INDF0, 1, 1  ; Swap the high and low
                    ; nibble of register pointed
                    ; to by the FSR
                    ; (FSR0H:FSR0L) Register,
                    ; placing the result back
                    ; into that register
```

Before Instruction
FSR0 = 0x5C2
Contents of Address
(FSR0) = 0x20
After Instruction
FSR0 = 0x5C2
Contents of Address
(FSR0) = 0x02

Example 3

```
SWAPF REG1, 1, 1   ; Swap the high and low
                   ; nibble of Register REG1
                   ; placing the result back
                   ; into that register
```

Before Instruction
REG1 = 0xA5
After Instruction
REG1 = 0x5A

TBLRD Table Read

Syntax: [*label*] TBLRD[*, *+, *-, or +*]

Operands: $0 \le m \le 3$

Operation:
```
if TBLRD *,
    (Prog Mem (TBLPTR)) → TABLAT;
    TBLPTR - No Change;
if TBLRD *+,
    (Prog Mem (TBLPTR)) → TABLAT;
    (TBLPTR) +1 → TBLPTR;
if TBLRD *-,
    (Prog Mem (TBLPTR)) → TABLAT;
    (TBLPTR) -1 → TBLPTR;
if TBLRD +*,
    (TBLPTR) +1 → TBLPTR;
    (Prog Mem (TBLPTR)) → TABLAT;
```

Status Affected: None

Encoding:

0000	0000	0000	10mm	* → mm = 00 *+ → mm = 01 *- → mm = 10 +* → mm = 11

Description: This instruction is used to read the contents of Program Memory. To address the program memory a pointer called Table Pointer (TBLPTR) is used.

The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2 Mbyte address range. The LSb of the TBLPTR selects which byte of the program memory location to access.

TBLPTR[0] = 0: Least Significant byte of Program Memory Word
TBLPTR[0] = 1: Most Significant byte of Program Memory Word

The Program Memory word address is the same as the TBLPTR address, except that the LSb of TBLPTR (TBLPTR[0]) is always forced to '0'.

The TBLRD instruction can modify the value of TBLPTR as follows:

- no change
- post-increment
- post-decrement
- pre-increment

Words: 1

Cycles: 2

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation

2nd cycle:

No operation	No operation (Table Pointer on Address bus)	No operation	No operation ($\overline{OE}$ goes low)

Example 31-3:Program Memory Contents for Examples

Word Address (points to low byte)	Data Word	
	MSb	LSb
0x00A356	0x12	0x34
0x00A358	0x56	0x28
0x00A35A	0xAA	0x55
0x00A35C	0xFF	0xFE
0x00A35E	0xB1	0x00

Word Address (points to low byte)	Data Word	
	MSb	LSb
0x000000	0x01	0x00
0x000002	0x03	0x02
0x000004	0x05	0x04
0x000006	0x07	0x06
0x000008	0x08	0x07

```
Example 1           TBLRD*+             ; Read byte addressed by
                                        ; TBLPTR, then increment
                                        ; TBLPTR
                Before Instruction
                    TABLAT                  =   x
                    TBLPTR                  =   0x00A356
                    Contents of Address (TBLPTR)=   0x34
                After Instruction
                    TABLAT                  =   0x34
                    TBLPTR                  =   0x00A357

Example 2           TBLRD+*             ; Increment TBLPTR, then
                                        ; Read byte addressed by
                                        ; TBLPTR
                Before Instruction
                    TABLAT                  =   x
                    TBLPTR                  =   0x00A357
                    Contents of Address (TBLPTR)=   0x12
                    Contents of Address (TBLPTR + 1)=  0x28
                After Instruction
                    TABLAT                  =   0x28
                    TBLPTR                  =   0x00A358

Example 3           TBLRD*-             ; Read byte addressed by
                                        ; TBLPTR, then decrement
                                        ; TBLPTR
                Before Instruction
                    TABLAT                  =   x
                    TBLPTR                  =   0x00A357
                    Contents of Address (TBLPTR)=   0x12
                After Instruction
                    TABLAT                  =   0x12
                    TBLPTR                  =   0x00A356

Example 4           TBLRD*              ; Read byte addressed by
                                        ; TBLPTR. TBLPTR is unchanged
                Before Instruction
                    TABLAT                  =   x
                    TBLPTR                  =   0x00A357
                    Contents of Address (TBLPTR)=   0x12
                After Instruction
                    TABLAT                  =   0x12
                    TBLPTR                  =   0x00A357
```

TBLWT Table Write

Syntax: [*label*] TBLWT[*, *+, *-, +*]

Operands: 0 ≤ m ≤ 3

Operation:
if TBLWT*,
(TABLAT) → Prog Mem (TBLPTR) or Holding Register[1];
TBLPTR - No Change;
if TBLWT*+,
(TABLAT) → Prog Mem (TBLPTR) or Holding Register[1];
(TBLPTR) +1 → TBLPTR;
if TBLWT*-,
(TABLAT) → Prog Mem (TBLPTR) or Holding Register[1];
(TBLPTR) -1 → TBLPTR;
if TBLWT+*,
(TBLPTR) +1 → TBLPTR;
(TABLAT) → Prog Mem (TBLPTR) or Holding Register[1];

Note 1: The use of a Holding Register(s) is device specific. Please refer to the Device Data Sheet for information on the operation of the TBLWT instruction with the Program Memory.

Status Affected: None

Encoding:

0000	0000	0000	11mm

```
*  → mm = 00
*+ → mm = 01
*- → mm = 10
+* → mm = 11
```

Description: This instruction is used to program the contents of Program Memory. To address the program memory a pointer called Table Pointer (TBLPTR) is used.

The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2 MByte address range. The LSb of the TBLPTR selects which byte of the program memory location to access.

TBLPTR[0] = 0: Least Significant byte of Program Memory Word
TBLPTR[0] = 1: Most Significant byte of Program Memory Word

The Program Memory word address is the same as the TBLPTR address, except that the LSb of TBLPTR (TBLPTR[0]) is always forced to '0'.

The TBLWT instruction can modify the value of TBLPTR as follows:

- no change
- post-increment
- post-decrement
- pre-increment

Words: 1

Cycles: 2 (many if long write to internal program memory)

Q Cycle Activity:

1st cycle:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation

2nd cycle:

No operation	No operation (Table Pointer on Address bus)	No operation	No operation (Table Latch on Address bus, $\overline{WR}$ goes low)

Example 31-4:Program Memory Contents for Examples

Word Address (points to low byte)	Original Data Word (before)		Example 1 Data Word (after)		Example 2 Data Word (after)		Example 3 Data Word (after)		Example 4 Data Word (after)	
	MSb	LSb	MSb	LSb	MSb	LSb	MSb	LSb	MSb	LSb
0x00A356	0x12	0x34	0x12	**0x55**	0x12	0x34	**0xAA**	0x34	0x12	0x34
0x00A358	0x56	0x28	0x56	0x28	0x56	**0xAA**	0x56	0x28	**0x5A**	0x28
0x00A35A	0xAA	0x55	0xAA	0x55	0xAA	0x55	0xAA	0x55	0xAA	0x55
TABLAT			0x55		0xAA		0xAA		0x5A	

Example 1

```
TBLWT*+          ; Write byte addressed by
                 ; TBLPTR, then increment TBLPTR
```

Before Instruction
TABLAT = 0x55
TBLPTR = 0x00A356
Contents of (TBLPTR) = 0x34
After Instruction
TBLPTR = 0x00A357
Contents of (TBLPTR) = 0x55

Example 2

```
TBLWT+*          ; Increment TBLPTR, then Write
                 ; byte addressed by TBLPTR
```

Before Instruction
TABLAT = 0xAA
TBLPTR = 0x00A357
Contents of (TBLPTR) = 0x12
Contents of (TBLPTR + 1) = 0x28
After Instruction
TBLPTR = 0x00A358
Contents of (TBLPTR) = 0x12
Contents of (TBLPTR + 1) = 0xAA

Example 3

```
TBLWT*-          ; Write byte addressed by
                 ; TBLPTR, then decrement TBLPTR
```

Before Instruction
TABLAT = 0xAA
TBLPTR = 0x00A357
Contents of (TBLPTR) = 0x12
After Instruction
TBLPTR = 0x00A356
Contents of (TBLPTR) = 0xAA

Example 4

```
TBLWT*           ; Write byte addressed by
                 ; TBLPTR. TBLPTR is unchanged
```

Before Instruction
TABLAT = 0x5A
TBLPTR = 0x00A359
Contents of (TBLPTR) = 0x56
After Instruction
TBLPTR = 0x00A359
Contents of (TBLPTR) = 0x5A

TSTFSZ Test f, Skip if 0

Syntax: [*label*] TSTFSZ f, a

Operands: $0 \le f \le 255$
$a \in [0,1]$

Operation: Skip if (f) = 0

Status Affected: None

Encoding:

0110	011a	ffff	ffff

Description: If Register 'f' = 0, the next instruction fetched is discarded and a NOP is executed (two NOPs if the fetched instruction is a two-cycle instruction).

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1 (2 or 3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

If skip (2nd cycle):

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation

If skip and followed by a two word instruction:

Q1	Q2	Q3	Q4
No operation	No operation	No operation	No operation
No operation	No operation	No operation	No operation

Example 1A

```
HERE    TSTFSZ  REG1, 1      ; If Register REG1 is zero
NZERO   •                    ; then skip the next
ZERO    •                    ; program memory address
```

Before Instruction

REG1 = 0xAF
PC = Address (HERE)

After Instruction

PC = Address (NZERO)

Example 2

```
HERE    TSTFSZ  REG1, 1      ; If Register REG1 is zero
NZERO   •                    ; then skip the next
ZERO    •                    ; program memory address
```

Before Instruction

REG1 = 0x00
PC = Address (HERE)

After Instruction

PC = Address (ZERO)

Example 3

```
HERE    TSTFSZ  REG1, 0      ; If Register REG1 is zero
NZERO   •                    ; then skip the next
ZERO    •                    ; program memory address
```

Case 1

Before Instruction

REG1 = 0xAF
Address of REG1 = 0x9A, Bank 3
0x9A, Bank 15 = 0x00
PC = Address (HERE)

After Instruction

PC = Address (ZERO)

Case 2

Before Instruction

REG1 = 0x00
Address of REG1 = 0x9A, Bank 3
0x9A, Bank 15 = 0xAF
PC = Address (HERE)

After Instruction

PC = Address (NZERO)

Example 4

```
HERE    TSTFSZ  INDF0, 1    ; If Register pointed to by
NZERO   •                   ; FSR0 (FSR0H:FSR0L) is
ZERO    •                   ; zero, then skip the next
                            ; program memory address
```

Case 1

Before Instruction

FSR0 = 0x6C2
Contents of Address (FSR0) = 0xAF
PC = Address (HERE)

After Instruction

FSR0 = 0x6C2
Contents of Address (FSR0) = 0xAF
PC = Address (NZERO)

Case 2

Before Instruction

FSR0 = 0x6C2
Contents of Address (FSR0) = 0x00
PC = Address (HERE)

After Instruction

FSR0 = 0x6C2
Contents of Address (FSR0) = 0x00
PC = Address (ZERO)

XORLW Exclusive OR Literal with W

Syntax: [*label*] XORLW k

Operands: $0 \le k \le 255$

Operation: (WREG).XOR. k → W

Status Affected: Z, N

Encoding:

0000	1010	kkkk	kkkk

Description: The contents of the WREG Register are XOR'ed with the eight bit literal 'k'. The result is placed in the WREG Register.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process data	Write to WREG Register

Example 1

```
XORLW   0xAF      ; Exclusive OR the value
                  ; in WREG with AFh
```

Before Instruction

```
WREG  =  0xB5     ; 1011 0101   (0xB5)
Z, N  =  x        ; 1010 1111   (0xAF)
```

After Instruction

```
                  ; ---------   ------
WREG  =  0x1A     ; 0001 1010   (0x1A)
Z     = 0
N     = 0
```

Example 2

```
XORLW  MYREG      ; Exclusive OR the value
                  ; in WREG with the address
                  ; of MYREG
```

Before Instruction

```
WREG     =  0xAF      ;  1010 1111
Address of MYREG †    ;
         =  0x37      ;  0011 0111
Z, N     =  x         ;  ---- ----
† MYREG is a symbol for a ;  1001 1000
data memory location
```

After Instruction

```
WREG  =  0x98
Z     =  0
N     =  1
```

Example 3

```
        XORLW   HIGH (LU_TABLE) ; Exclusive OR the value
                                ; in WREG with the high
                                ; byte of the address of
                                ; LU_TABLE
Before Instruction              ;
    WREG    =   0xAF            ;  1010 1111
    Address of LU_TABLE †       ;
            =   0x9375          ;  1001 0011
    Z, N    =   x               ;  ---- ----
    † LU_TABLE is a label for an;
    address in program memory   ;
After Instruction               ;
    WREG    =   0x3C            ;  0011 1100
    Z       =   0
    N       =   0
```

XORWF Exclusive OR W with f

Syntax: [*label*] XORWF f, d, a

Operands: $0 \le f \le 255$
$d \in [0,1]$
$a \in [0,1]$

Operation: (WREG).XOR. (f) → destination

Status Affected: Z, N

Encoding:

0001	10da	ffff	ffff

Description: Exclusive OR the contents of the WREG Register with Register 'f'.

The 'd' bit selects the destination for the operation.
If 'd' is 1; the result is stored back in the File Register 'f'.
If 'd' is 0; the result is stored in the WREG Register.

The 'a' bit selects which bank is accessed for the operation.
If 'a' is 1; the bank specified by the BSR Register is used.
If 'a' is 0; the access bank is used.

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read Register 'f'	Process data	Write to destination

Example 1

```
        XORWF  REG1, 1, 1   ; Exclusive OR the value
                            ; in WREG with the value in
                            ; REG1
```

Before Instruction

```
REG1   =  0xAF      ; 1010 1111   (0xAF)
WREG   =  0xB5      ; 1011 0101   (0xB5)
Z, N   =  x         ; ---------   ------
```

After Instruction

```
REG1   =  0x1A      ; 0001 1010   (0x1A)
WREG   =  0xB5
Z      =  0
N      =  0
```

Example 2

```
        XORWF  REG1, 0, 1   ; Exclusive OR the value
                            ; in WREG with the value in
                            ; REG1. Place result in
                            ; WREG
Before Instruction
        REG1   =  0xAF      ; 1010 1111   (0xAF)
        WREG   =  0xB5      ; 1011 0101   (0xB5)
        Z, N   =  x         ; ---------   ------
After Instruction
        REG1   =  0xAF      ;
        WREG   =  0x1A      ; 0001 1010   (0x1A)
        Z      =  0
        N      =  0
```

Example 3

```
        XORWF  INDF0, 1, 1  ; Exclusive OR the value
                            ; in WREG with the value
                            ; pointed to by the FSR0
                            ; (FSR0H:FSR0L) Register
Before Instruction          ;
        WREG    =  0xB5     ;  1011 0101
        FSR0    =  0x7C2    ;
        Contents of Address ;
          (FSR0) =  0xAF    ;  1010 1111
        Z, N    =  x        ;
After Instruction           ;
        WREG    =  0xB5     ;
        FSR0    =  0x7C2    ;
        Contents of Address ;
          (FSR0) =  0x1A    ;  0001 1010
        Z       =  0        ;
        N       =  0        ;
```

Example 4

```
        XORWF  REG1, 1, 0   ; Exclusive OR the value
                            ; in WREG with the value
                            ; at address REG1 in the
                            ; access bank. Place
                            ; result in access bank
Before Instruction          ;
        WREG    =  0xF8     ;  1111 1000
        REG1    =  0x01     ;  0000 0001
        Contents of Address ;
          (REG1) in access bank ;
                =  0xAA     ;  1010 1010
        Z, N    =  x        ;
After Instruction           ;
        WREG    =  0xF8     ;
        REG1    =  0x01     ;
        Contents of Address ;
          (REG1) in access bank ;
                =  0x52     ;  0101 0010
        Z       =  0        ;
        N       =  0        ;
```

31.8 Design Tips

Question 1: ***I have seen references to "Read-Modify-Write" instructions in your data sheet, but I do not know what that is. Can you explain what it is and why I need to know this?***

Answer 1:

An easy example of a Read-Modify-Write (or R-M-W) instruction is the bit clear instruction BCF. You might think that the processor just clears the bit, which on a port output pin would clear the pin. What actually happens is the whole port (or register) is first read, THEN the bit is cleared, then the new modified value is written back to the port (or register). Actually, any instruction that depends on a value currently in the register is going to be a Read-Modify-Write instruction. This includes ADDWF, SUBWF, BCF, BSF, INCF, XORWF, etc... Instructions that do not depend on the current register value, like MOVWF, CLRF, and so on are not R-M-W instructions.

One situation where you would want to consider the affects of a R-M-W instruction is a port that is continuously changed from input to output and back. For example, say you have TRISB set to all outputs, and write all ones to the PORTB Register, all of the PORTB pins will go high. Now, say you turn pin RB3 into an input, which happens to go low. A BCF PORTB,6 is then executed to drive pin RB6 low. If you then turn RB3 back into an output, it will now drive low, even though the last value you put there was a one. What happened was that the BCF of the other pin (RB6) caused the whole port to be read, including the zero on RB3 when it was an input. Then, bit 6 was changed as requested, but since RB3 was read as a zero, zero will also be placed back into that port latch, overwriting the one that was there before. When the pin is turned back into an output, the new value was reflected. Try using the LATx register instead of the PORTx register for this read-modify-write operation.

Question 2: ***When I perform a BCF, other pins get cleared in the port. Why?***

Answer 2:

There are a few possibilities, two are:

1. Another case where a R-M-W instruction may seem to change other pin values unexpectedly can be illustrated as follows: Suppose you make PORTC all outputs and drive the pins low. On each of the port pins is an LED connected to ground, such that a high output lights it. Across each LED is a 100 µF capacitor. Let's also suppose that the processor is running very fast, say 20 MHz. Now if you go down the port setting each pin in order; BSF PORTC,0 then BSF PORTC,1 then BSF PORTC,2 and so on, you may see that only the last pin was set, and only the last LED actually turns on. This is because the capacitors take a while to charge. As each pin was set, the pin before it was not charged yet and so was read as a zero. This zero is written back out to the port latch (R-M-W, remember) which clears the bit you just tried to set the instruction before. This is usually only a concern at high speeds and for successive port operations, but it can happen, so take it into consideration.
2. If this is on a PIC16C7X device, you may not have configured the I/O pins properly in the ADCON1 Register. If a pin is configured for analog input, any read of that pin will read a zero, regardless of the voltage on the pin. This is an exception to the normal rule that the pin state is always read. You can still configure an analog pin as an output in the TRIS Register, and drive the pin high or low by writing to it, but you will always read a zero. Therefore, if you execute a Read-Modify-Write instruction (see previous question), all analog pins are read as zero, and those not directly modified by the instruction will be written back to the port latch as zero. A pin configured as analog is expected to have values that may be neither high nor low to a digital pin, or floating. Floating inputs on digital pins are a no-no, and can lead to high current draw in the input buffer, so the input buffer is disabled.

31.9 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced family (that is they may be written for the Baseline, the Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the Instruction Set are:

Title	Application Note #

No related Application Notes at this time.

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

31.10 Revision History

Revision A

This is the initial released revision of the Enhanced MCU Instruction Set description.

Section 32. Electrical Specifications

HIGHLIGHTS

32.1 Introduction

This section is intended to present the electrical specifications that may be specified in a particular device data sheet and their meaning. This section is **NOT** intended to give the values of these specifications. For the device specific values you **must** refer to the device's data sheet. All values shown in this section should be considered as Example Values.

In the description of the device and the functional modules (previous sections), there have been references to electrical specification parameters. These references have been hyperlinked in the electronic version to aid in the use of this manual.

Note:	Before starting any design, Microchip HIGHLY recommends that you acquire the most recent copy of the device data sheet and review the electrical specifications to ensure that they will meet your requirements.

Throughout this section, certain terms will be used. Table 32-1 shows the conventions that will be used.

Table 32-1: Term Conventions

Term	Description
PIC18**C**XXX(1)	For EPROM Program Memory devices tested to standard voltage range
PIC18**LC**XXX(1)	For EPROM Program Memory devices tested to extended voltage range
PIC18**F**XXX(1)	For FLASH Program Memory devices tested to standard voltage range
PIC18**LF**XXX(1)	For FLASH Program Memory devices tested to extended voltage range
PIC18**CR**XXX(1)	For ROM Program Memory devices tested to standard voltage range
PIC18**LCR**XXX(1)	For ROM Program Memory devices tested to extended voltage range
LP osc	For devices configured with the LP device oscillator selected
XT osc	For devices configured with the XT device oscillator selected
HS osc	For devices configured with the HS device oscillator selected
HS+PLL osc	For devices configured with the HS+PLL device oscillator selected
RC osc	For devices configured with the RC device oscillator selected
RCIO osc	For devices configured with the RCIO device oscillator selected
EC osc	For devices configured with the EC device oscillator selected
ECIO osc	For devices configured with the ECIO device oscillator selected
Commercial	For devices with the commercial temperature range grading ($0°C \le T_A \le +70°C$)
Industrial	For devices with the industrial temperature range grading ($-40°C \le T_A \le +85°C$)
Extended	For devices with the extended temperature range grading ($-40°C \le T_A \le +125°C$)

Note 1: In Electrical Specification examples, we will use PIC18CXXX for the standard voltage range devices and PIC18LCXXX for the extended voltage range devices.

32.2 Absolute Maximums

The Absolute Maximum Ratings specify the worst case conditions that can be applied to the device. These ratings are not meant as operational specifications. Stresses above the listed values may cause damage to the device. Specifications are not always stand-alone, that is, the specification may have other requirements as well.

An example of this is the “maximum current sourced/sunk by any I/O pin”. The number of I/O pins that can be sinking/sourcing current, at any one time, is dependent upon the maximum current sunk/source by the port(s) (combined) and the maximum current into the VDD pin or out of the VSS pin. In this example, the physical reason is the Power and Ground bus width to the I/O ports and internal logic. If these specifications are exceeded, then electromigration may occur on these Power and Ground buses. Over time, electromigration would cause these buses to open (be disconnected from the pin) and, therefore, cause the logic attached to these buses to stop operating. So, exceeding the absolute specifications may cause device reliability issues.

Input Clamp Current is defined as the current through the diode to VSS/VDD if pin voltage exceeds specification.

Example Absolute Maximum Ratings †

Ambient temperature under bias	-55 to +125°C
Storage temperature	-65°C to +150°C
Voltage on any pin with respect to VSS (except VDD, $\overline{\text{MCLR}}$, and RA4)	-0.3V to (VDD + 0.3V)
Voltage on VDD with respect to VSS	-0.3 to +7.5V
Voltage on $\overline{\text{MCLR}}$ with respect to VSS **(2)**	0 to +13.25V
Voltage on RA4 with respect to Vss	0 to +8.5V
Total power dissipation **(1)**	1.0W
Maximum current out of VSS pin	300 mA
Maximum current into VDD pin	250 mA
Input clamp current, IIK (VI < 0 or VI > VDD)	± 20 mA
Output clamp current, IOK (VO < 0 or VO > VDD)	± 20 mA
Maximum output current sunk by any I/O pin	25 mA
Maximum output current sourced by any I/O pin	25 mA
Maximum current sunk by PORTA, PORTB, and PORTE (combined)	200 mA
Maximum current sourced by PORTA, PORTB, and PORTE (combined)	200 mA
Maximum current sunk by PORTC and PORTD (combined)	200 mA
Maximum current sourced by PORTC and PORTD (combined)	200 mA
Maximum current sunk by PORTF and PORTG (combined)	100 mA
Maximum current sourced by PORTF and PORTG (combined)	100 mA

Note 1: Power dissipation is calculated as follows:
Pdis = VDD x {IDD - ∑ IOH} + ∑ {(VDD - VOH) x IOH} + ∑(Vol x IOL)

2: Voltage spikes below VSS at the MCLR/VPP pin, inducing currents greater than 80 mA, may cause latch-up. Thus, a series resistor of 50-100Ω should be used when applying a “low” level to the MCLR/VPP pin, rather than pulling this pin directly to VSS.

† NOTICE: Stresses above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

32.3 Voltage vs Frequency Graph

Windowed devices are superset devices with all oscillator configurations tested to the specification ranges of the **C** and **LC** devices. The temperature range that the device is tested to should be considered commercial, though at a later time, they may be tested to industrial or extended temperature levels.

Figure 32-1 and Figure 32-2 show proposed voltage vs frequency graphs for the **C** and **LC** devices.

Note:	Devices that are designated Engineering Sample are tested to the current engineering test program at time of the device testing. There is no implied warranty that these devices have been tested to any or all specifications in the Device Data Sheet.

Battery applications usually require an extended voltage range. Devices marked **LC** have an extended voltage range.

The voltage vs frequency graphs show what is the maximum frequency of operation for a given voltage. Figure 32-1 is for a **C** device, while Figure 32-2 is for an **LC** device. Notice that for Figure 32-2, there is a slope from 6MHz to 40 MHz. An equation is given in the figure which will allow you to calculate the maximum frequency of operation for a given voltage.

Figure 32-1: Example PIC18CXXX Voltage Frequency Graph

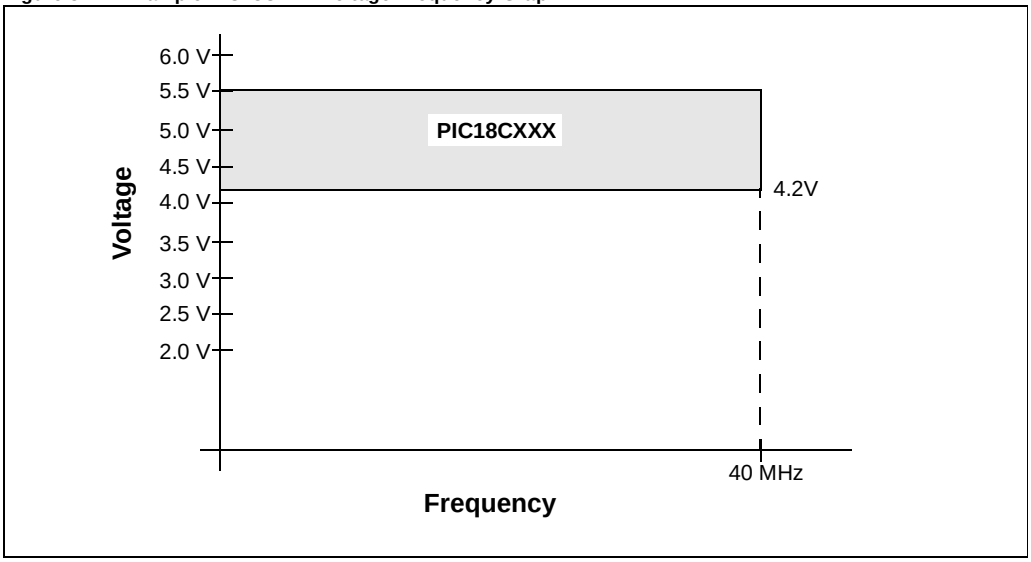
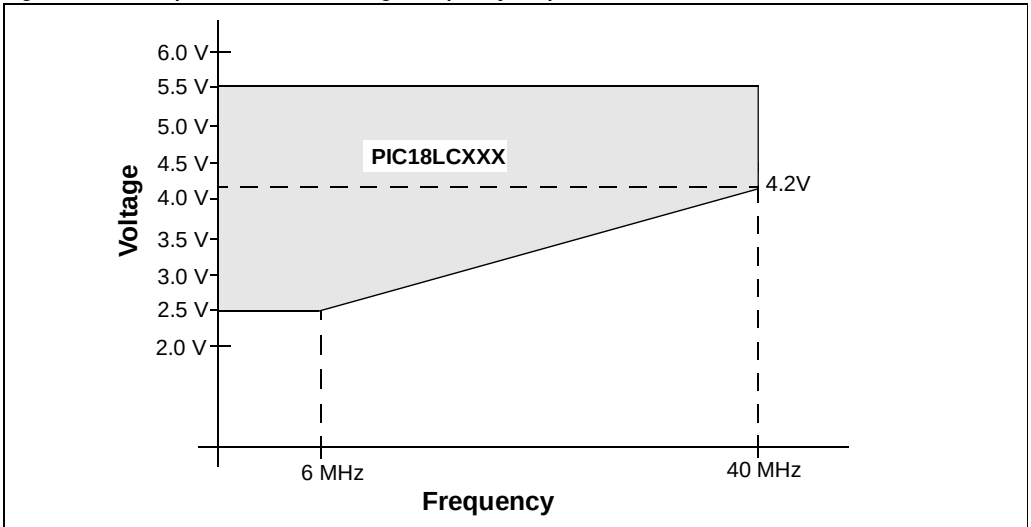


Figure 32-2: Example PIC18LCXXX Voltage Frequency Graph



FMAX = (20.0 MHz/V) (VDDAPPMIN - 2.5 V) + 6 MHz

Note: VDDAPPMIN is the minimum voltage of the PICmicro® device in the application.

32.4 Device Voltage Specifications

These specifications relate to the device VDD, power-up, and function.

Supply Voltage is the voltage level that must be applied to the device VDD pins for the proper functional operation.

RAM Data Retention Voltage is the minimum level that the device voltage may be at and still retain the RAM's data value.

VDD Start Voltage to ensure the internal Power-on Reset signal, is the level that VDD must start from, to ensure that the POR circuitry will operate properly.

VDD Rise Rate to ensure internal Power-on Reset signal, is the minimum slope that VDD must rise to cause the POR circuitry to trip.

Brown-out Reset Voltage is the voltage range where the brown-out circuitry may trip. When the BOR circuitry trips, the device will either be in Brown-out Reset, or has just come out of Brown-out Reset.

Table 32-2: Example DC Characteristics

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended				
Param No.	**Symbol**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
	VDD	**Supply Voltage**					
D001		PIC18**C**XXX	4.2	—	5.5	V	
		PIC18**LC**XXX	2.5	—	5.5	V	HS, XT, RC and LP osc mode
D002	VDR	**RAM Data Retention Voltage** (1)	1.5	—	—	V	
D003	VPOR	**VDD Start Voltage** to ensure internal Power-on Reset signal	—	—	0.7	V	See section on Power-on Reset for details
D004	SVDD	**VDD Rise Rate** to ensure internal Power-on Reset signal	0.05	—	—	V/ms	See section on Power-on Reset for details
D005	VBOR	**Brown-out Reset Voltage**					
		BORV1:BORV0 = 11	1.8	—	1.91	V	For PIC18**LC**xxx VDDMIN = 1.8V
		BORV1:BORV0 = 11	2.5	—	2.66		For PIC18**LC**xxx VDDMIN > 1.8V
		BORV1:BORV0 = 10	2.7	—	2.86		
		BORV1:BORV0 = 01	4.2	—	4.46		
		BORV1:BORV0 = 00	4.5	—	4.78		
D007	VBHYS	Brown-out Hysteresis	30	—	100	mV	

Note 1: This is the limit to which VDD can be lowered in SLEEP mode or during a device RESET without losing RAM data.

32.5 Device Current Specifications

IDD is referred to as supply current and is the current (I) consumed by the device when in operating mode. This test is taken with all I/O as inputs, either pulled high or low. That is, there are no floating inputs, nor are any pins driving an output (with a load).

IPD is referred to as power-down current and is the current (I) consumed by the device when in SLEEP mode (power-down), referred to as power-down current. These tests are taken with all I/O as inputs, either pulled high or low. That is, there are no floating inputs, nor are any pins driving an output (with a load), weak pull-ups are disabled.

A device may have certain features and modules that can operate while the device is in SLEEP mode. Some of these modules are:

- Watchdog Timer (WDT)
- Low Voltage Detect (LVD)
- Brown-out Reset (BOR) circuitry
- Timer1 Oscillator
- Analog to Digital converter
- Comparators
- Voltage Reference
- CAN Module

When all features are disabled, the device will consume the lowest possible current (the leakage current). If any of these features are operating while the device is in SLEEP, a higher current will occur. The difference in current between the lowest power mode (everything off) and only that one feature enabled (such as the WDT), is what we call the **Module Differential Current**. If more then one feature is enabled, then the expected current can easily be calculated as: the base current (everything disabled and in SLEEP mode) plus all Module Differential Currents (delta currents). Example 32-1 shows an example of calculating the typical currents for a device at 5V, with the WDT and Timer1 oscillator enabled.

Note:	Some modules (such as the Brown-out Reset and Low Voltage Detect) use a common resource (an internal reference voltage generator). This resource may consume a significant percentage of the total modules current when enabled. Since 2 modules are using this, the total current will be less then the calculation.

Example 32-1: IPD Calculations with WDT and Timer1 Oscillator Enabled (@ 5V)

Base Current	14 nA	; Device leakage current
WDT Delta Current	14 µA	; 14 µA - 14 nA = 14 µA
TMR1 Delta Current	22 µA	; 22 µA - 14 nA = 22 µA
Total SLEEP Current	36 µA	;

Table 32-3: Example DC Characteristics

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended				
Param No.	**Symbol**	**Characteristic**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
D010	IDD	**Supply Current (2,4)**	—	—	4	µA	XT, RC, RCIO osc configurations FOSC = 4 MHz, VDD = 4.2V
D010A			—	—	48	mA	LP osc configuration FOSC = 32 kHz, VDD = 4.2V
D010C			—	—	45	mA	EC, ECIO osc configurations, Fosc = 40 MHz, VDD = 5.5V
D013			—	—	50	mA	HS osc configurations Fosc = 25 MHz, VDD = 5.5V
D013			—	—	50	mA	HS4 osc configuration Fosc = 10 MHz, VDD = 5.5V
D014			— —	— —	48 TBD	µA µA	OSCB osc configuration FOSC = 32 kHz, VDD = 4.2V FOSC = 32 kHz, VDD = 4.2V, 25°C
	IPD	**Power-down Current (3)**					
D020			— —	<1 —	TBD 36	µA µA	VDD = 4.2V, -40°C to +85°C VDD = 5.5V, -40°C to +85°C
D020A			—	—	TBD	µA	VDD = 4.2V, 25°C
D021B			— —	— —	TBD 42	µA µA	VDD = 4.2V, -40°C to +125°C VDD = 5.5V, -40°C to +125°C

Legend: Shading of rows is to assist in readability of the table.

Note 1: Not applicable.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors, such as I/O pin loading and switching rate, oscillator type, internal code execution pattern and temperature, also have an impact on the current consumption.
The test conditions for all IDD measurements in active operation mode are:
OSC1 = external square wave, from rail to rail; all I/O pins set to inputs, pulled to VDD
MCLR = VDD; WDT enabled/disabled as specified.

3: The power-down current in SLEEP mode does not depend on the oscillator type. Power-down current is measured with the part in SLEEP mode, with all I/O pins set to inputs, tied to VDD or VSS, and all features that add delta current disabled (such as WDT, Timer1 Oscillator, ...).

4: For RC osc configuration, current through REXT is not included. The current through the resistor can be estimated by the formula Ir = VDD/2REXT (mA) with REXT in kOhm.

Table 32-3: Example DC Characteristics (Continued)

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended				
Param No.	**Symbol**	**Characteristic**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
		Module Differential Current					
D022	ΔIWDT	Watchdog Timer	—	—	25	µA	VDD = 5.5V, -40°C to +85°C
			—	—	TBD	µA	VDD = 5.5V, -40°C to +125°C
			—	—	TBD	µA	VDD = 4.2V, 25°C
			—	—	12	µA	VDD = 2.5V
			—	—	TBD	µA	VDD = 2.5V, +25°C
D022A	ΔIBOR	Brown-out Reset	—	—	50	µA	VDD = 5.5V, -40°C to +85°C
			—	—	TBD	µA	VDD = 5.5V, -40°C to +125°C
			—	—	TBD	µA	VDD = 4.2V, 25°C
			—	—	12	µA	VDD = 2.5V
			—	—	TBD	µA	VDD = 2.5V, +25°C
D022B	ΔILVD	Low Voltage Detect	—	—	TBD	µA	VDD = 4.2V, -40°C to +85°C
			—	—	TBD	µA	VDD = 4.2V, -40°C to +125°C
			—	—	TBD	µA	VDD = 4.2V, +25°C
			—	—	50	µA	VDD = 2.5V
			—	—	TBD	µA	VDD = 2.5V, +25°C
D022C	ΔIDDC	A/D Converter Current	—	—	TBD	µA	VDD = 5.5V, -40°C to +85°C
			—	—	TBD	µA	VDD = 5.5V, -40°C to +125°C
			—	—	TBD	µA	VDD = 4.2V, 25°C
			—	—	TBD	µA	VDD = 2.5V
			—	—	TBD	µA	VDD = 2.5V, +25°C
D025	ΔIOSCB	Timer1 Oscillator	—	—	TBD	µA	VDD = 4.2V, -40°C to +85°C
			—	—	TBD	µA	VDD = 4.2V, -40°C to +125°C
			—	—	TBD	µA	VDD = 4.2V, 25°C
			—	—	3	µA	VDD = 2.5V
			—	—	TBD	µA	VDD = 2.5V, +25°C

Legend: Shading of rows is to assist in readability of the table.

Note 1: Not applicable.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors, such as I/O pin loading and switching rate, oscillator type, internal code execution pattern and temperature, also have an impact on the current consumption.
The test conditions for all IDD measurements in active operation mode are:
OSC1 = external square wave, from rail to rail; all I/O pins set to inputs, pulled to VDD
$\overline{MCLR}$ = VDD; WDT enabled/disabled as specified.

3: The power-down current in SLEEP mode does not depend on the oscillator type. Power-down current is measured with the part in SLEEP mode, with all I/O pins set to inputs, tied to VDD or VSS, and all features that add delta current disabled (such as WDT, Timer1 Oscillator, ...).

4: For RC osc configuration, current through REXT is not included. The current through the resistor can be estimated by the formula Ir = VDD/2REXT (mA) with REXT in kOhm.

32.6 Input Threshold Levels

The **Input Low Voltage** (V_{IL}) is the maximum voltage level that will be read as a logic '0'. An input may not read a '0' at a voltage level above this. All designs should be to the specification, since device to device (and to a much lesser extent pin to pin) variations will cause this level to vary.

The **Input High Voltage** (V_{IH}) is the minimum voltage level that will be read as a logic '1'. An input may not read a '1' at a voltage level below this. All designs should be to the specification, since device to device (and to a much lesser extent pin to pin) variations will cause this level to vary.

The I/O pins with TTL levels are shown with two specifications. One is the industry standard TTL specification, which is specified for the voltage range of 4.5V to 5.5V. The other specifies operation over the entire voltage range of the device. The better of these two specifications may be used in the design (see **Note 2** in Table 32-4).

Table 32-4: Example DC Characteristics

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended Operating voltage VDD range as described in DC spec Table 32-2.			
Param No.	**Symbol**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
	VIL	**Input Low Voltage**				
		I/O ports:				
D030		with TTL buffer	VSS	0.15VDD	V	For entire VDD range (2)
D030A			—	0.8	V	4.5V ≤ VDD ≤ 5.5V (2)
D031		with Schmitt Trigger buffer	VSS	0.2VDD	V	
		RC3 and RC4	VSS	0.3VDD	V	
D032		$\overline{MCLR}$	VSS	0.2VDD	V	
D032A		OSC1 (in XT, HS and LP modes) and T1OSI	VSS	0.3VDD	V	
D033		OSC1 (in RC mode) (1)	VSS	0.2VDD	V	
	VIH	**Input High Voltage**				
		I/O ports:				
D040		with TTL buffer	0.25VDD + 0.8V	VDD	V	For entire VDD range (2)
D040A			2.0	VDD	V	4.5V ≤ VDD ≤ 5.5V (2)
D041		with Schmitt Trigger buffer	0.8VDD	VDD	V	
		RC3 and RC4	0.7VDD	VDD	V	
D042		$\overline{MCLR}$	0.8VDD	VDD	V	
D042A		OSC1 (in XT, HS and LP modes) and T1OSI	0.7VDD	VDD	V	
D043		OSC1 (RC mode) (1)	0.9VDD	VDD	V	
D050	VHYS	Hysteresis of Schmitt Trigger Inputs	TBD	TBD	V	

Note 1: In RC oscillator configuration, the OSC1/CLKIN pin is a Schmitt Trigger input. It is not recommended that the PICmicro be driven with an external clock while in RC mode.

2: The better of the two specifications may be used. For VIL, this would be the higher voltage and for VIH, this would be the lower voltage.

32.7 I/O Current Specifications

The PORT **Weak Pull-up Current** is the additional current consumed when the weak pull-ups are enabled.

Leakage Currents are the currents that the device consumes, since the devices are manufactured in the real world and do not adhere to their ideal characteristics. Ideally, there should be no current on an input, but due to the real world, there is always some parasitic path that consumes negligible current.

Table 32-5: Example DC Characteristics

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial and -40°C ≤ TA ≤ +125°C for extended Operating voltage VDD range as described in DC spec Table 32-2.			
Param No.	**Symbol**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
	IIL	**Input Leakage Current (2,3)**				
D060		I/O ports	—	±1	µA	VSS ≤ VPIN ≤ VDD, Pin at hi-impedance
D061		$\overline{\text{MCLR}}$	—	±5	µA	VSS ≤ VPIN ≤ VDD
D063		OSC1	—	±5	µA	VSS ≤ VPIN ≤ VDD
		Weak Pull-up Current				
D070	IPURB	PORTB weak pull-up current	50	400	µA	VDD = 5V, VPIN = VSS

Note 1: Not applicable.

2: The leakage current on the MCLR pin is strongly dependent on the applied voltage level. The specified levels represent normal operating conditions. Higher leakage current may be measured at different input voltages.

3: Negative current is defined as current sourced by the pin.

32.8 Output Drive Levels

The **Output Low Voltage** (V_{OL}) is the pin's output voltage for a low level. The V_{OL} of an I/O pin is dependent on the current sunk by that pin. If an I/O pin is shorted to V_{DD}, no matter the drive capability of the I/O pin, a low level would not be reached (and the device would consume excessive drive current). The V_{OL} is the output voltage that the I/O pin will drive, given the I/O does not need to sink more then the I_{OL} current (at the specified device voltage), as specified in the conditions portion of the specification.

The **Output High Voltage** (V_{OH}) is the pin's output voltage for a high level. The V_{OH} of an I/O pin is dependent on the current sourced by that pin. If an I/O pin is shorted to V_{SS}, no matter the drive capability of the I/O pin, a high level would not be reached (and the device would consume excessive drive current). The V_{OH} is the output voltage that the I/O pin will drive, given the I/O does not need to source more then the I_{OH} current (at the specified device voltage), as specified in the conditions portion of the specification.

Table 32-6: Example DC Characteristics

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial and -40°C ≤ TA ≤ +125°C for extended Operating voltage VDD range as described in DC spec Table 32-2.			
Param No.	**Symbol**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
	V_{OL}	**Output Low Voltage**				
D080		I/O ports	—	0.6	V	I_{OL} = 8.5 mA, V_{DD} = 4.5V, -40°C to +85°C
D080A			—	0.6	V	I_{OL} = 7.0 mA, V_{DD} = 4.5V, -40°C to +125°C
D083		OSC2/CLKOUT (RC mode)	—	0.6	V	I_{OL} = 1.6 mA, V_{DD} = 4.5V, -40°C to +85°C
D083A			—	0.6	V	I_{OL} = 1.2 mA, V_{DD} = 4.5V, -40°C to +125°C
	V_{OH}	**Output High Voltage** (1)				
D090		I/O ports	V_{DD} - 0.7	—	V	I_{OH} = -3.0 mA, V_{DD} = 4.5V, -40°C to +85°C
D090A			V_{DD} - 0.7	—	V	I_{OH} = -2.5 mA, V_{DD} = 4.5V, -40°C to +125°C
D092		OSC2/CLKOUT (RC mode)	V_{DD} - 0.7	—	V	I_{OH} = -1.3 mA, V_{DD} = 4.5V, -40°C to +85°C
D092A			V_{DD} - 0.7	—	V	I_{OH} = -1.0 mA, V_{DD} = 4.5V, -40°C to +125°C
D150	V_{OD}	**Open-drain High Voltage**	—	7.5	V	RA4 pin

Note 1: Negative current is defined as current sourced by the pin.

32.9 I/O Capacitive Loading

These loadings affect the specifications for the timing specifications. If the loading in your application is different, then you will need to determine how this will affect the characteristics of the device in your system. Capacitances less then these specifications should not have an effect on a system.

Table 32-7: Example DC Characteristics

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial and -40°C ≤ TA ≤ +125°C for extended Operating voltage VDD range as described in DC spec Table 32-2.				
Param No.	**Symbol**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
		Capacitive Loading Specs on Output Pins					
D101	CIO	All I/O pins and OSC2 (in RC mode)	—	—	50	pF	To meet the Timing Specifications of the Device
D102	CB	SCL, SDA	—	—	400	pF	In I^2C mode

32.10 Low Voltage Detect (LVD)

Low Voltage Detect is internal circuitry which will set a flag when the device voltage crosses the specified trip point.

Figure 32-3: Low-Voltage Detect Characteristics

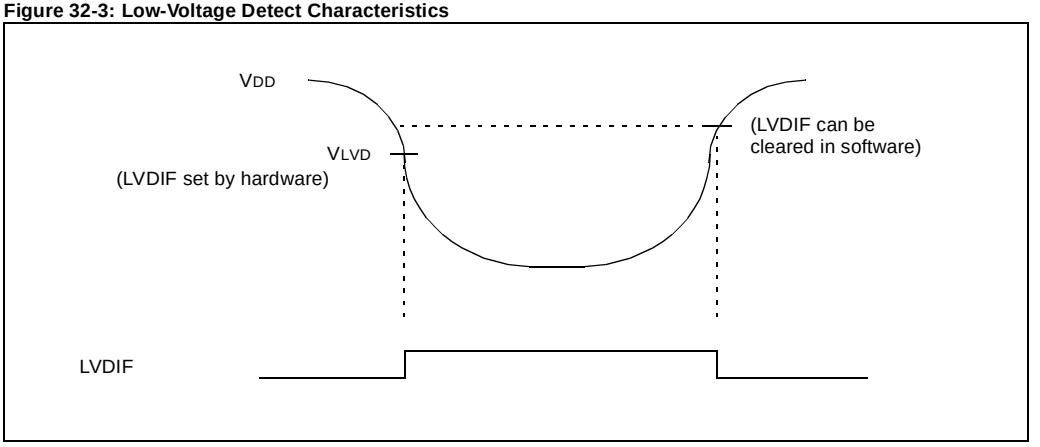


Table 32-8: Example Low Voltage Detect Requirements

			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial and -40°C ≤ TA ≤ +125°C for extended Operating voltage VDD range as described in DC spec Table 32-2.				
Param No.	**Symbol**	**Characteristic**		**Min**	**Max**	**Units**	**Conditions**
D420	VLVD	LVD Voltage	LVDL3:LVDL0 = 0100	2.5	2.66	V	
			LVDL3:LVDL0 = 0101	2.7	2.86	V	
			LVDL3:LVDL0 = 0110	2.8	2.98	V	
			LVDL3:LVDL0 = 0111	3.0	3.2	V	
			LVDL3:LVDL0 = 1000	3.3	3.52	V	
			LVDL3:LVDL0 = 1001	3.5	3.72	V	
			LVDL3:LVDL0 = 1010	3.6	3.84	V	
			LVDL3:LVDL0 = 1011	3.8	4.04	V	
			LVDL3:LVDL0 = 1100	4.0	4.26	V	
			LVDL3:LVDL0 = 1101	4.2	4.46	V	
			LVDL3:LVDL0 = 1110	4.5	4.78	V	

32.11 EPROM/FLASH/Data EEPROM

Table 32-9 shows the specifications for programming of the internal EPROM program memory.
Table 32-10 shows the specifications of the FLASH program memory and Data EEPROM.

Table 32-9: Example Program Memory Programming Requirements

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +40°C Operating voltage VDD range as described in DC spec Table 32-2.			
Param. No.	**Sym**	**Characteristic**	**Min**	**Max**	**Units**	**Conditions**
		Internal Program Memory Programming Specs (1)				
D110	VPP	Voltage on MCLR/VPP pin	12.75	13.25	V	(Note 2)
D111	VDDP	Supply voltage during programming	4.75	5.25	V	
D112	IPP	Current into MCLR/VPP pin	—	50	mA	
D113	IDDP	Supply current during programming	—	30	mA	
D114	TPROG	Programming pulse width	100	1000	µs	Terminated via internal/external interrupt or a RESET
D115	TERASE	EPROM erase time				
		Device operation ≤ 3V	4	—	hrs	See Table 32-3
		Device operation ≥ 3V	TBD	—	hrs	See Table 32-3

Note 1: These specifications are for the programming of the on-chip program memory EEPROM through the use of the table write instructions. The complete programming specifications can be found in: PIC18CXXX Programming Specifications (Literature number DS39028).

2: The MCLR/VPP pin may be kept in this range at times other than programming, but is not recommended.

Table 32-10: Example Data EEPROM/Flash Characteristics

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature 0°C ≤ TA ≤ +70°C for commercial, -40°C ≤ TA ≤ +85°C for industrial and -40°C ≤ TA ≤ +125°C for extended Operating voltage VDD range as described in DC spec Table 32-2.				
Param No.	**Symbol**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
		Data EEPROM Memory					
D120	ED	Endurance	1M	10M	—	E/W	25°C at 5V
D121	VDRW	VDD for read/write	VMIN	—	VMAX	V	VMIN = Minimum operating voltage VMAX = Maximum operating voltage
D122	TDEW	Erase/Write cycle time	—	—	10	ms	
		Program Flash Memory					
D130	EP	Endurance	100	1000	—	E/W	
D131	VPR	VDD for read	VMIN	—	VMAX	V	VMIN = Minimum operating voltage VMAX = Maximum operating voltage
D132	VPEW	VDD for erase/write	4.5	—	5.5	V	
D133	TPEW	Erase/Write cycle time	—	—	10	ms	

† Data in "Typ" column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Legend: E/W means Erase/Write cycles.

32.12 Comparators and Voltage Reference

Table 32-11: Example Comparator Characteristics

DC CHARACTERISTICS				Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial and -40°C ≤ TA ≤ +125°C for extended Operating voltage VDD range as described in DC spec Table 32-2.			
Param No.	**Symbol**	**Characteristics**		**Min**	**Max**	**Units**	**Comments**
D300	VIOFF	Input offset voltage		—	± 10	mV	
D301	VICM	Input common mode voltage		0	VDD - 1.5	V	
D302	CMRR	Common Mode Rejection Ratio		35	—	db	
300	TRESP	Response Time (1)	PIC18**C**XXX	—	400	ns	
300A			PIC18**LC**XXX	—	600	ns	
301	TMC2OV	Comparator Mode Change to Output Valid		—	10	µs	

Note 1: Response time measured with one comparator input at (VDD - 1.5)/2, while the other input transitions from VSS to VDD.

Table 32-12: Example Voltage Reference Characteristics

DC CHARACTERISTICS			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial and -40°C ≤ TA ≤ +125°C for extended Operating voltage VDD range as described in DC spec Table 32-2.				
Param No.	**Symbol**	**Characteristics**	**Min**	**Typ**	**Max**	**Units**	**Comments**
D310	VRES	Resolution	VDD/24	—	VDD/32	V	
D311	VRAA	Absolute Accuracy	— —	— —	1/4 1/2	LSb LSb	Low Range (VRR = 1) High Range (VRR = 0)
D312	VRUR	Unit Resistor Value (R)	—	2k	—	Ω	
310	TSET	Settling Time (1)	—	—	10	µs	

Note 1: Settling time measured while VRR = 1 and VR3:VR0 transitions from `0000` to `1111`.

Table 32-13: Example Fixed Voltage Reference Characteristics

			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial and -40°C ≤ TA ≤ +125°C for extended Operating voltage VDD range as described in DC spec Table 32-2.				
Param No.	**Symbol**	**Characteristic**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
D400	VRL	Output Voltage	2.0	2.048	2.1	V	VDD ≥ 2.5V
	VRH		4.0	4.096	4.2	V	VDD ≥ 4.5V
D402	TCVOUT	Ouput Voltage Drift	—	15	50	ppm/°C	(Note 1)
D403	En	Output Noise Voltage	—	TBD	—	µVp-p	0.1 Hz to 10 Hz
			—	TBD	—		10 Hz to 10 kHz
D404	IVREFSO	External Load Source	—	—	5	mA	
D405	IVREFSI	External Load Sink	—	—	-5	mA	
D406		Load Regulation	—	—	TBD	mV/mA	Isource = 0 mA to 5 mA
			—	—	TBD		Isink = 0 mA to 5 mA
D407		Line Regulation	—	—	50	µV/V	
D401A		VRL Quiescent Supply Current	—	30	50	µA	VRH, BOR, and LVD disabled. No load on VRL.
D401B		VRH Quiescent Supply Current	—	30	50	µA	VRL, BOR, and LVD disabled. No load on VRH.

32.13 Timing Parameter Symbology

The timing parameter symbols have been created with one of the following formats:

1. TppS2ppS
2. TppS
3. TCC:ST (I^2C specifications only)
4. Ts (I^2C specifications only)

T			
F	Frequency	T	Time

Lowercase letters (pp) and their meanings:

pp			
cc	CCP1	osc	OSC1
ck	CLKOUT	rd	$\overline{RD}$
cs	$\overline{CS}$	rw	$\overline{RD}$ or $\overline{WR}$
di	SDI	sc	SCK
do	SDO	ss	$\overline{SS}$
dt	Data in	t0	T0CKI
io	I/O port	t1	T1CKI
mc	$\overline{MCLR}$	wr	$\overline{WR}$

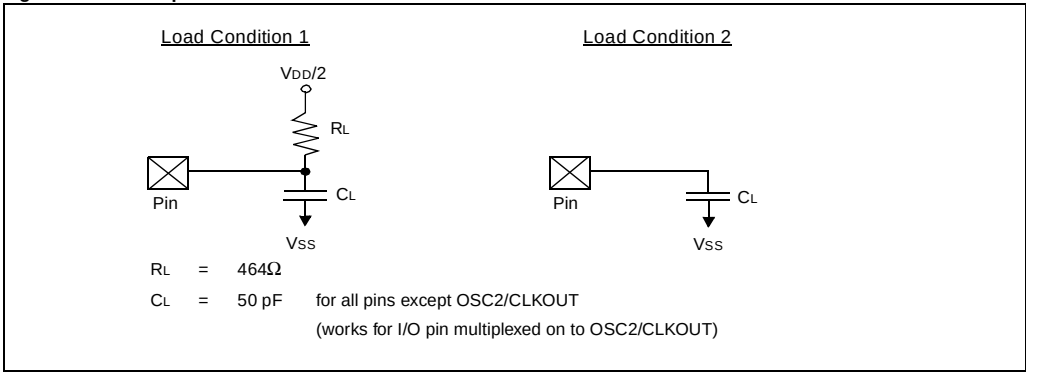
Uppercase letters and their meanings:

S			
F	Fall	P	Period
H	High	R	Rise
I	Invalid (Hi-impedance)	V	Valid
L	Low	Z	Hi-impedance
I^2C only			
AA	Output Access	High	High
BUF	Bus free	Low	Low

TCC:ST (I^2C specifications only)

CC			
HD	Hold	SU	Setup
ST			
DAT	DATA input hold	STO	STOP condition
STA	START condition		

Figure 32-4: Example Load Conditions



32.14 Example External Clock Timing Waveforms and Requirements

Figure 32-5: Example External Clock Timing Waveforms

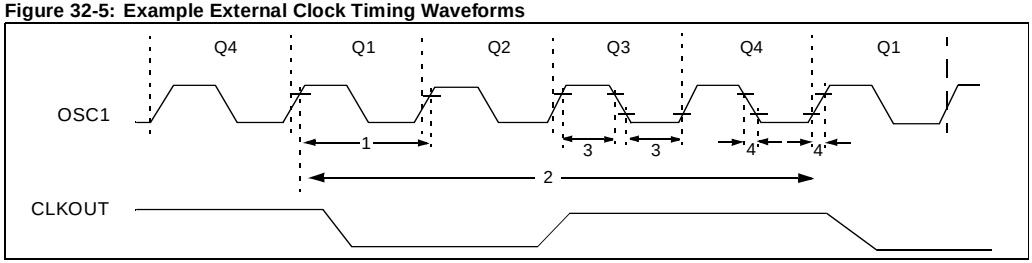


Table 32-14: Example External Clock Timing Requirements

Param. No.	Symbol	Characteristic	Min	Max	Units	Conditions
1A	Fosc	External CLKIN Frequency (1)	DC	4	MHz	XT osc
			DC	40	MHz	HS osc
			4	10	MHz	HS4 osc
			DC	40	kHz	LP osc
			DC	40	MHz	EC
		Oscillator Frequency (1)	DC	4	MHz	RC osc
			0.1	4	MHz	XT osc
			4	25	MHz	HS osc
			4	10	MHz	HS4 osc
			5	200	kHz	LP osc mode
1	Tosc	External CLKIN Period (1)	250	—	ns	XT and RC osc
			40	—	ns	HS osc
			100	—	ns	HS4 osc
			5	—	µs	LP osc
			5	—	ns	EC
		Oscillator Period (1)	250	—	ns	RC osc
			250	10,000	ns	XT osc
			100	10,000	ns	HS osc
			40	100	ns	HS4 osc
			5	—	µs	LP osc
2	TCY	Instruction Cycle Time (1)	100	—	ns	TCY = 4/FOSC
3	TosL, TosH	External Clock in (OSC1) High or Low Time				
			30	—	ns	XT osc
			2.5	—	µs	LP osc
			10	—	ns	HS osc
4	TosR, TosF	External Clock in (OSC1) Rise or Fall Time	—	20	ns	XT osc
			—	50	ns	LP osc
			—	7.5	ns	HS osc

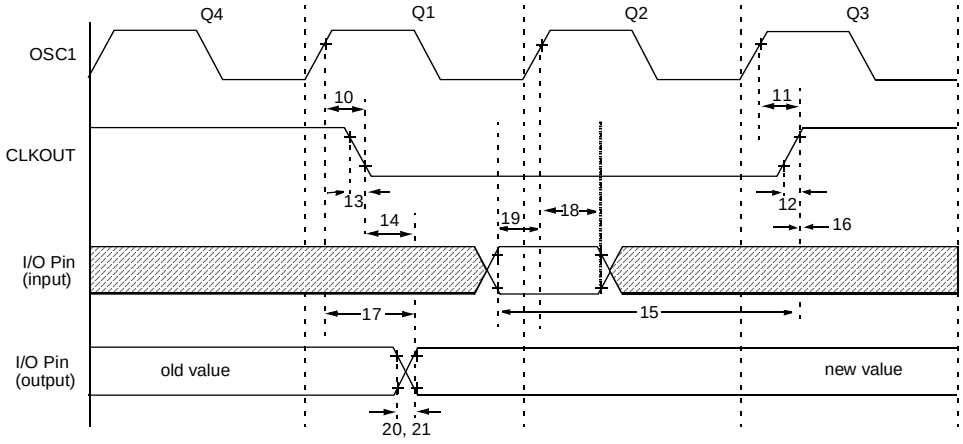
Note 1: Instruction cycle period (TCY) equals four times the input oscillator time-base period. All specified values are based on characterization data for that particular oscillator type under standard operating conditions with the device executing code. Exceeding these specified limits may result in an unstable oscillator operation and/or higher than expected current consumption. All devices are tested to operate at "min." values with an external clock applied to the OSC1/CLKIN pin.
When an external clock input is used, the "Max." cycle time limit is "DC" (no clock) for all devices.

32.15 Example Phase Lock Loop (PLL) Timing Waveforms and Requirements

Table 32-15: Example PLL Clock Timing Specification (V_{DD} = 4.2V - 5.5V)

Param No.	Symbol	Characteristic	Min	Max	Units	Conditions
7	T_{PLL}	PLL Start-up Time (Lock Time)	—	2	ms	
	Δ_{CLK}	CLKOUT Stability (Jitter) using PLL	-2	+2	%	

Figure 32-6: Example CLKOUT and I/O Timing Waveforms



Note: Refer to Figure 32-4 for load conditions.

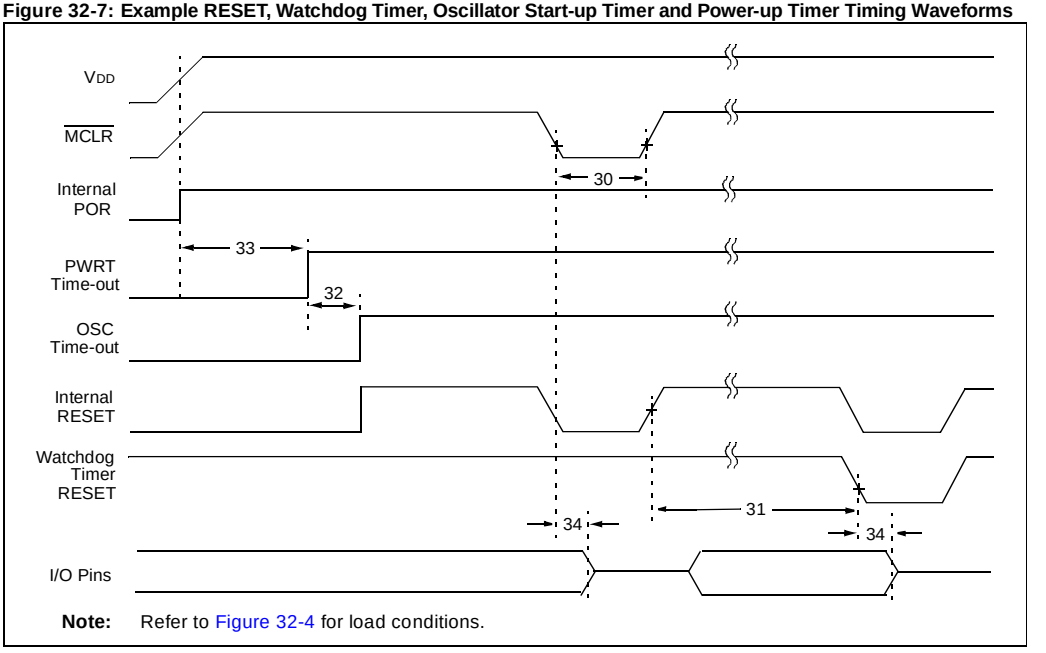
Table 32-16: Example CLKOUT and I/O Timing Requirements

Param. No.	Symbol	Characteristic		Min	Typ	Max	Units	Conditions
10	TosH2ckL	OSC1↑ to CLKOUT↓		—	75	200	ns	(Note 1)
11	TosH2ckH	OSC1↑ to CLKOUT↑		—	75	200	ns	(Note 1)
12	TckR	CLKOUT rise time		—	35	100	ns	(Note 1)
13	TckF	CLKOUT fall time		—	35	100	ns	(Note 1)
14	TckL2ioV	CLKOUT ↓ to Port out valid		—	—	0.5TCY + 20	ns	(Note 1)
15	TioV2ckH	Port in valid before CLKOUT ↑		0.25TCY + 25	—	—	ns	(Note 1)
16	TckH2ioI	Port in hold after CLKOUT ↑		0	—	—	ns	(Note 1)
17	TosH2ioV	OSC1↑ (Q1 cycle) to Port out valid		—	50	150	ns	
18	TosH2ioI	OSC1↑ (Q2 cycle) to Port input invalid (I/O in hold time)	PIC18**C**XXX	100	—	—	ns	
18A			PIC18**LC**XXX	200	—	—	ns	
19	TioV2osH	Port input valid to OSC1↑ (I/O in setup time)		0	—	—	ns	
20	TioR	Port output rise time	PIC18**C**XXX	—	10	25	ns	
20A			PIC18**LC**XXX	—	—	60	ns	
21	TioF	Port output fall time	PIC18**C**XXX	—	10	25	ns	
21A			PIC18**LC**XXX	—	—	60	ns	
22††	Tinp	INT pin high or low time		TCY	—	—	ns	
23††	Trbp	RB<7:4> change INT high or low time		TCY	—	—	ns	
24††	Trcp	RC<7:4> change INT high or low time		20			ns	

†† These parameters are asynchronous events not related to any internal clock edges.
Note 1: Measurements are taken in RC Mode where CLKOUT output is 4 x TOSC.

32.16 Example Power-up and RESET Timing Waveforms and Requirements

Figure 32-7: Example RESET, Watchdog Timer, Oscillator Start-up Timer and Power-up Timer Timing Waveforms



Note: Refer to Figure 32-4 for load conditions.

Figure 32-8: Brown-out Reset Timing

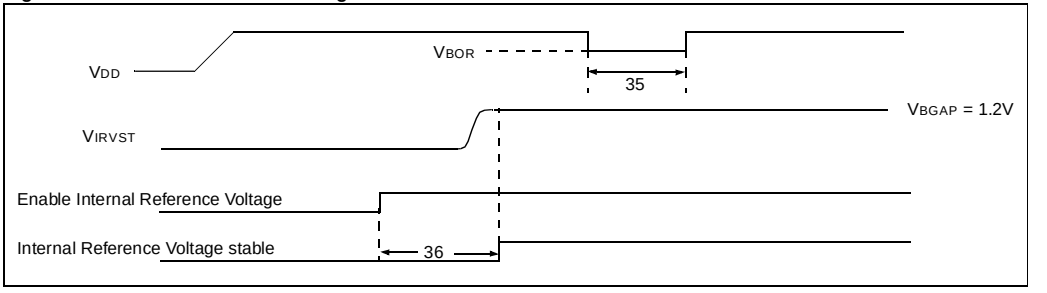
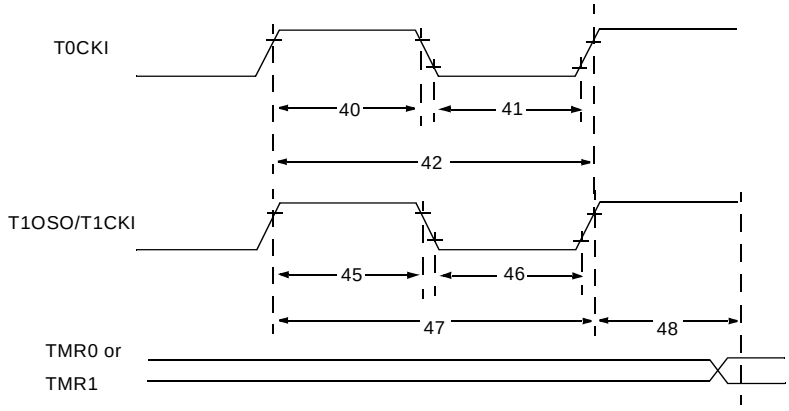


Table 32-17: Example RESET, Watchdog Timer, Oscillator Start-up Timer, Brown-out Reset, and Power-up Timer Requirements

Param. No.	Symbol	Characteristic	Min	Typ	Max	Units	Conditions
30	TmcL	$\overline{\text{MCLR}}$ Pulse Width (low)	2	—	—	µs	
31	TWDT	Watchdog Timer Time-out Period (No Prescaler)	7	18	33	ms	
32	TOST	Oscillation Start-up Timer Period	1024TOSC	—	1024TOSC	ns	TOSC = OSC1 period
33	TPWRT	Power up Timer Period	28	72	132	ms	
34	TIOZ	I/O Hi-impedance from $\overline{\text{MCLR}}$ Low or Watchdog Timer Reset	—	2	—	µs	
35	TBOR	Brown-out Reset Pulse Width	200	—	—	µs	VDD ≤ VBOR (See D005)
36	TIVRST	Time for Internal Reference Voltage to become stable	—	20	50	µs	

32.17 Example Timer0 and Timer1 Timing Waveforms and Requirements

Figure 32-9: Example Timer0 and Timer1 External Clock Timings Waveforms



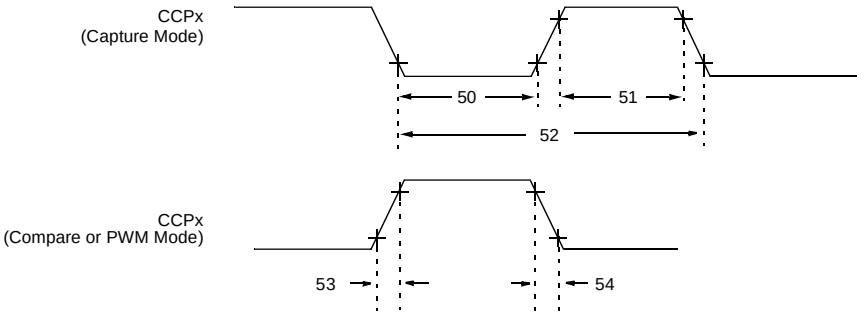
Note: Refer to Figure 32-4 for load conditions.

Table 32-18: Example Timer0 and Timer1 External Clock Requirements

Param No.	Symbol	Characteristic			Min	Max	Units	Conditions
40	Tt0H	T0CKI High Pulse Width		No Prescaler	0.5TCY + 20	—	ns	
				With Prescaler	10	—	ns	
41	Tt0L	T0CKI Low Pulse Width		No Prescaler	0.5TCY + 20	—	ns	
				With Prescaler	10	—	ns	
42	Tt0P	T0CKI Period		No Prescaler	TCY + 10	—	ns	
				With Prescaler	Greater of: 20 ns or TCY + 40 / N	—	ns	N = prescale value (1, 2, 4,..., 256)
45	Tt1H	T1CKI High Time	Synchronous, no prescaler		0.5TCY + 20	—	ns	
			Synchronous, with prescaler	PIC18**C**XXX	10	—	ns	
				PIC18**LC**XXX	25	—	ns	
			Asynchronous	PIC18**C**XXX	30	—	ns	
				PIC18**LC**XXX	50	—	ns	
46	Tt1L	T1CKI Low Time	Synchronous, no prescaler		0.5TCY + 5	—	ns	
			Synchronous, with prescaler	PIC18**C**XXX	10	—	ns	
				PIC18**LC**XXX	25	—	ns	
			Asynchronous	PIC18**C**XXX	30	—	ns	
				PIC18**LC**XXX	TBD	TBD		
47	Tt1P	T1CKI Input Period	Synchronous		Greater of: 20 ns or TCY + 40 / N	—	ns	N = prescale value (1, 2, 4, 8)
			Asynchronous		60	—	ns	
	Ft1	T1CKI oscillator input frequency range			DC	50	kHz	
48	Tcke2tmrl	Delay from external T1CKI clock edge to timer increment			2Tosc	7Tosc	ns	

32.18 Example CCP Timing Waveforms and Requirements

Figure 32-10: Example Capture/Compare/PWM Timings Waveforms



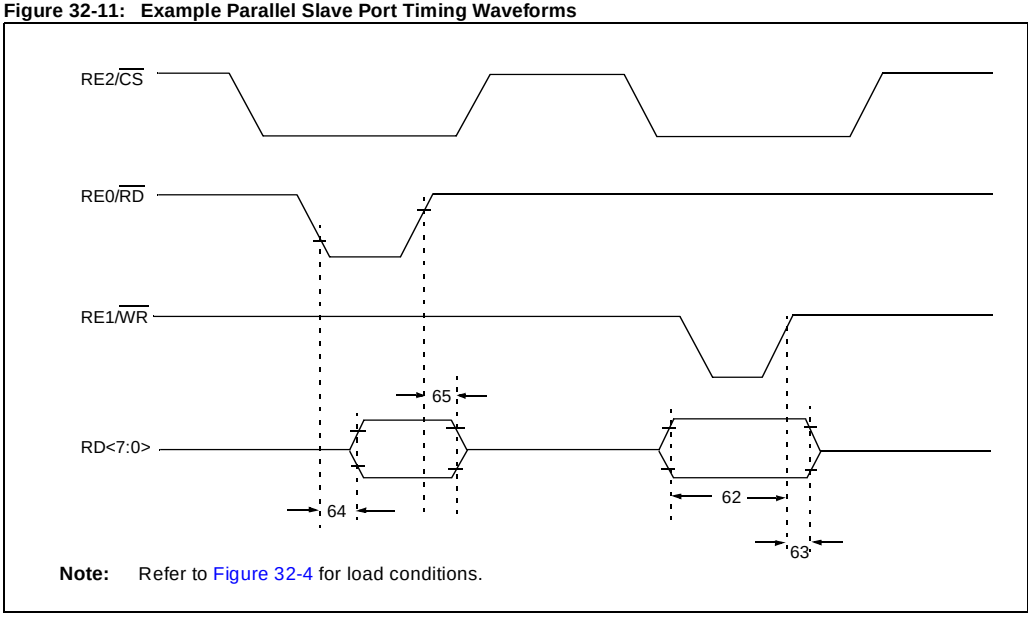
Note: Refer to Figure 32-4 for load conditions.

Table 32-19: Example Capture/Compare/PWM Requirements

Param. No.	Symbol	Characteristic			Min	Max	Units	Conditions
50	TccL	CCPx input low time	No Prescaler		0.5TCY + 20	—	ns	
			With Prescaler	PIC18**C**XXX	10	—	ns	
				PIC18**LC**XXX	20	—	ns	
51	TccH	CCPx input high time	No Prescaler		0.5TCY + 20	—	ns	
			With Prescaler	PIC18**C**XXX	10	—	ns	
				PIC18**LC**XXX	20	—	ns	
52	TccP	CCPx input period			3TCY + 40 / N	—	ns	N = prescale value (1, 4 or 16)
53	TccR	CCPx output rise time		PIC18**C**XXX	—	25	ns	
				PIC18**LC**XXX	—	45	ns	
54	TccF	CCPx output fall time		PIC18**C**XXX	—	25	ns	
				PIC18**LC**XXX	—	45	ns	

32.19 Example Parallel Slave Port (PSP) Timing Waveforms and Requirements

Figure 32-11: Example Parallel Slave Port Timing Waveforms



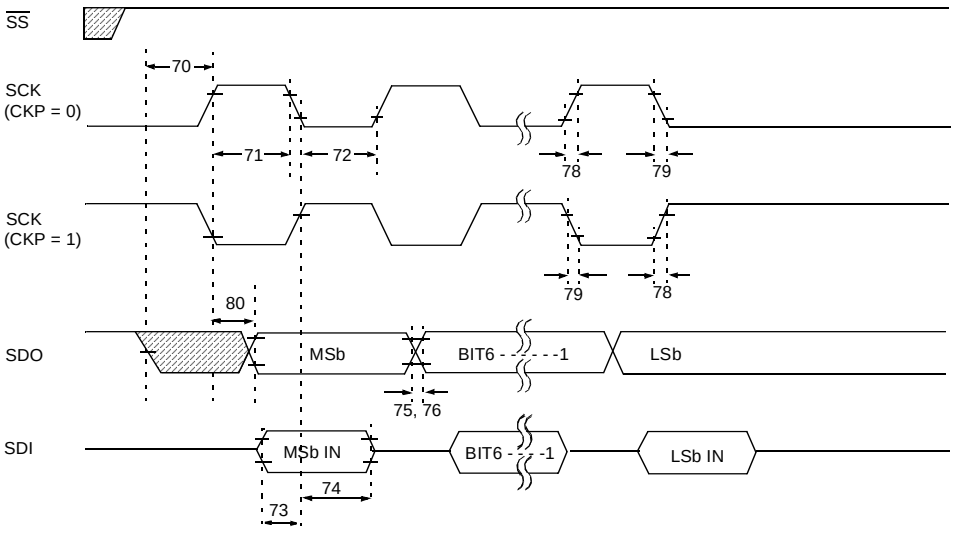
Note: Refer to Figure 32-4 for load conditions.

Table 32-20: Example Parallel Slave Port Requirements

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
62	TdtV2wrH	Data-in valid before $\overline{WR}\uparrow$ or $\overline{CS}\uparrow$ (setup time)		20 25	— —	ns ns	 Extended Temp range
63	TwrH2dtI	$\overline{WR}\uparrow$ or $\overline{CS}\uparrow$ to data-in invalid (hold time)	PIC18**C**XXX	20	—	ns	
			PIC18**LC**XXX	35	—	ns	
64	TrdL2dtV	$\overline{RD}\downarrow$ and $\overline{CS}\downarrow$ to data-out valid		— —	80 90	ns ns	 Extended Temp range
65	TrdH2dtI	$\overline{RD}\uparrow$ or $\overline{CS}\downarrow$ to data-out invalid		10	30	ns	
66	TibfINH	Inhibit of the IBF flag bit being cleared from $\overline{WR}\uparrow$ or $\overline{CS}\uparrow$		—	3TCY	ns	

32.20 Example SSP and Master SSP SPI Mode Timing Waveforms and Requirements

Figure 32-12: Example SPI Master Mode Timing (CKE = 0)



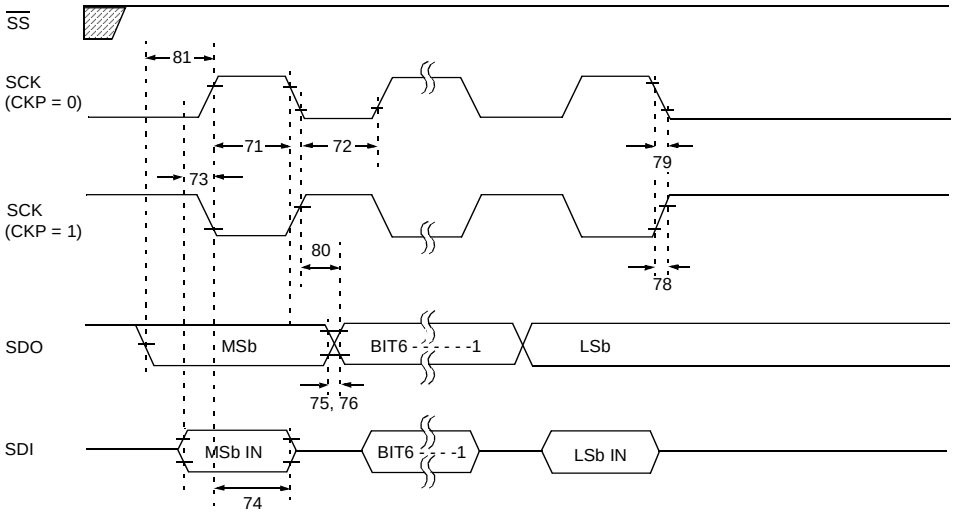
Note: Refer to Figure 32-4 for load conditions.

Table 32-21: Example SPI Mode Requirements (Master Mode, CKE = 0)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
70	TssL2scH, TssL2scL	$\overline{SS}\downarrow$ to SCK$\downarrow$ or SCK$\uparrow$ input		TCY	—	ns	
71	TscH	SCK input high time (slave mode)	Continuous	1.25TCY + 30	—	ns	
71A			Single Byte	40	—	ns	(Note 1)
72	TscL	SCK input low time (slave mode)	Continuous	1.25TCY + 30	—	ns	
72A			Single Byte	40	—	ns	(Note 1)
73	TdiV2scH, TdiV2scL	Setup time of SDI data input to SCK edge		100	—	ns	
73A	TB2B	Last clock edge of Byte1 to the 1st clock edge of Byte2		1.5TCY + 40	—	ns	(Note 2)
74	TscH2diL, TscL2diL	Hold time of SDI data input to SCK edge		100	—	ns	
75	TdoR	SDO data output rise time	PIC18**C**XXX	—	25	ns	
			PIC18**LC**XXX	—	45	ns	
76	TdoF	SDO data output fall time		—	25	ns	
78	TscR	SCK output rise time (master mode)	PIC18**C**XXX	—	25	ns	
			PIC18**LC**XXX	—	45	ns	
79	TscF	SCK output fall time (master mode)		—	25	ns	
80	TscH2doV, TscL2doV	SDO data output valid after SCK edge	PIC18**C**XXX	—	50	ns	
			PIC18**LC**XXX	—	100	ns	

Note 1: Requires the use of Parameter # 73A.
2: Only if Parameter #s 71A and 72A are used.

Figure 32-13: Example SPI Master Mode Timing (CKE = 1)



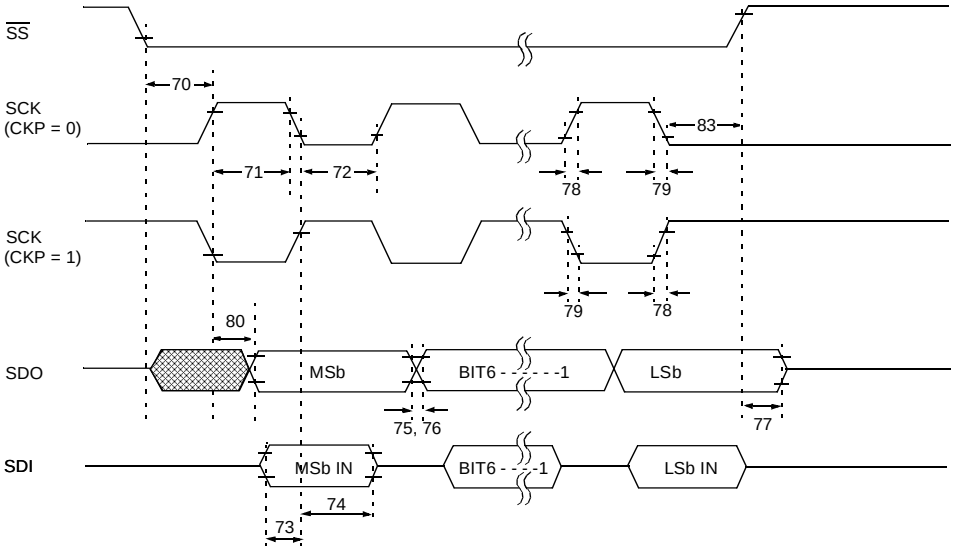
Note: Refer to Figure 32-4 for load conditions.

Table 32-22: Example SPI Mode Requirements (Master Mode, CKE = 1)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
71	TscH	SCK input high time (slave mode)	Continuous	1.25TCY + 30	—	ns	
71A			Single Byte	40	—	ns	(Note 1)
72	TscL	SCK input low time (slave mode)	Continuous	1.25TCY + 30	—	ns	
72A			Single Byte	40	—	ns	(Note 1)
73	TdiV2scH, TdiV2scL	Setup time of SDI data input to SCK edge		100	—	ns	
73A	TB2B	Last clock edge of Byte1 to the 1st clock edge of Byte2		1.5TCY + 40	—	ns	(Note 2)
74	TscH2diL, TscL2diL	Hold time of SDI data input to SCK edge		100	—	ns	
75	TdoR	SDO data output rise time	PIC18**C**XXX	—	25	ns	
			PIC18**LC**XXX	—	45	ns	
76	TdoF	SDO data output fall time		—	25	ns	
78	TscR	SCK output rise time (master mode)	PIC18**C**XXX	—	25	ns	
			PIC18**LC**XXX	—	45	ns	
79	TscF	SCK output fall time (master mode)		—	25	ns	
80	TscH2doV, TscL2doV	SDO data output valid after SCK edge	PIC18**C**XXX	—	50	ns	
			PIC18**LC**XXX	—	100	ns	
81	TdoV2scH, TdoV2scL	SDO data output setup to SCK edge		TCY	—	ns	

Note 1: Requires the use of Parameter # 73A.
2: Only if Parameter #s 71A and 72A are used.

Figure 32-14: Example SPI Slave Mode Timing (CKE = 0)



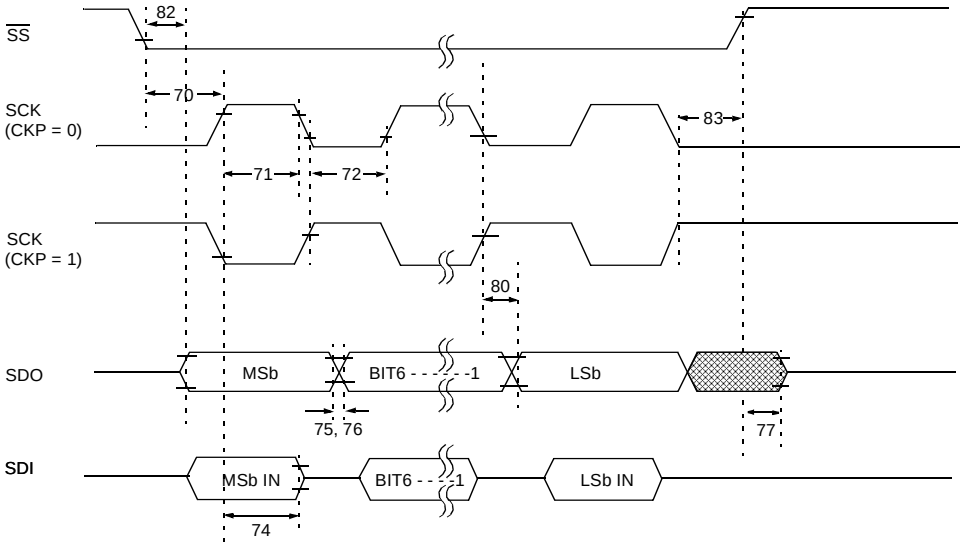
Note: Refer to Figure 32-4 for load conditions.

Table 32-23: Example SPI Mode Requirements (Slave Mode Timing (CKE = 0)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
70	TssL2scH, TssL2scL	$\overline{SS}\downarrow$ to SCK$\downarrow$ or SCK$\uparrow$ input		TCY	—	ns	
71	TscH	SCK input high time (slave mode)	Continuous	1.25TCY + 30	—	ns	
71A			Single Byte	40	—	ns	(Note 1)
72	TscL	SCK input low time (slave mode)	Continuous	1.25TCY + 30	—	ns	
72A			Single Byte	40	—	ns	(Note 1)
73	TdiV2scH, TdiV2scL	Setup time of SDI data input to SCK edge		100	—	ns	
73A	TB2B	Last clock edge of Byte1 to the 1st clock edge of Byte2		1.5TCY + 40	—	ns	(Note 2)
74	TscH2diL, TscL2diL	Hold time of SDI data input to SCK edge		100	—	ns	
75	TdoR	SDO data output rise time	PIC18**C**XXX	—	25	ns	
			PIC18**LC**XXX	—	45	ns	
76	TdoF	SDO data output fall time		—	25	ns	
77	TssH2doZ	$\overline{SS}\uparrow$ to SDO output hi-impedance		10	50	ns	
78	TscR	SCK output rise time (master mode)	PIC18**C**XXX	—	25	ns	
			PIC18**LC**XXX	—	45	ns	
79	TscF	SCK output fall time (master mode)		—	25	ns	
80	TscH2doV, TscL2doV	SDO data output valid after SCK edge	PIC18**C**XXX	—	50	ns	
			PIC18**LC**XXX	—	100	ns	
83	TscH2ssH, TscL2ssH	$\overline{SS}\uparrow$ after SCK edge		1.5TCY + 40	—	ns	

Note 1: Requires the use of Parameter # 73A.
2: Only if Parameter #s 71A and 72A are used.

Figure 32-15: Example SPI Slave Mode Timing (CKE = 1)



Note: Refer to Figure 32-4 for load conditions.

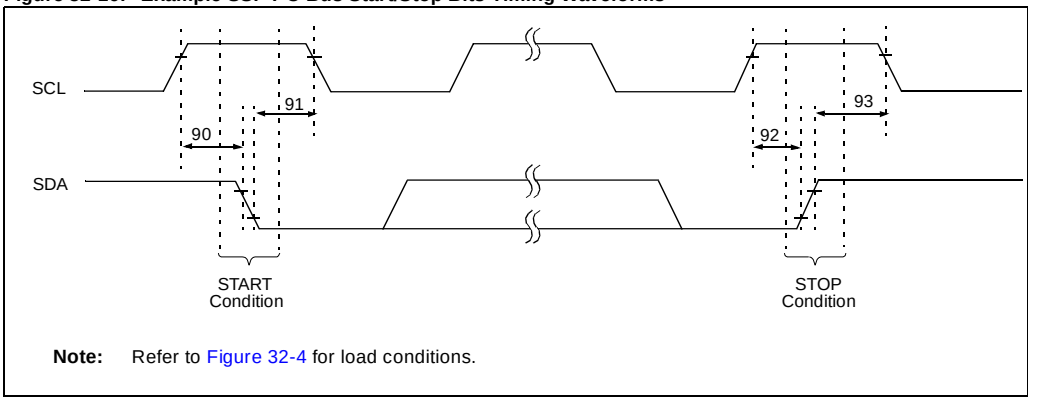
Table 32-24: Example SPI Slave Mode Mode Requirements (CKE = 1)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
70	TssL2scH, TssL2scL	$\overline{SS}\downarrow$ to SCK↓ or SCK↑ input		TCY	—	ns	
71	TscH	SCK input high time (slave mode)	Continuous	1.25TCY + 30	—	ns	
71A			Single Byte	40	—	ns	(Note 1)
72	TscL	SCK input low time (slave mode)	Continuous	1.25TCY + 30	—	ns	
72A			Single Byte	40	—	ns	(Note 1)
73A	TB2B	Last clock edge of Byte1 to the 1st clock edge of Byte2		1.5TCY + 40	—	ns	(Note 2)
74	TscH2diL, TscL2diL	Hold time of SDI data input to SCK edge		100	—	ns	
75	TdoR	SDO data output rise time	PIC18**C**XXX	—	25	ns	
			PIC18**LC**XXX	—	45	ns	
76	TdoF	SDO data output fall time		—	25	ns	
77	TssH2doZ	$\overline{SS}\uparrow$ to SDO output hi-impedance		10	50	ns	
78	TscR	SCK output rise time (master mode)	PIC18**C**XXX	—	25	ns	
			PIC18**LC**XXX	—	45	ns	
79	TscF	SCK output fall time (master mode)		—	25	ns	
80	TscH2doV, TscL2doV	SDO data output valid after SCK edge	PIC18**C**XXX	—	50	ns	
			PIC18**LC**XXX	—	100	ns	
82	TssL2doV	SDO data output valid after $\overline{SS}\downarrow$ edge	PIC18**C**XXX	—	50	ns	
			PIC18**LC**XXX	—	100	ns	
83	TscH2ssH, TscL2ssH	$\overline{SS}\uparrow$ after SCK edge		1.5TCY + 40	—	ns	

Note 1: Requires the use of Parameter # 73A.
2: Only if Parameter #s 71A and 72A are used.

32.21 Example SSP I²C Mode Timing Waveforms and Requirements

Figure 32-16: Example SSP I²C Bus Start/Stop Bits Timing Waveforms

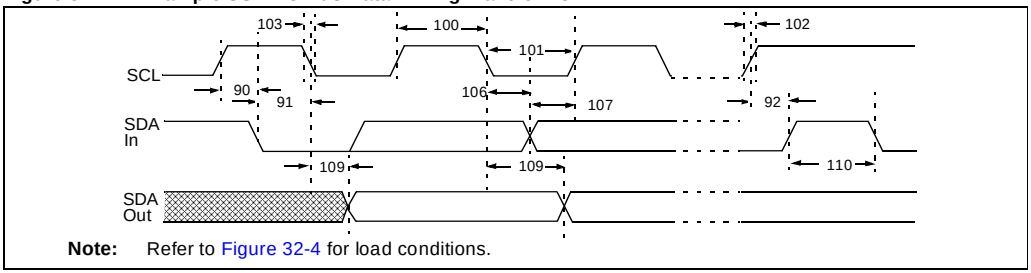


Note: Refer to Figure 32-4 for load conditions.

Table 32-25: Example SSP I²C Bus Start/Stop Bits Requirements (Slave Mode)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
90	TSU:STA	START condition	100 kHz mode	4700	—	ns	Only relevant for repeated START condition
		Setup time	400 kHz mode	600	—		
91	THD:STA	START condition	100 kHz mode	4000	—	ns	After this period the first clock pulse is generated
		Hold time	400 kHz mode	600	—		
92	TSU:STO	STOP condition	100 kHz mode	4700	—	ns	
		Setup time	400 kHz mode	600	—		
93	THD:STO	STOP condition	100 kHz mode	4000	—	ns	
		Hold time	400 kHz mode	600	—		

Figure 32-17: Example SSP I²C Bus Data Timing Waveforms



Note: Refer to Figure 32-4 for load conditions.

Table 32-26: Example SSP I²C bus Data Requirements (Slave Mode)

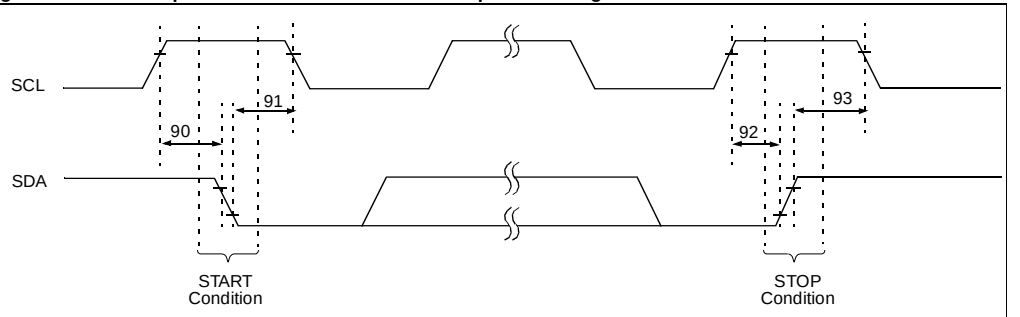
Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
100	THIGH	Clock high time	100 kHz mode	4.0	—	µs	PIC18CXXX must operate at a minimum of 1.5 MHz
			400 kHz mode	0.6	—	µs	PIC18CXXX must operate at a minimum of 10 MHz
			SSP Module	1.5TCY	—	ns	
101	TLOW	Clock low time	100 kHz mode	4.7	—	µs	PIC18CXXX must operate at a minimum of 1.5 MHz
			400 kHz mode	1.3	—	µs	PIC18CXXX must operate at a minimum of 10 MHz
			SSP Module	1.5TCY	—	ns	
102	TR	SDA and SCL rise time	100 kHz mode	—	1000	ns	
			400 kHz mode	20 + 0.1Cb	300	ns	Cb is specified to be from 10 to 400 pF
103	TF	SDA and SCL fall time	100 kHz mode	—	300	ns	
			400 kHz mode	20 + 0.1Cb	300	ns	Cb is specified to be from 10 to 400 pF
90	TSU:STA	START condition setup time	100 kHz mode	4.7	—	µs	Only relevant for repeated START condition
			400 kHz mode	0.6	—	µs	
91	THD:STA	START condition hold time	100 kHz mode	4.0	—	µs	After this period the first clock pulse is generated
			400 kHz mode	0.6	—	µs	
106	THD:DAT	Data input hold time	100 kHz mode	0	—	ns	
			400 kHz mode	0	0.9	µs	
107	TSU:DAT	Data input setup time	100 kHz mode	250	—	ns	(Note 2)
			400 kHz mode	100	—	ns	
92	TSU:STO	STOP condition setup time	100 kHz mode	4.7	—	µs	
			400 kHz mode	0.6	—	µs	
109	TAA	Output valid from clock	100 kHz mode	—	3500	ns	(Note 1)
			400 kHz mode	—	—	ns	
110	TBUF	Bus free time	100 kHz mode	4.7	—	µs	Time the bus must be free before a new transmission can start
			400 kHz mode	1.3	—	µs	
D102	Cb	Bus capacitive loading		—	400	pF	

Note 1: As a transmitter, the device must provide this internal minimum delay time to bridge the undefined region (min. 300 ns) of the falling edge of SCL to avoid unintended generation of START or STOP conditions.

2: A fast-mode I²C-bus device can be used in a standard-mode I²C-bus system, but the requirement tsu;DAT ≥ 250 ns must then be met. This will automatically be the case if the device does not stretch the LOW period of the SCL signal. If such a device does stretch the LOW period of the SCL signal, it must output the next data bit to the SDA line.
TR max. + tsu;DAT = 1000 + 250 = 1250 ns (according to the standard-mode I²C bus specification) before the SCL line is released.

32.22 Example Master SSP I²C Mode Timing Waveforms and Requirements

Figure 32-18: Example Master SSP I²C Bus Start/Stop Bits Timing Waveforms



Note: Refer to Figure 32-4 for load conditions.

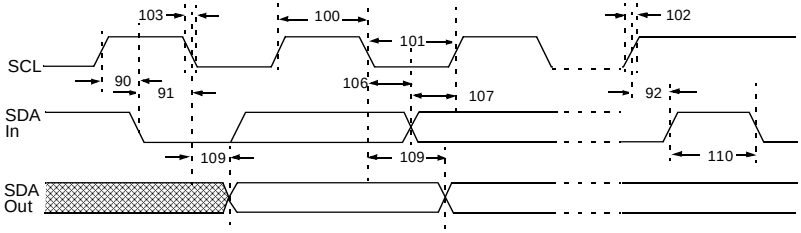
Table 32-27: Example Master SSP I²C Bus Start/Stop Bits Requirements

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
90	TSU:STA	START condition Setup time	100 kHz mode	2(TOSC)(BRG + 1) §	—	ns	Only relevant for repeated START condition
			400 kHz mode	2(TOSC)(BRG + 1) §	—		
			1 MHz mode (1)	2(TOSC)(BRG + 1) §	—		
91	THD:STA	START condition Hold time	100 kHz mode	2(TOSC)(BRG + 1) §	—	ns	After this period the first clock pulse is generated
			400 kHz mode	2(TOSC)(BRG + 1) §	—		
			1 MHz mode (1)	2(TOSC)(BRG + 1) §	—		
92	TSU:STO	STOP condition Setup time	100 kHz mode	2(TOSC)(BRG + 1) §	—	ns	
			400 kHz mode	2(TOSC)(BRG + 1) §	—		
			1 MHz mode (1)	2(TOSC)(BRG + 1) §	—		
93	THD:STO	STOP condition Hold time	100 kHz mode	2(TOSC)(BRG + 1) §	—	ns	
			400 kHz mode	2(TOSC)(BRG + 1) §	—		
			1 MHz mode (1)	2(TOSC)(BRG + 1) §	—		

§ For the value required by the I²C specification, please refer to Figure A-11 of the **"Appendix."**

Note 1: Maximum pin capacitance = 10 pF for all I²C pins.

Figure 32-19: Example Master SSP I²C Bus Data Timing



Note: Refer to Figure 32-4 for load conditions.

Table 32-28: Example Master SSP I²C Bus Data Requirements

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
100	THIGH	Clock high time	100 kHz mode	2(TOSC)(BRG + 1) §	—	ms	
			400 kHz mode	2(TOSC)(BRG + 1) §	—	ms	
			1 MHz mode (1)	2(TOSC)(BRG + 1) §	—	ms	
101	TLOW	Clock low time	100 kHz mode	2(TOSC)(BRG + 1) §	—	ms	
			400 kHz mode	2(TOSC)(BRG + 1) §	—	ms	
			1 MHz mode (1)	2(TOSC)(BRG + 1) §	—	ms	
102	TR	SDA and SCL rise time	100 kHz mode	—	1000	ns	Cb is specified to be from 10 to 400 pF
			400 kHz mode	20 + 0.1Cb	300	ns	
			1 MHz mode (1)	—	300	ns	
103	TF	SDA and SCL fall time	100 kHz mode	—	300	ns	Cb is specified to be from 10 to 400 pF
			400 kHz mode	20 + 0.1Cb	300	ns	
			1 MHz mode (1)	—	100	ns	
90	TSU:STA	START condition setup time	100 kHz mode	2(TOSC)(BRG + 1) §	—	ms	Only relevant for repeated START condition
			400 kHz mode	2(TOSC)(BRG + 1) §	—	ms	
			1 MHz mode (1)	2(TOSC)(BRG + 1) §	—	ms	
91	THD:STA	START condition hold time	100 kHz mode	2(TOSC)(BRG + 1) §	—	ms	After this period the first clock pulse is generated
			400 kHz mode	2(TOSC)(BRG + 1) §	—	ms	
			1 MHz mode (1)	2(TOSC)(BRG + 1) §	—	ms	
106	THD:DAT	Data input hold time	100 kHz mode	0	—	ns	
			400 kHz mode	0	0.9	ms	
			1 MHz mode (1)	TBD	—	ns	
107	TSU:DAT	Data input setup time	100 kHz mode	250	—	ns	(Note 2)
			400 kHz mode	100	—	ns	
			1 MHz mode (1)	TBD	—	ns	
92	TSU:STO	STOP condition setup time	100 kHz mode	2(TOSC)(BRG + 1) §	—	ms	
			400 kHz mode	2(TOSC)(BRG + 1) §	—	ms	
			1 MHz mode (1)	2(TOSC)(BRG + 1) §	—	ms	
109	TAA	Output valid from clock	100 kHz mode	—	3500	ns	
			400 kHz mode	—	1000	ns	
			1 MHz mode (1)	—	—	ns	
110	TBUF	Bus free time	100 kHz mode	4.7 ‡	—	ms	Time the bus must be free before a new transmission can start
			400 kHz mode	1.3 ‡	—	ms	
			1 MHz mode (1)	TBD	—	ms	
D102	Cb	Bus capacitive loading		—	400	pF	

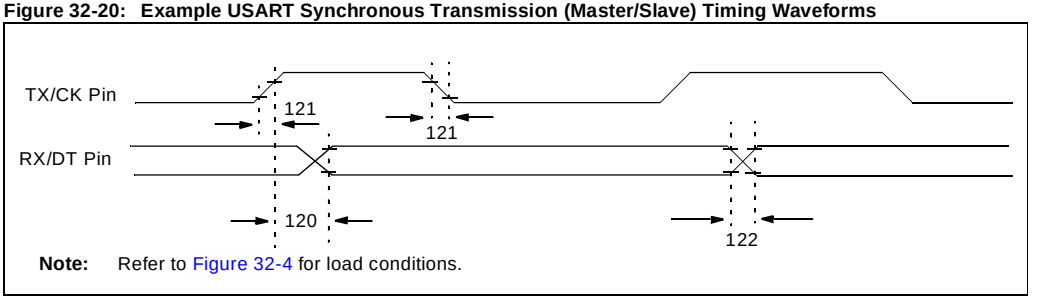
§ For the value required by the I²C specification, please refer to Figure A-11 of the **"Appendix."**

Note 1: Maximum pin capacitance = 10 pF for all I²C pins.

2: A fast-mode I²C-bus device can be used in a standard-mode I²C-bus system, but parameter 107 ≥ 250 ns must then be met. This will automatically be the case if the device does not stretch the LOW period of the SCL signal. If such a device does stretch the LOW period of the SCL signal, it must output the next data bit to the SDA line. Parameter 102.+ parameter 107 = 1000 + 250 = 1250 ns (for 100 kHz-mode) before the SCL line is released.

32.23 Example USART/SCI Timing Waveforms and Requirements

Figure 32-20: Example USART Synchronous Transmission (Master/Slave) Timing Waveforms

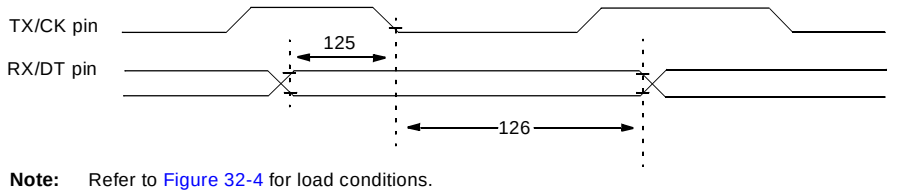


Note: Refer to Figure 32-4 for load conditions.

Table 32-29: Example USART Synchronous Transmission Requirements

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
120	TckH2dtV	SYNC XMIT (MASTER & SLAVE) Clock high to data out valid	PIC18**C**XXX	—	40	ns	
			PIC18**LC**XXX	—	100	ns	
121	Tckrf	Clock out rise time and fall time (Master Mode)	PIC18**C**XXX	—	20	ns	
			PIC18**LC**XXX	—	50	ns	
122	Tdtrf	Data out rise time and fall time	PIC18**C**XXX	—	20	ns	
			PIC18**LC**XXX	—	50	ns	

Figure 32-21: Example USART Synchronous Receive (Master/Slave) Timing Waveforms



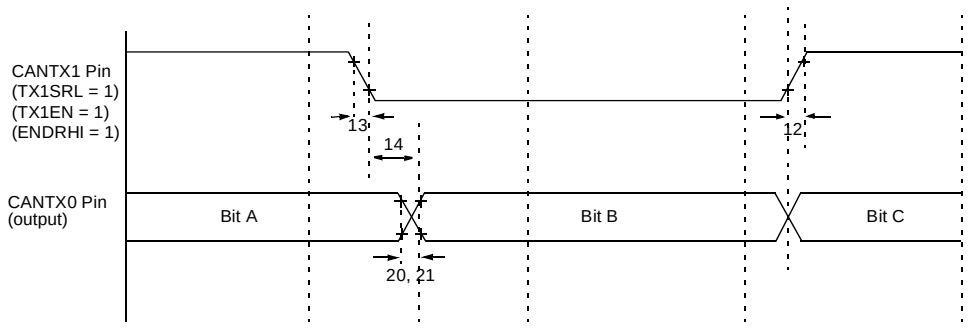
Note: Refer to Figure 32-4 for load conditions.

Table 32-30: Example USART Synchronous Receive Requirements

Param. No.	Symbol	Characteristic	Min	Max	Units	Conditions
125	TdtV2ckl	SYNC RCV (MASTER & SLAVE) Data hold before CK ↓ (DT hold time)	10	—	ns	
126	TckL2dtl	Data hold after CK ↓ (DT hold time)	15	—	ns	

32.24 CAN Specifications

Figure 32-22: Example CAN Timing Waveforms



Note: Refer to Figure 32-4 for load conditions.

Table 32-31: Example CAN Timing

Param. No.	Symbol	Characteristic		Min	Typ †	Max	Units	Conditions
20 20A	TioR	Port output rise time (1)	PIC18**C**XXX	—	10	25	ns	
			PIC18**LC**XXX	—	—	60	ns	
21 21A	TioF	Port output fall time (1)	PIC18**C**XXX	—	10	25	ns	
			PIC18**LC**XXX	—	—	60	ns	
500	Tcanclk2ioV	CANCLK ↓ or CANCLK ↑ to Port out valid		-20	—	20	ns	
501	TrxcanL	Wake-up noise filter		50	—	—	ns	

† These parameters are asynchronous events not related to any internal clock edges.

Note 1: The CAN Clock is driven by the I/O pin drivers, so it has the same timing specification.

32.25 Example 8-bit A/D Timing Waveforms and Requirements

Table 32-32: Example 8-bit A/D Converter Characteristics

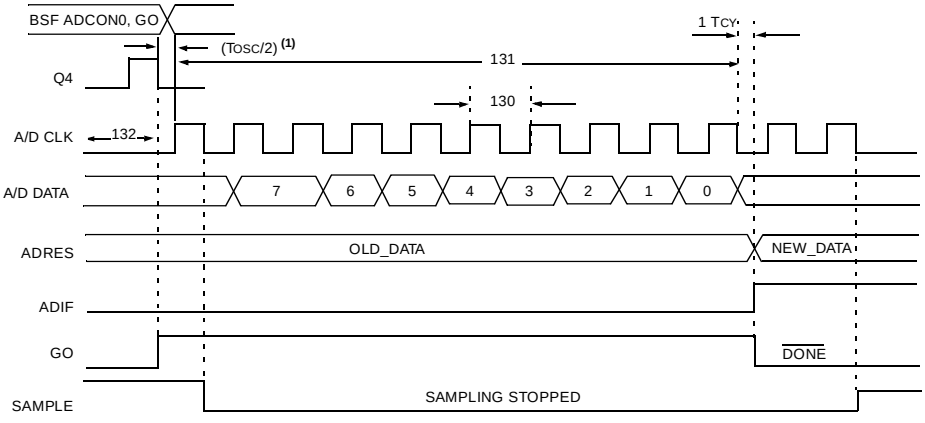
Param No.	Symbol	Characteristic		Min	Typ	Max	Units	Conditions
A01	NR	Resolution (2)		— —	— —	8 8	bits bits	VREF = VDD ≥ 3.0V VREF = VDD = 2.5V
A02	EABS	Total Absolute error (2)		— —	— —	< ± 1 TBD	LSb LSb	VREF = VDD ≥ 3.0V VREF = VDD = 2.5V
A03	EIL	Integral linearity error (2)		— —	— —	< ± 1 TBD	LSb LSb	VREF = VDD ≥ 3.0V VREF = VDD = 2.5V
A04	EDL	Differential linearity error (2)		— —	— —	< ± 1 TBD	LSb LSb	VREF = VDD ≥ 3.0V VREF = VDD = 2.5V
A05	EFS	Full scale error (2)		— —	— —	< ± 1 TBD	LSb LSb	VREF = VDD ≥ 3.0V VREF = VDD = 2.5V
A06	EOFF	Offset error (2)		— —	— —	< ± 1 TBD	LSb LSb	VREF = VDD ≥ 3.0V VREF = VDD = 2.5V
A10	—	Monotonicity		guaranteed (3)			—	VSS ≤ VAIN ≤ VREF
A20	VREF	Reference voltage		0V	—	AVDD	V	VREF delta when changing voltage levels on VREF inputs
A20A		(VREFH - VREFL)		3.0V	—	AVDD	V	Absolute minimum electrical specification to ensure 10-bit accuracy
A21	VREF+	Reference voltage high		AVSS + 3.0		AVDD + 0.3	V	
A22	VREF-	Reference voltage low		AVSS - 0.3		AVDD - 3.0	V	
A25	VAIN	Analog input voltage		AVSS - 0.3	—	AVREF + 0.3	V	
A30	ZAIN	Impedance of analog voltage source		—	—	10.0	kΩ	
A40	IAD	A/D conversion current (VDD)	PIC18**C**XXX	—	180	—	µA	Average current consumption when A/D is on (1)
			PIC18**LC**XXX	—	90	—	µA	
A50	IREF	VREF input current (2)		10 —	— —	1000 10	µA µA	During VAIN acquisition. Based on differential of VHOLD to VAIN to charge CHOLD. See the A/D Converter section. During A/D Conversion cycle

Note 1: When A/D is off, it will not consume any current other than minor leakage current. The power-down current spec includes any such leakage from the A/D module.
VREF current is from RA3 pin or VDD pin, whichever is selected as reference input.

2: VSS ≤ VAIN ≤ VREF

3: The A/D conversion result either increases or remains constant as the analog input increases.

Figure 32-23: Example 8-bit A/D Conversion Timing Waveforms



Note 1: If the A/D clock source is selected as RC, a time of TCY is added before the A/D clock starts. This allows the SLEEP instruction to be executed.

Table 32-33: Example 8-bit A/D Conversion Requirements

Param No.	Symbol	Characteristic		Min	Max	Units	Conditions
130	TAD	A/D clock period	PIC18**C**XXX	1.6	—	µs	TOSC based, VREF ≥ 3.0V
			PIC18**LC**XXX	2.0	—	µs	TOSC based, VREF full range
			PIC18**C**XXX	2.0	6.0	µs	A/D RC Mode
			PIC18**LC**XXX	3.0	9.0	µs	A/D RC Mode
131	TCNV	Conversion time (not including S/H time) (1)		11	11	TAD	
132	TACQ	Acquisition time		Note 2	—	µs	
134	TGO	Q4 to A/D clock start		2TOSC	2TOSC	—	If the A/D clock source is selected as RC, a time of TCY is added before the A/D clock starts. This allows the SLEEP instruction to be executed.
135	TSWC	Switching Time from convert → sample		1	1	TAD	
136	TAMP	Amplifier settling time (2)		1 (5)	—	µs	This may be used if the "new" input voltage has not changed by more than 1LSb (i.e., 5 (20) mV @ 5.12V) from the last sampled voltage (as stated on CHOLD).

Note 1: ADRES register may be read on the following TCY cycle.
2: See the A/D Converter section for minimum requirements.

32.26 Example 10-bit A/D Timing Waveforms and Requirements

Table 32-34: Example 10-bit A/D Converter Characteristics

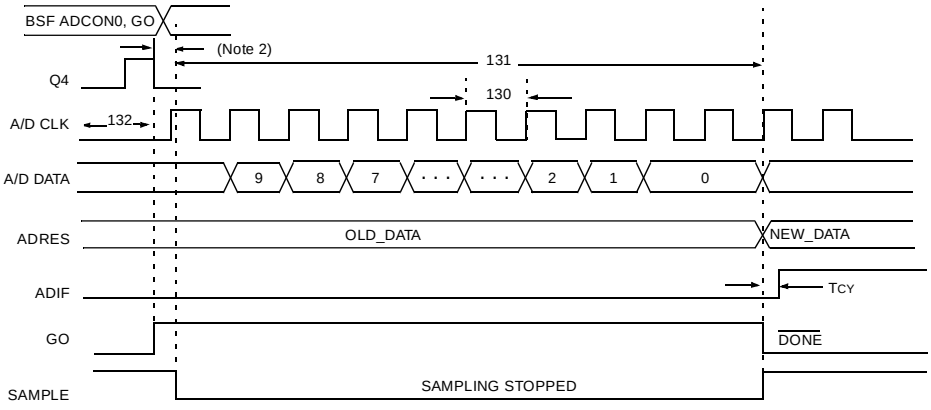
Param No.	Symbol	Characteristic		Min	Typ	Max	Units	Conditions
A01	NR	Resolution		— —	— —	10 TBD	bit bit	VREF = VDD ≥ 3.0V VREF = VDD < 3.0V
A03	EIL	Integral linearity error		— —	— —	< ±1 TBD	LSb LSb	VREF = VDD ≥ 3.0V VREF = VDD < 3.0V
A04	EDL	Differential linearity error		— —	— —	< ±1 TBD	LSb LSb	VREF = VDD ≥ 3.0V VREF = VDD < 3.0V
A05	EFS	Full scale error		— —	— —	< ±1 TBD	LSb LSb	VREF = VDD ≥ 3.0V VREF = VDD < 3.0V
A06	EOFF	Offset error		— —	— —	< ±1 TBD	LSb LSb	VREF = VDD ≥ 3.0V VREF = VDD < 3.0V
A10	—	Monotonicity		guaranteed (3)			—	VSS ≤ VAIN ≤ VREF
A20 A20A	VREF	Reference voltage (VREFH - VREFL)		0V 3V	— —	— —	V V	 For 10-bit resolution
A21	VREFH	Reference voltage High		AVSS	—	AVDD + 0.3V	V	
A22	VREFL	Reference voltage Low		AVSS - 0.3V	—	AVDD	V	
A25	VAIN	Analog input voltage		AVSS - 0.3V	—	VREF + 0.3V	V	
A30	ZAIN	Recommended impedance of analog voltage source		—	—	10.0	kΩ	
A40	IAD	A/D conversion current (VDD)	PIC18**C**XXX	—	180	—	µA	Average current consumption when A/D is on (1)
			PIC18**LC**XXX	—	90	—	µA	
A50	IREF	VREF input current (Note 2)		10 —	— —	1000 10	µA µA	During VAIN acquisition. Based on differential of VHOLD to VAIN. To charge CHOLD see the **"10-bit A/D Converter"** section. During A/D conversion cycle.

Note 1: When A/D is off, it will not consume any current other than minor leakage current. The power-down current spec includes any such leakage from the A/D module.
VREF current is from RG0 and RG1 pins or AVDD and AVSS pins, whichever is selected as reference input.

2: VSS ≤ VAIN ≤ VREF.

3: The A/D conversion result either increases or remains constant as the analog input increases.

Figure 32-24: Example 10-bit A/D Conversion Timing Waveforms



Note 1: If the A/D clock source is selected as RC, a time of TCY is added before the A/D clock starts. This allows the SLEEP instruction to be executed.

2: This is a minimal RC delay (typically 100 nS), which also disconnects the holding capacitor from the analog input.

Table 32-35: Example 10-bit A/D Conversion Requirements

Param No.	Symbol	Characteristic		Min	Max	Units	Conditions
130	TAD	A/D clock period	PIC18**C**XXX	1.6	20 (5)	µs	TOSC based, VREF ≥ 3.0V
			PIC18**LC**XXX	3.0	20 (5)	µs	TOSC based, VREF full range
			PIC18**C**XXX	2.0	6.0	µs	A/D RC Mode
			PIC18**LC**XXX	3.0	9.0	µs	A/D RC Mode
131	TCNV	Conversion time (not including acquisition time) (1)		11 §	12 §	TAD	
132	TACQ	Acquisition time (3)		15 10	— —	µs µs	-40°C ≤ Temp ≤ 125°C 0°C ≤ Temp ≤ 125°C
135	TSWC	Switching Time from convert → sample		—	Note 4		
136	TAMP	Amplifier settling time (2)		1	—	µs	This may be used if the "new" input voltage has not changed by more than 1LSb (i.e., 5 mV @ 5.12V) from the last sampled voltage (as stated on CHOLD).

Note 1: ADRES register may be read on the following TCY cycle.

2: See the **"10-bit A/D Converter"** section for minimum conditions when input voltage has changed more than 1 LSb.

3: The time for the holding capacitor to acquire the "New" input voltage when the voltage changes full scale after the conversion (AVDD to AVSS, or AVSS to AVDD). The source impedance (RS) on the input channels is 50 Ω.

4: On the next Q4 cycle of the device clock.

5: The time of the A/D clock period is dependent on the device frequency and the TAD clock divider.

32.27 Design Tips

No related design tips at this time.

32.28 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced MCU family (that is, they may be written for the Base-Line, the Mid-Range, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to the Electrical Specifications are:

Title	Application Note #

No related Application Notes at this time.

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

32.29 Revision History

Revision A

This is the initial released revision of the Electrical Specifications description.

Section 33. Device Characteristics

HIGHLIGHTS

33.1 Introduction

Microchip Technology Inc. provides characterization information on the devices that it manufactures. This information becomes available after the devices have undergone a complete characterization and the data has been analyzed. This data is taken on both device testers and on bench setups. The characterization data gives the designer a better understanding of the device characteristics, to better judge the acceptability of the device to the application.

33.2 Characterization vs. Electrical Specification

The difference between this information and the Electrical specifications can be classified as what the user should expect the devices to do vs. what Microchip tests the devices to do. The characterization graphs and tables provided are for design guidance and are not tested nor guaranteed.

There may be differences between what the characterization shows as the limits vs. that which is tested, as shown in the Electrical Specification section. This results from capabilities of the production tester equipment, plus whatever guard band that may be necessary.

33.3 DC and AC Characteristics Graphs and Tables

Each table gives specific information that may be useful design information. These values are taken under fixed circumstances. Measurements taken in your application may not lead to the same values if your circumstances are not the same.

In some graphs or tables the data presented are outside specified operating range (i.e., outside specified V_{DD} range). This is for information only and devices will operate properly only within the specified range.

Note:	The data presented in the device Data Sheet Characterization section is a statistical summary of data collected on units from different lots over a period of time and matrix samples. 'Typical' represents the mean of the distribution at, 25°C, while 'max' or 'min' represents (mean +3σ) and (mean -3σ) respectively, where σ is standard deviation.

33.3.1 IPD vs. VDD

IPD is the current (I) that the device consumes when the device is in SLEEP mode (power-down), referred to as Power-down Current. These tests are taken with all I/O as inputs, either pulled high or low. That is, there are no floating inputs, nor are any pins driving an output (with a load).

The characterization shows graphs for both the Watchdog Timer (WDT) disabled and enabled. This is required since the WDT requires an on-chip RC oscillator which consumes additional current.

The device may have certain features and modules that can operate while the device is in SLEEP mode. Some of these modules are:

- Watchdog Timer (WDT)
- Brown-out Reset (BOR) circuitry
- Timer1
- Analog to Digital converter
- LCD module
- Comparators
- Voltage Reference

If these features are operating while the device is in SLEEP mode, a higher current will be consumed. When all features are disabled, the device will consume the lowest possible current (the leakage current). If more then one feature is enabled, then the expected current can easily be calculated as the base current (everything disabled and in SLEEP mode), plus all delta currents. Example 33-1 shows an example of calculating the typical currents for a device at 5V, with the WDT and Timer1 oscillator enabled.

Example 33-1: IPD Calculations with WDT and Timer1 Oscillator Enabled (@ 5V)

Base Current	14 nA	; Device leakage current
WDT Delta Current	14 µA	; 14 µA - 14 nA = 14 µA
Timer1 Delta Current	22 µA	; 22 µA - 14 nA = 22 µA
Total SLEEP Current	36 µA	;

Figure 33-1: Example Typical IPD vs. VDD (WDT Disabled, RC Mode)

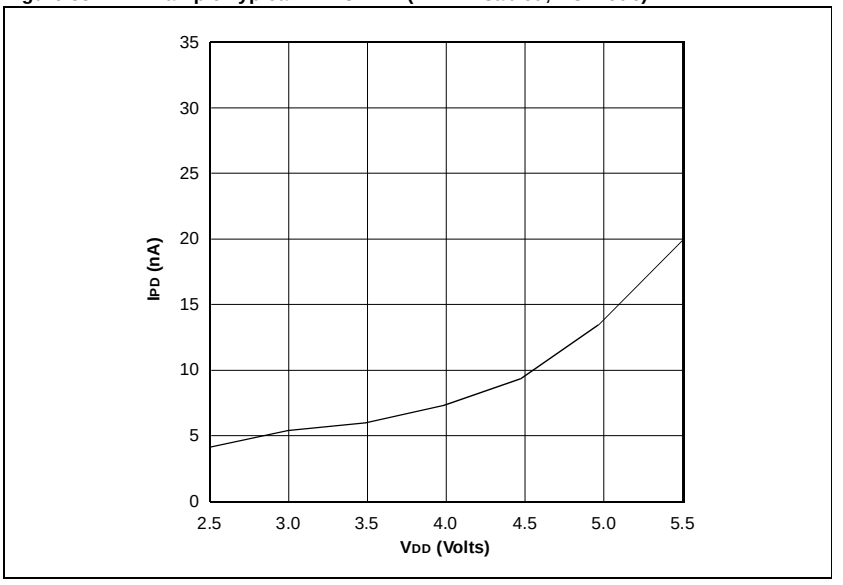


Figure 33-2: Example Maximum IPD vs. VDD (WDT Disabled, RC Mode)

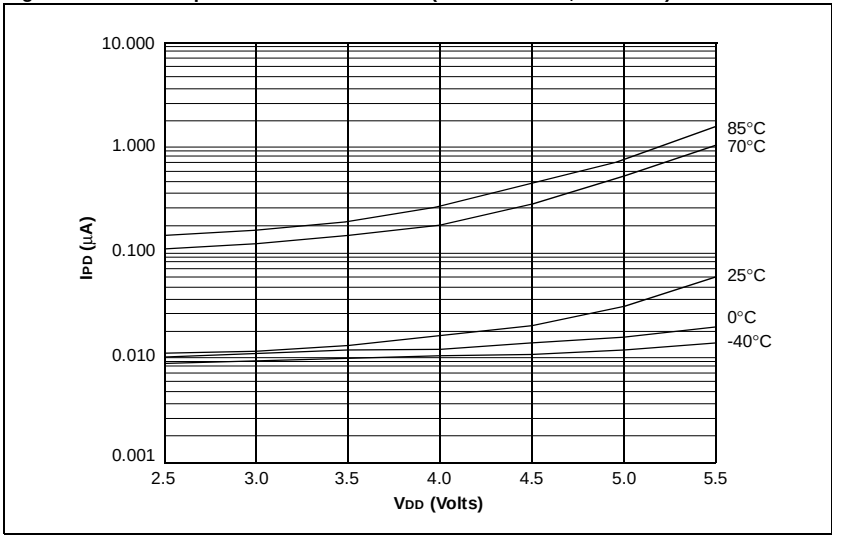


Figure 33-3: Example Typical IPD vs. VDD @ 25°C (WDT Enabled, RC Mode)

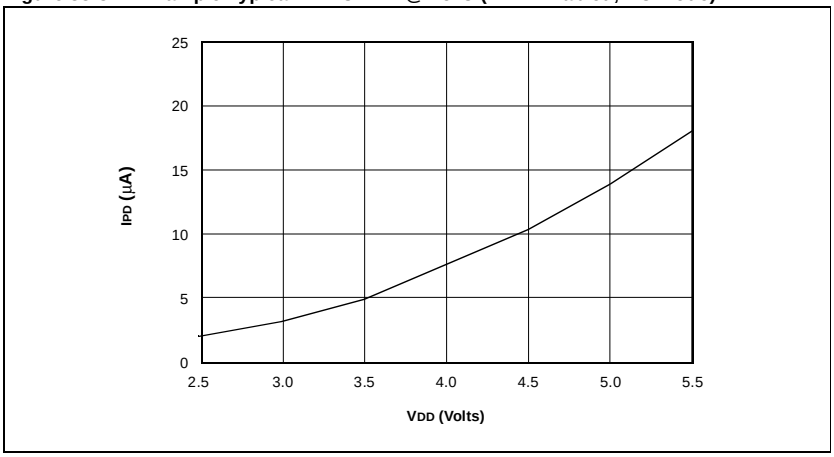


Figure 33-4: Example Maximum IPD vs. VDD (WDT Enabled, RC Mode)

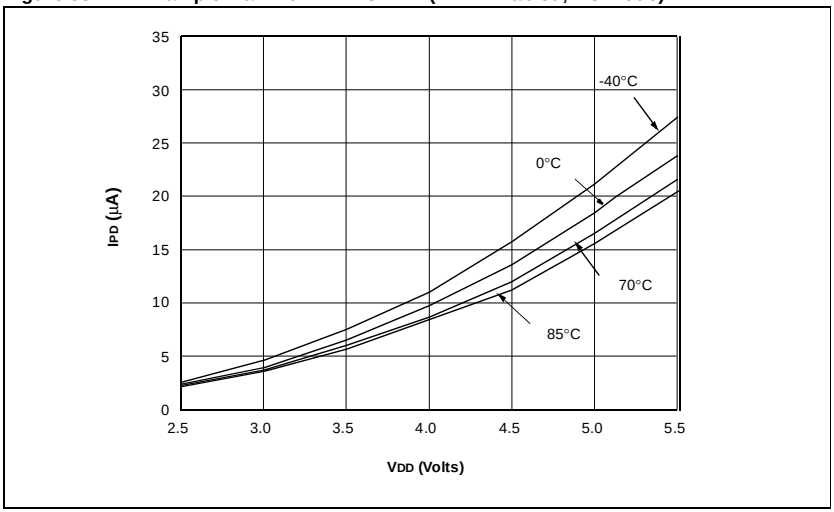
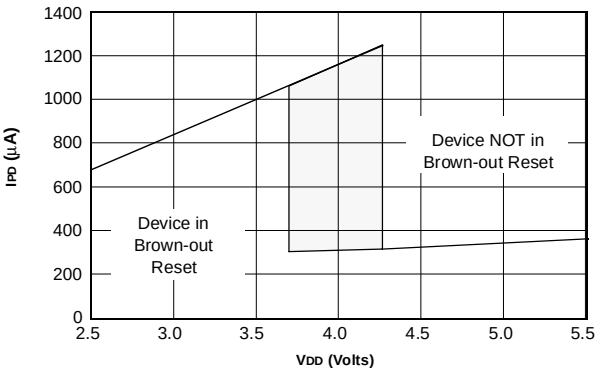
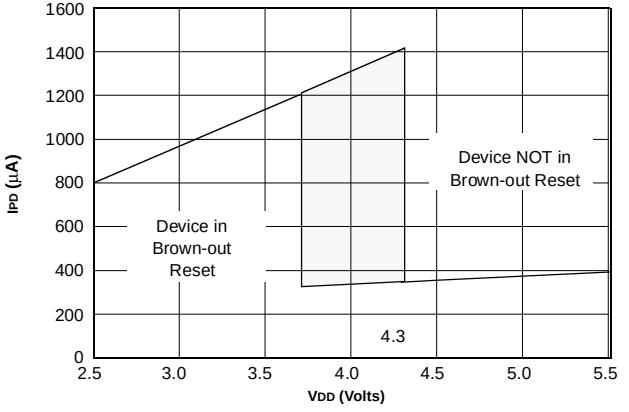


Figure 33-5: Example Typical IPD vs. VDD Brown-out Detect Enabled (RC Mode)



The shaded region represents the built-in hysteresis of the Brown-out Reset circuitry.

Figure 33-6: Example Maximum IPD vs. VDD Brown-out Detect Enabled (85°C to -40°C, RC Mode)



The shaded region represents the built-in hysteresis of the Brown-out Reset circuitry.

Figure 33-7: Example Typical IPD vs. Timer1 Enabled (32 kHz, RC0/RC1 = 33 pF/33 pF, RC Mode)

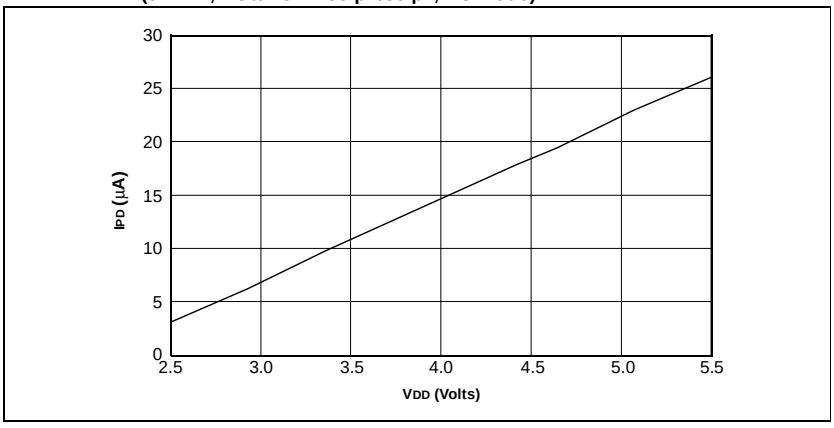
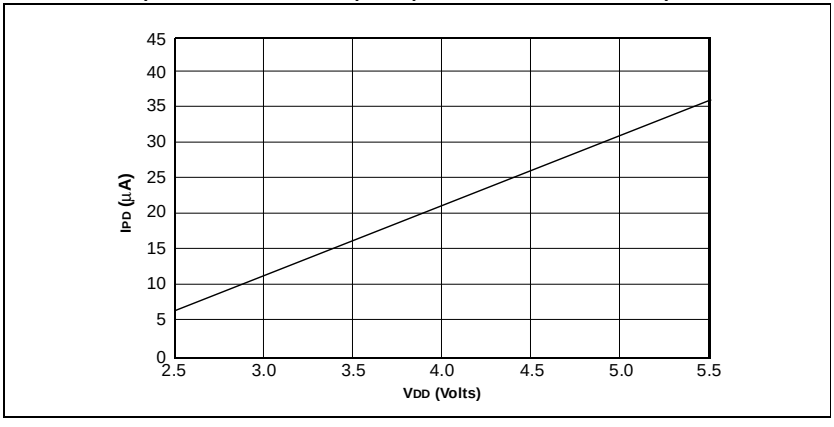


Figure 33-8: Example Maximum IPD vs. Timer1 Enabled (32 kHz, RC0/RC1 = 33 pF/33 pF, -40°C to 85°C, RC Mode)



33.3.2 IDD vs. Frequency

IDD is the current (I) that the device consumes when the device is in operating mode. This test is taken with all I/O as inputs, either pulled high or low. That is, there are no floating inputs, nor are any pins driving an output (with a load).

The IDD vs. Frequency charts measure the results on a Microchip automated bench setup, called the DCS (**D**ata **C**ollection **S**ystem). The DCS accurately reflects the device and specified component values, that is, it does not add stray capacitance or current.

33.3.2.1 RC Measurements

For the RC measurement, the DCS selects a resistor and capacitor value and then, varies the voltage over the specified range. As the voltage is changed, the frequency of operation changes. For a fixed RC, as VDD increases, the frequency increases. After the measurement at this RC has been taken, the RC value is changed and the measurements are taken again. Each point on the graph corresponds to a device voltage, resistor value (R), and capacitor value (C).

Figure 33-9: Example Typical IDD vs. Frequency (RC Mode @ 22 pF, 25°C)

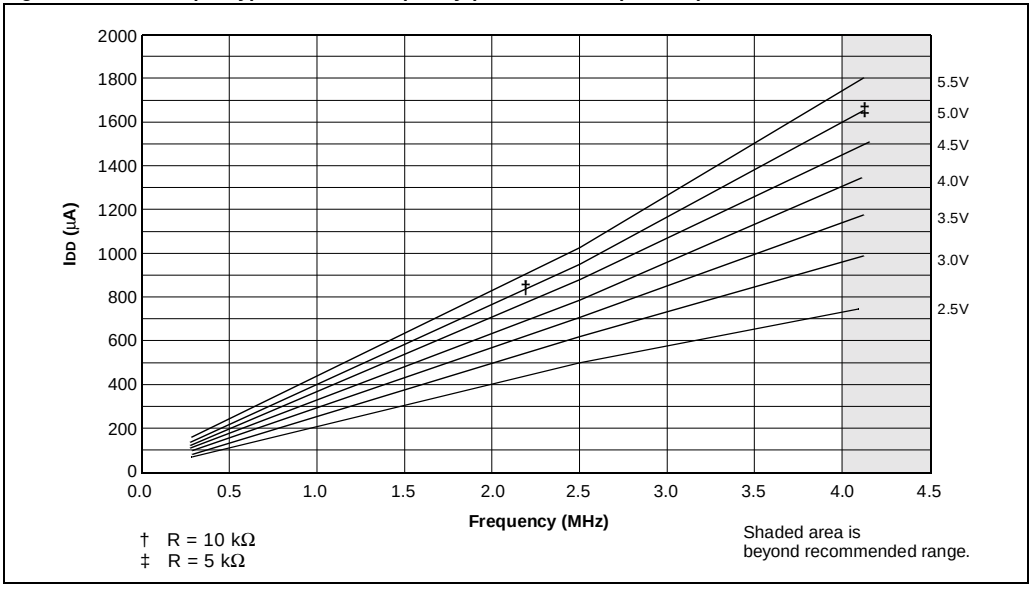


Figure 33-10: Example Maximum IDD vs. Frequency (RC Mode @ 22 pF, -40°C to 85°C)

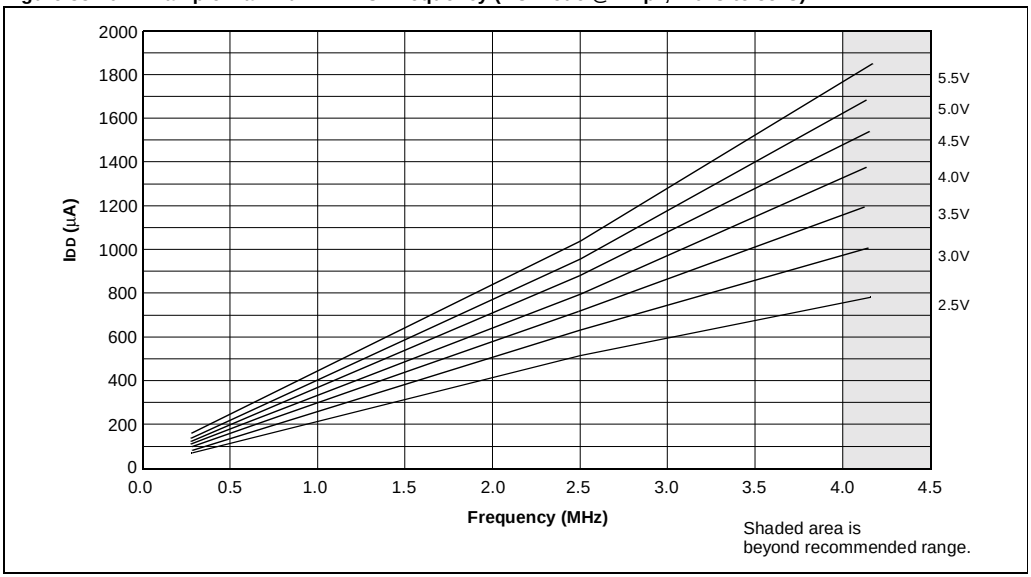


Figure 33-11: Example Typical IDD vs. Frequency (RC Mode @ 100 pF, 25°C)

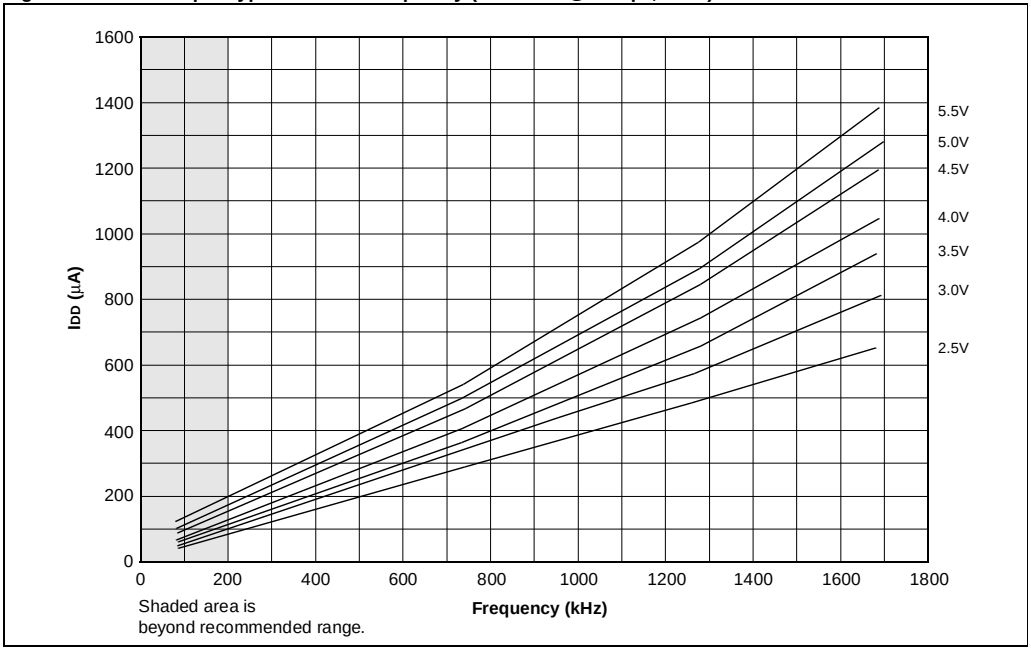


Figure 33-12: Example Maximum IDD vs. Frequency (RC Mode @ 100 pF, -40°C to 85°C)

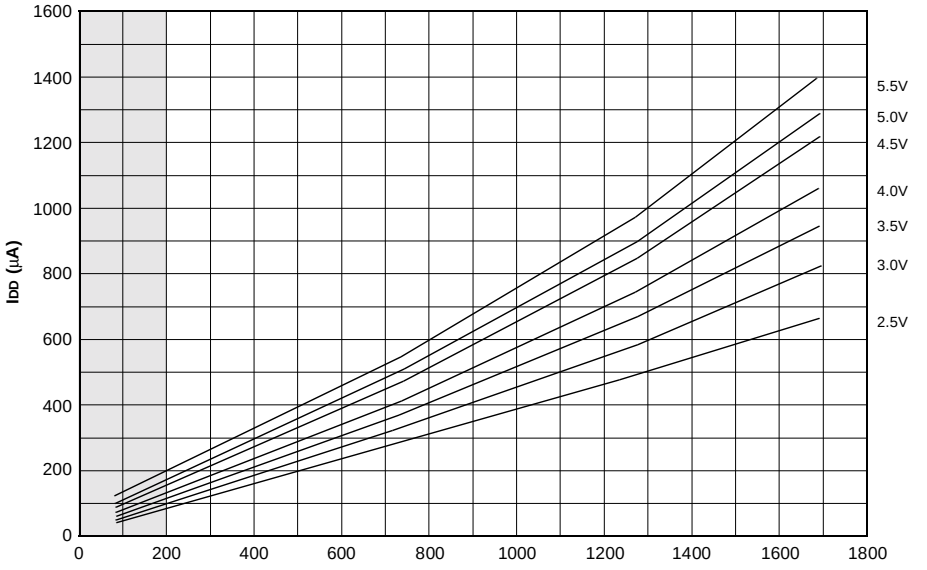


Figure 33-13: Example Typical IDD vs. Frequency (RC Mode @ 300 pF, 25°C)

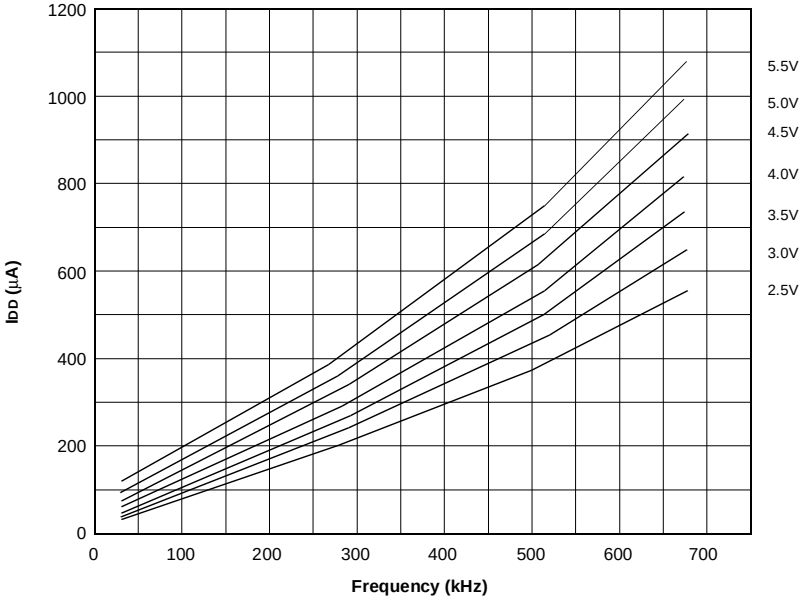


Figure 33-14: Example Maximum IDD vs. Frequency (RC Mode @ 300 pF, -40°C to 85°C)

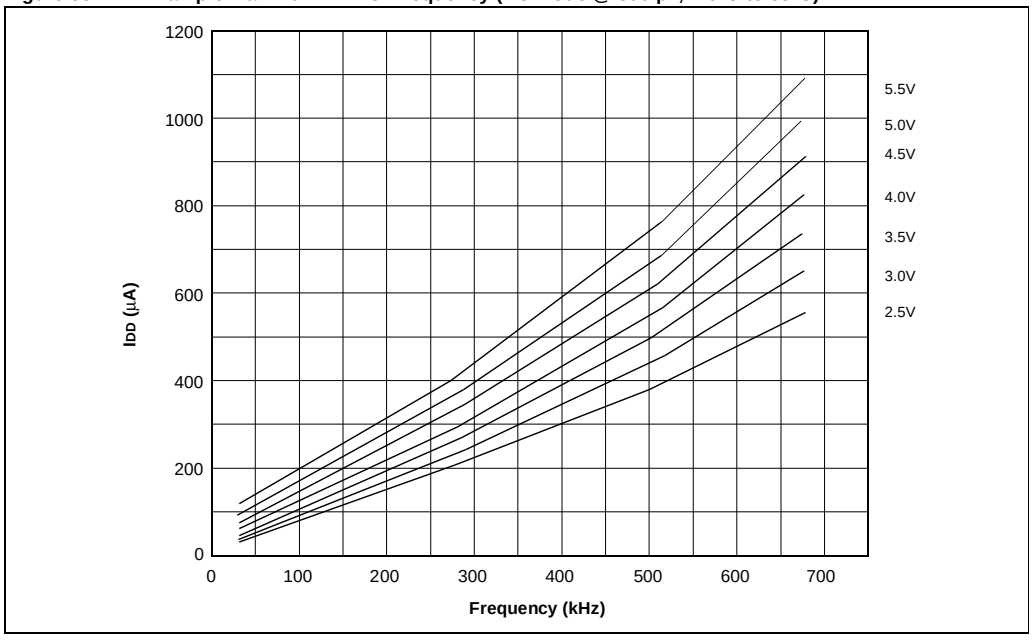
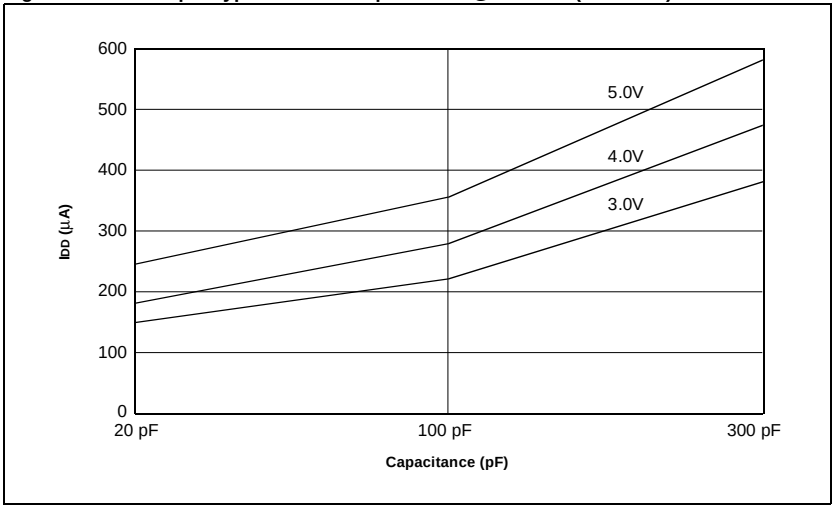


Figure 33-15:Example Typical IDD vs. Capacitance @ 500 kHz (RC Mode)



33.3.2.2 Crystal Oscillator Measurements

On the Data Collection System, there are several crystals. For this test, a crystal is multiplexed into the device circuit and the crystal's capacitance values can be varied. The capacitance and voltage values are varied to determine the best characteristics (current, oscillator waveform and oscillator start-up), and then the currents are measured over voltage. The next crystal oscillator is then switched in and the procedure is repeated.

Figure 33-16: Example Typical IDD vs. Frequency (LP Mode, 25°C)

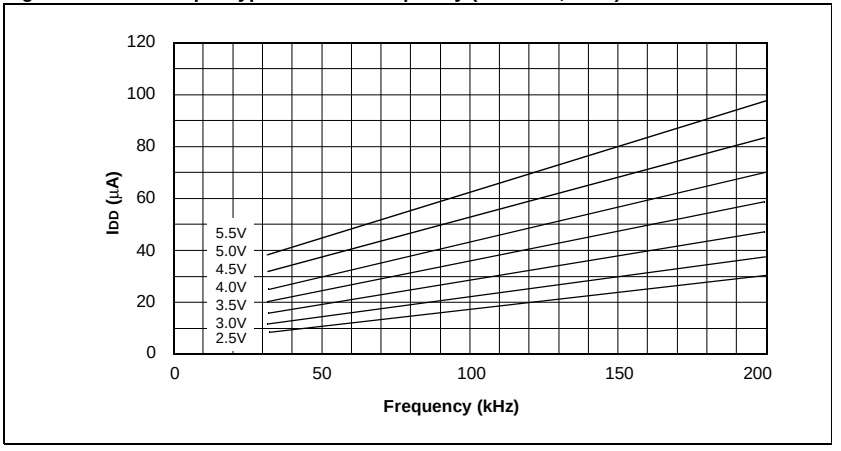


Figure 33-17: Example Maximum IDD vs. Frequency (LP Mode, -40°C to 85°C)

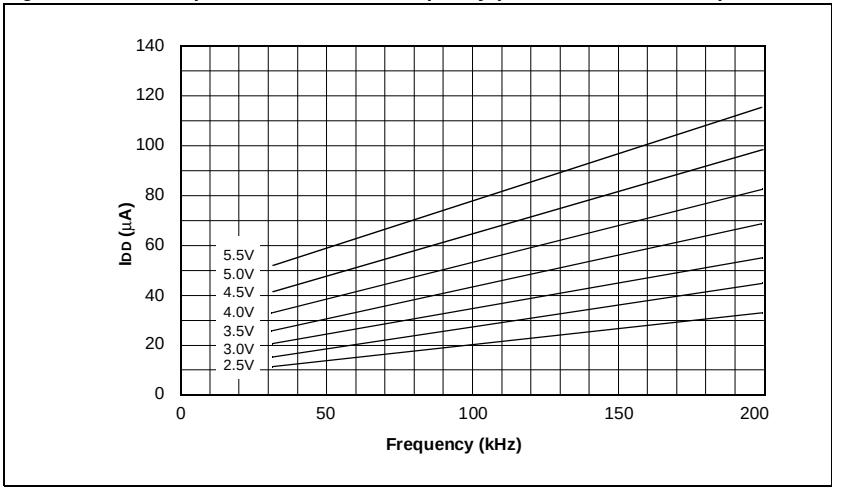


Figure 33-18: Example Typical IDD vs. Frequency (XT Mode, 25°C)

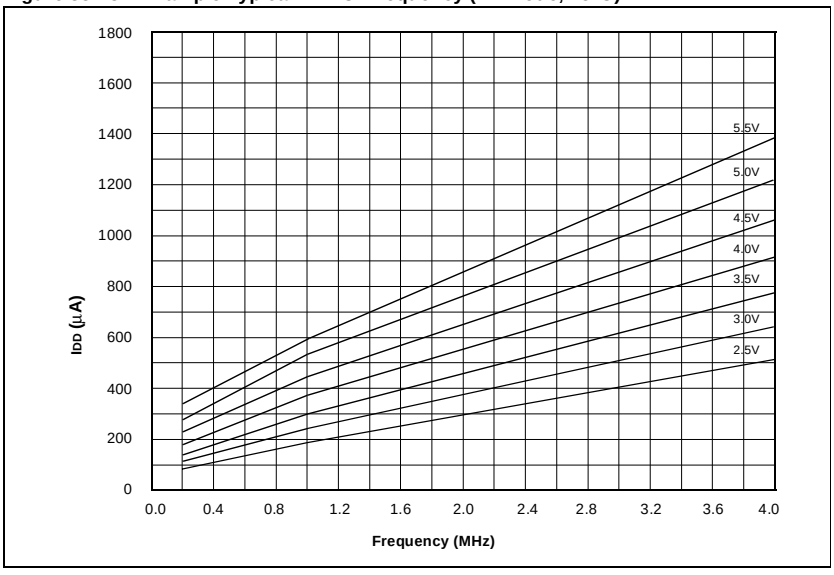


Figure 33-19: Example Maximum IDD vs. Frequency (XT Mode, -40°C to 85°C)

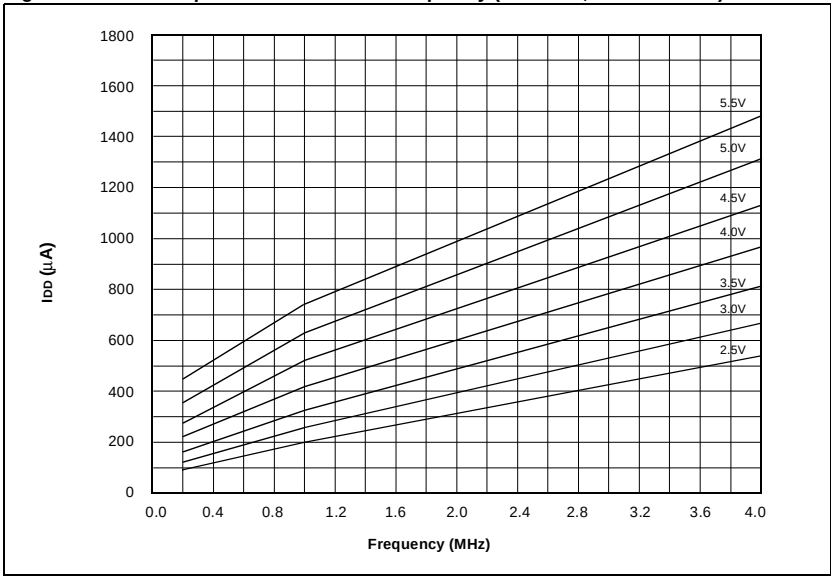


Figure 33-20: Example Typical IDD vs. Frequency (HS Mode, 25°C)

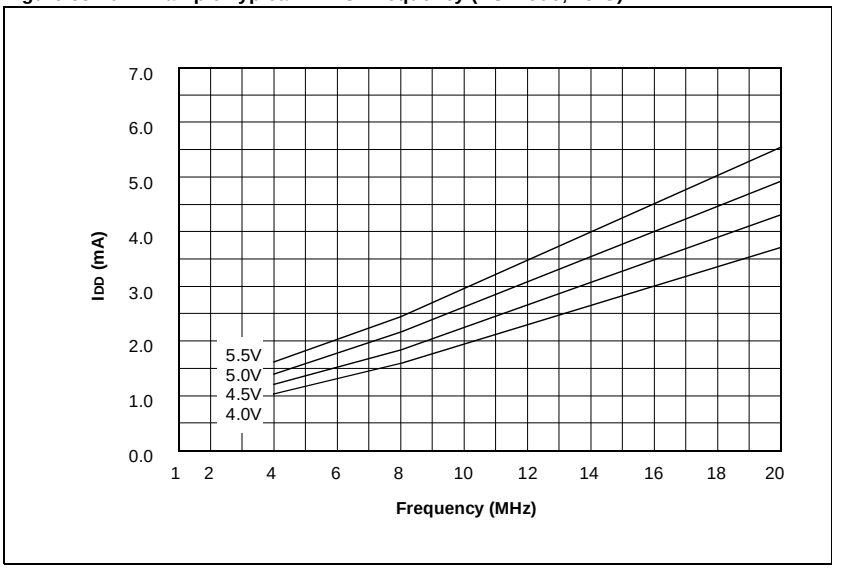
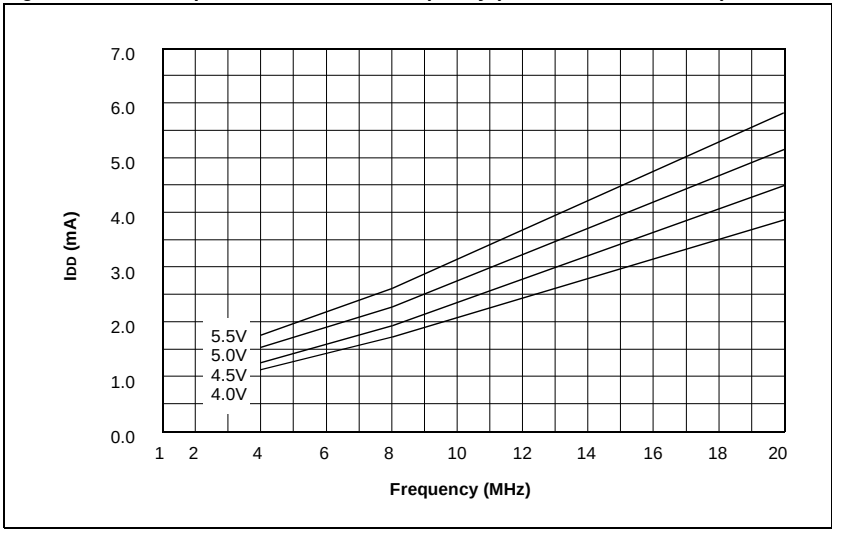


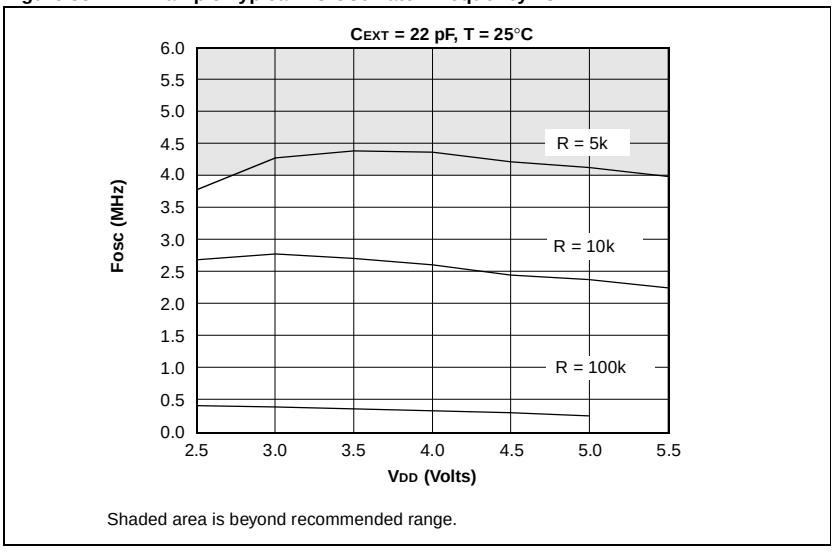
Figure 33-21: Example Maximum IDD vs. Frequency (HS Mode, -40°C to 85°C)



33.3.3 RC Oscillator Frequency

These tables show the effects of the RC oscillator frequency as the device voltage varies. In these measurements, a capacitor and resistor value are selected and then, the frequency of the RC is measured, as the device voltage varies. The table shows the typical frequency for a R and C value at 5V, as well as the variation from this frequency that can be expected, due to device processing.

Figure 33-22: Example Typical RC Oscillator Frequency vs. VDD



Shaded area is beyond recommended range.

Figure 33-23: Example Typical RC Oscillator Frequency vs. VDD

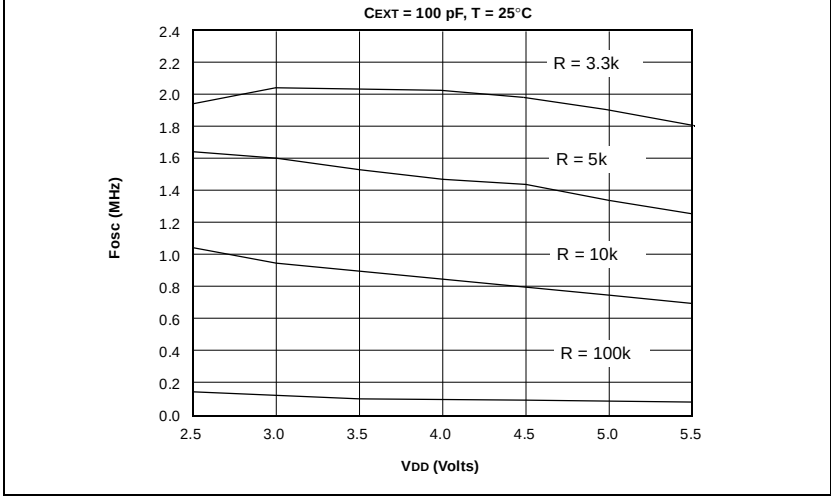


Figure 33-24: Example Typical RC Oscillator Frequency vs. VDD

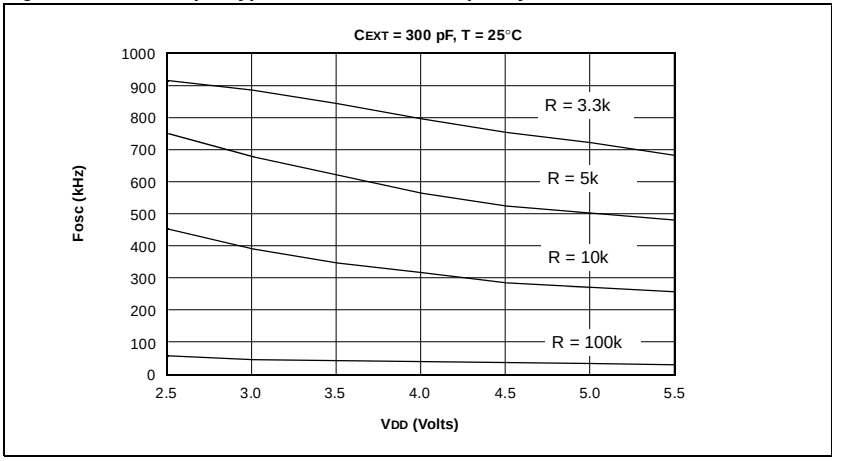


Table 33-1: Example RC Oscillator Frequencies

CEXT	REXT	Average	
		Fosc @ 5V, 25°C	
22 pF	5k	4.12 MHz	± 1.4%
	10k	2.35 MHz	± 1.4%
	100k	268 kHz	± 1.1%
100 pF	3.3k	1.80 MHz	± 1.0%
	5k	1.27 MHz	± 1.0%
	10k	688 kHz	± 1.2%
	100k	77.2 kHz	± 1.0%
300 pF	3.3k	707 kHz	± 1.4%
	5k	501 kHz	± 1.2%
	10k	269 kHz	± 1.6%
	100k	28.3 kHz	± 1.1%

The percentage variation indicated here is part to part variation due to normal process distribution. The variation indicated is ±3 standard deviation from average value for VDD = 5V.

33.3.4 Oscillator Transconductance

Transconductance of the oscillator indicates the gain of the oscillator. As the transconductance increases, the gain of the oscillator circuit increases, which causes the current consumption of the oscillator circuit to increase. Also, as the transconductance increases, the maximum frequency that the oscillator circuit can support also increases, or the start-up time of the oscillator decreases.

Figure 33-25: Example Transconductance (gm) of HS Oscillator vs. VDD

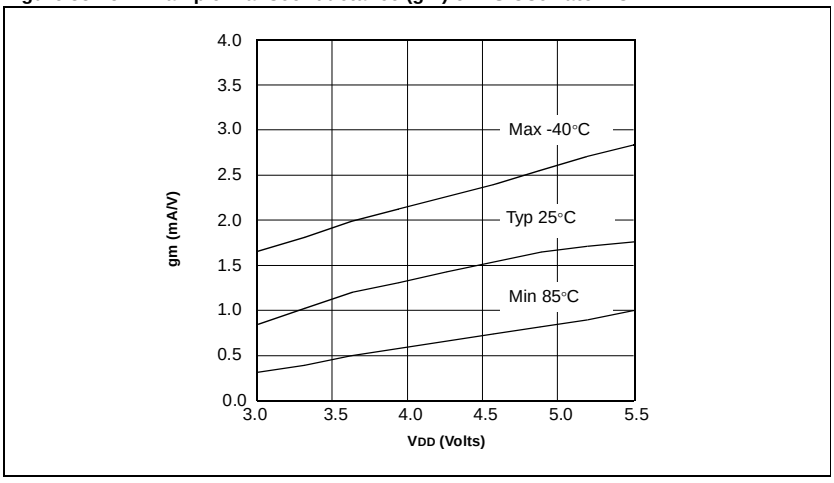


Figure 33-26: Example Transconductance (gm) of LP Oscillator vs. VDD

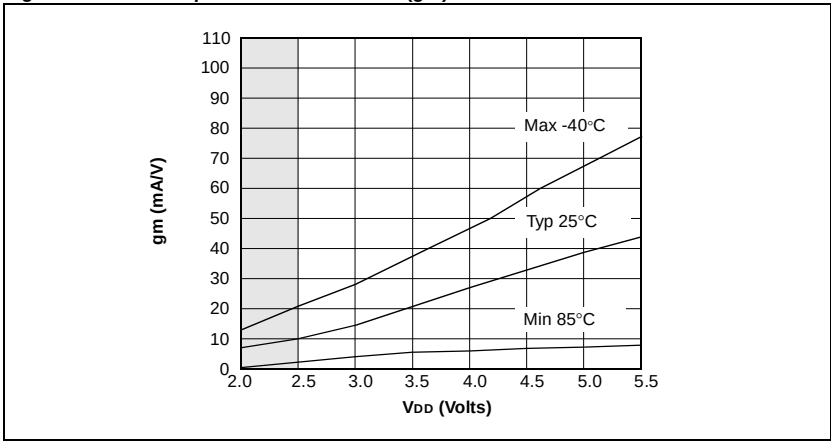
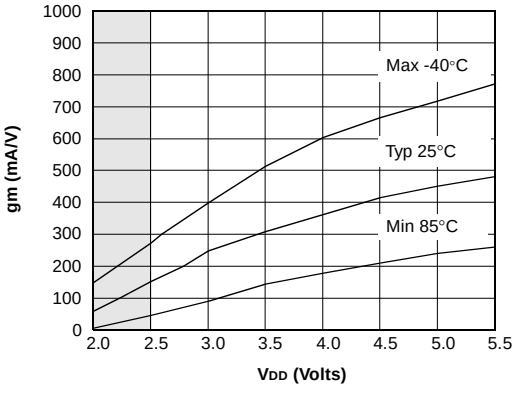


Figure 33-27: Example Transconductance (gm) of XT Oscillator vs. VDD



Shaded area is beyond recommended range.

33.3.5 Crystal Start-up Time

These graphs show the start-up time that one should expect to see at the specified voltage level, for a given crystal/capacitor combination.

Figure 33-28: Example Typical XTAL Start-up Time vs. VDD (LP Mode, 25°C)

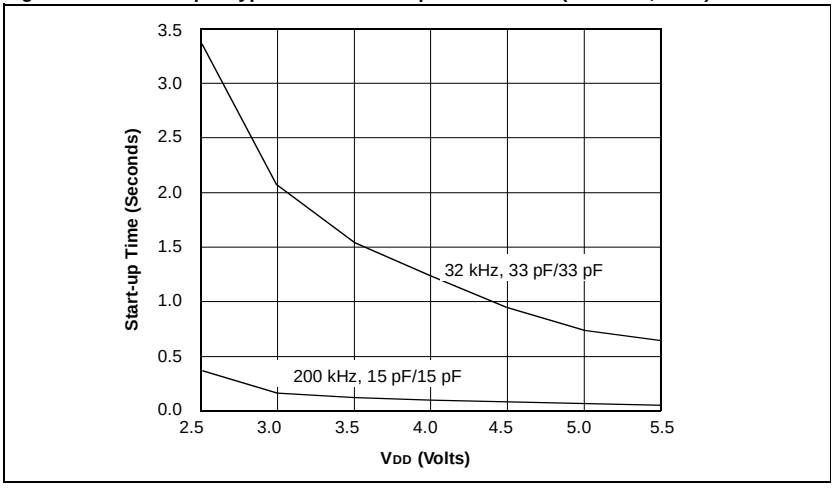


Figure 33-29: Example Typical XTAL Start-up Time vs. VDD (HS Mode, 25°C)

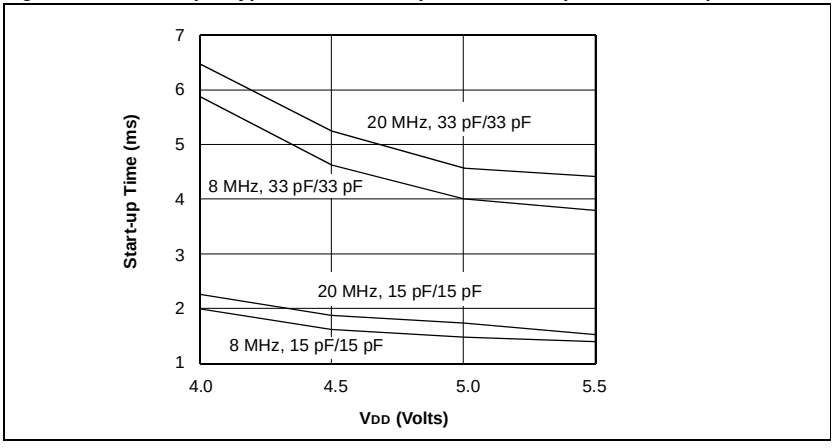


Figure 33-30: Example Typical XTAL Start-up Time vs. VDD (XT Mode, 25°C)

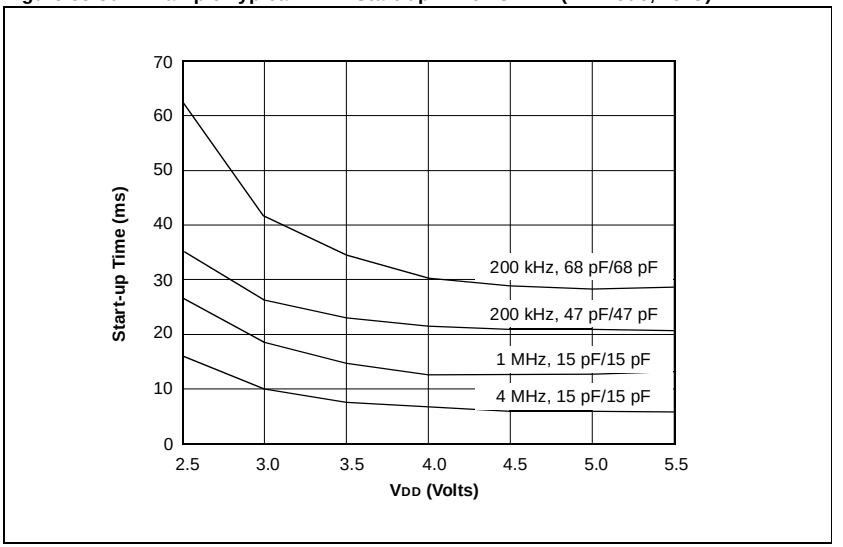


Figure 33-31: Example V_{TH} (Input Threshold Trip Point Voltage) of I/O Pins vs. V_{DD}

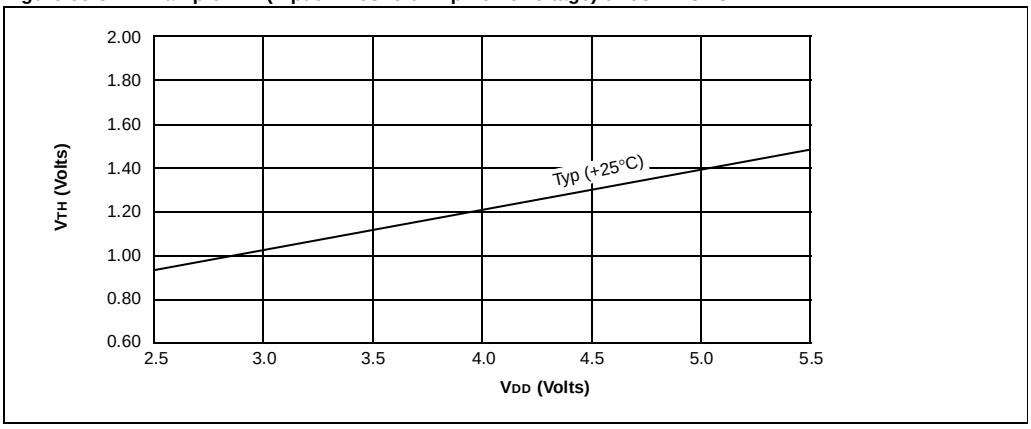
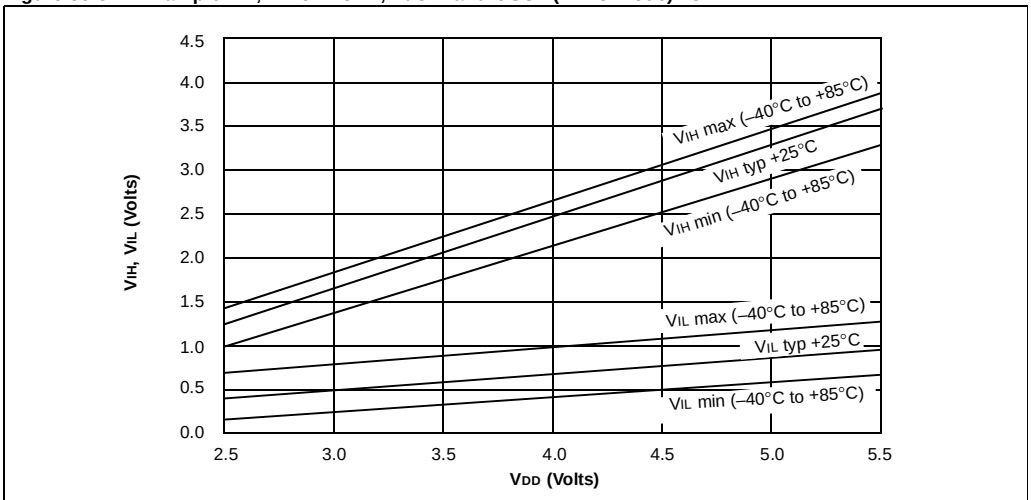


Figure 33-32: Example V_{IH}, V_{IL} of $\overline{MCLR}$, T0CKI and OSC1 (in RC Mode) vs. V_{DD}



Note: These input pins have Schmitt Trigger input buffers.

Figure 33-33: Example V_{TH} (Input Threshold Trip Point Voltage) of OSC1 Input (in XT, HS, and LP modes) vs. V_{DD}

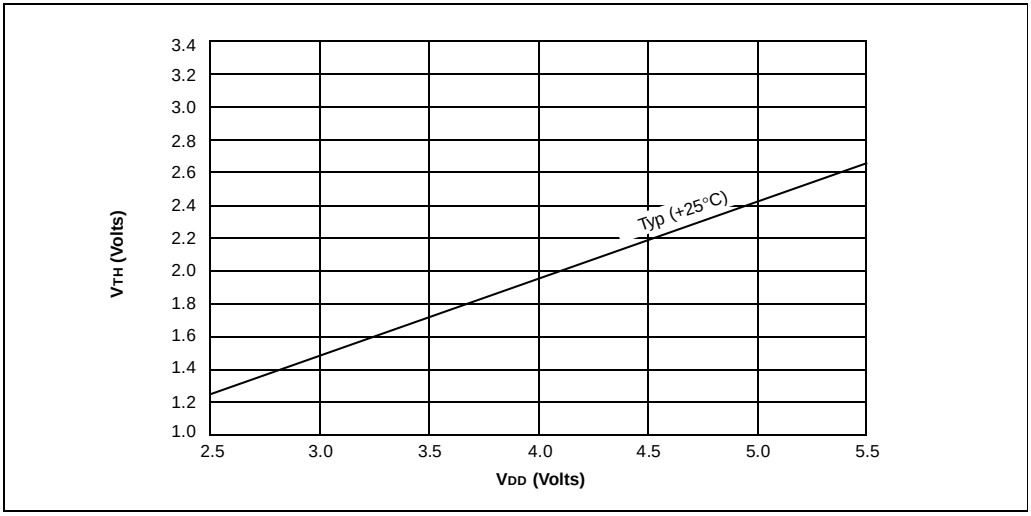


Figure 33-34: Example WDT Timer Time-out Period vs. V_{DD}

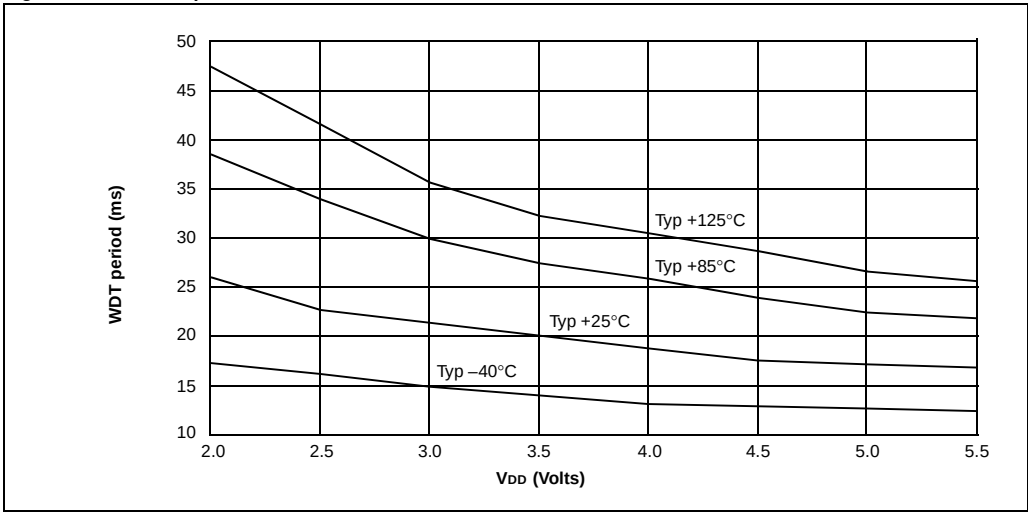


Figure 33-35: Example I_{OH} vs. V_{OH}, V_{DD} = 3 V

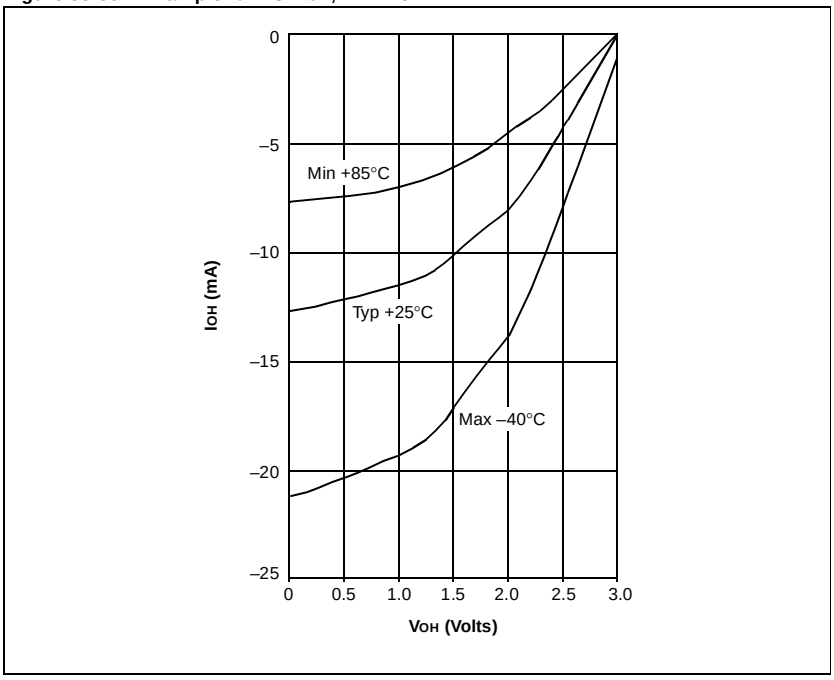


Figure 33-36: Example I_{OH} vs. V_{OH}, V_{DD} = 5 V

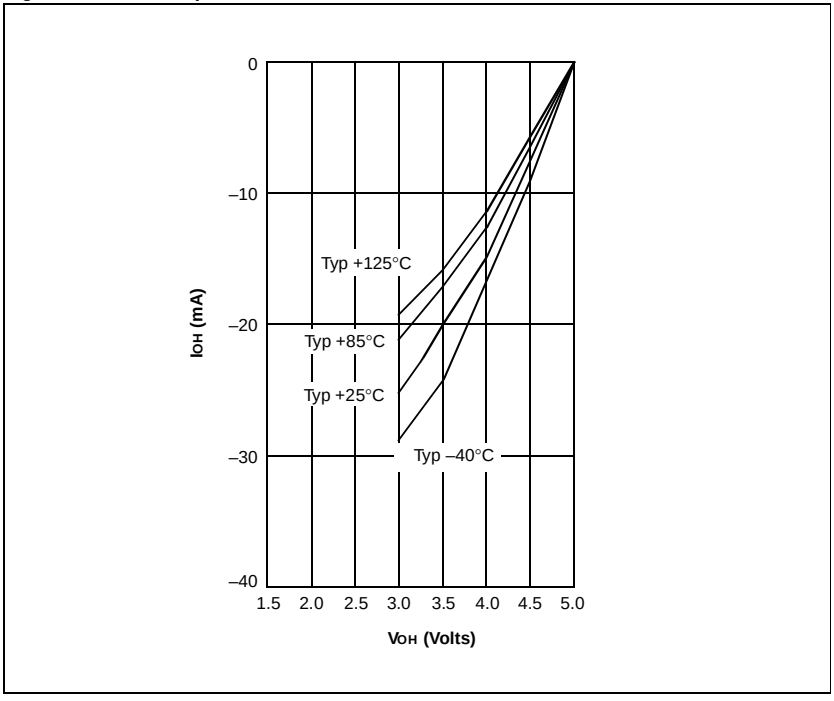


Figure 33-37: Example I_{OL} vs. V_{OL}, V_{DD} = 3 V

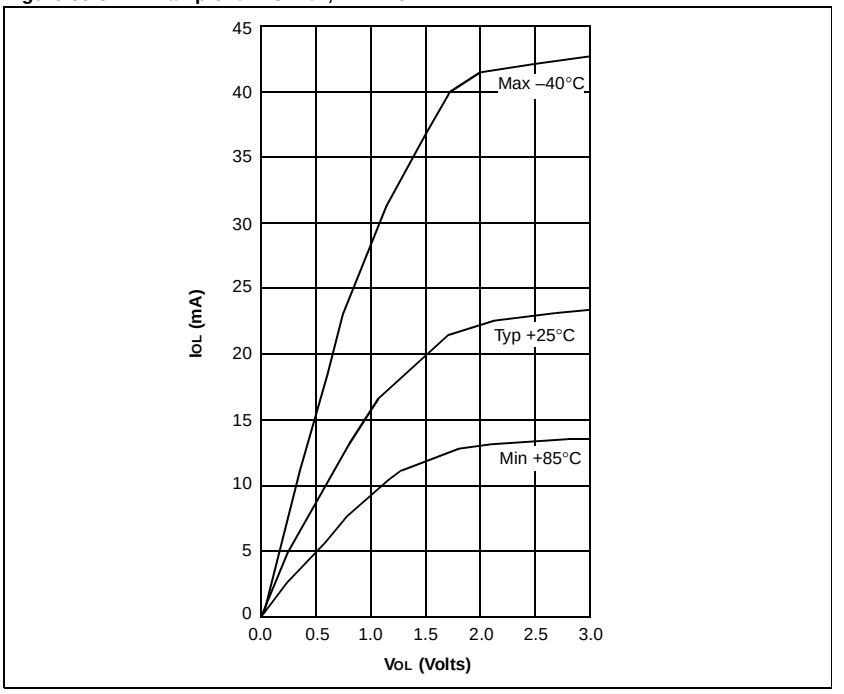
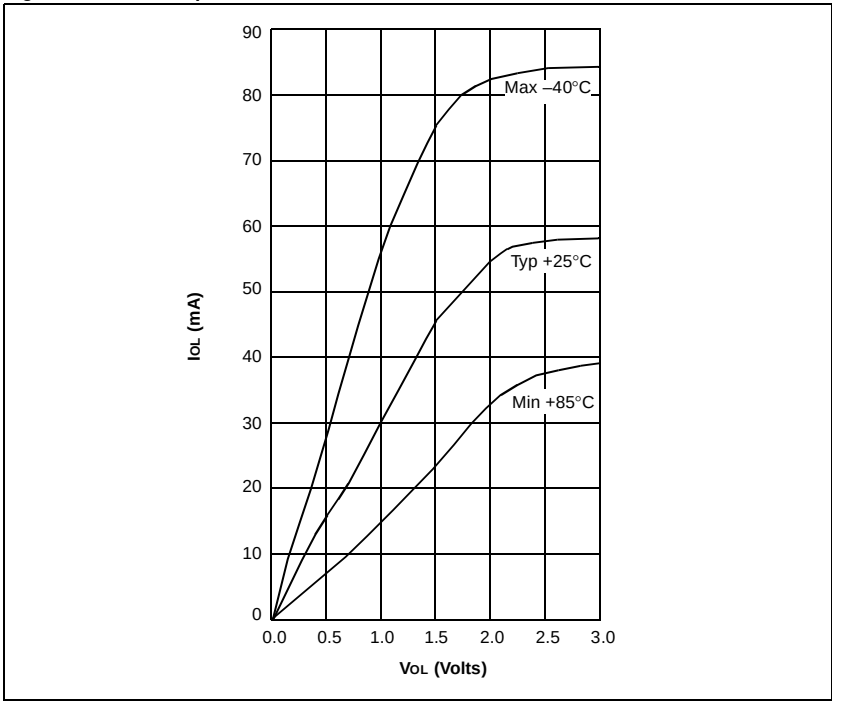


Figure 33-38: Example I_{OL} vs. V_{OL}, V_{DD} = 5 V



33.3.6 Tested Crystals and Their Capacitor Values

This table shows the crystal frequency and manufacturer that was used for every test in this section, as well as the capacitor values/ranges that exhibited the best characteristics.

Table 33-2: Example Capacitor Selection for Crystal Oscillators

Osc Type	Crystal Frequency	Capacitor Range C1	Capacitor Range C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	8 MHz	15-33 pF	15-33 pF
	20 MHz	15-33 pF	15-33 pF
Note: Higher capacitance increases the stability of the oscillator but also increases the start-up time. These values are for design guidance only. ***Rs*** may be required in HS mode, as well as XT mode, to avoid overdriving crystals with low drive level specification. Since each crystal has its own characteristics, the user should consult the crystal manufacturer for appropriate values of external components or verify oscillator performance.			
Crystals Used:			
32 kHz	Epson C-001R32.768K-A	± 20 PPM	
200 kHz	STD XTL 200.000KHz	± 20 PPM	
1 MHz	ECS ECS-10-13-1	± 50 PPM	
4 MHz	ECS ECS-40-20-1	± 50 PPM	
8 MHz	EPSON CA-301 8.000M-C	± 30 PPM	
20 MHz	EPSON CA-301 20.000M-C	± 30 PPM	

33.3.7 Example EPROM Memory Erase Times

The UV erase time of an EPROM cell depends on the geometry size of the EPROM cell and the manufacturing technology. Table 33-3 shows some of the expected erase times for each different device.

Table 33-3: Example of Typical EPROM Erase Time Recommendations

Example Device	Wavelength (Angstroms)	Intensity ($\mu W/cm^2$)	Distance from UV Lamp (inches)	Typical Time (1) (minutes)
1	2537	12,000	1	60

Note 1: If these criteria are not met, the erase times will be different.

2: Refer to the device data sheet for the typical erase times for a device.

33.4 Revision History

Revision A

This is the initial released revision of the Device Characteristics description.

Section 34. Development Tools

HIGHLIGHTS

This section of the manual contains the following major topics:

34.1 Introduction

Microchip offers a wide range of integrated development tools to ease the application development process. These tools can be broken down into the core development tools and the supplemental tools.

The core tools are as follows:

- MPLAB Integrated Development Environment, including full featured editor
- Language Products
  - MPASM Assembler
  - MPLAB-CXX C Compiler
    - -MPLAB-C17
    - -MPLAB-C18
  - MPLINK/MPLIB Linker Librarian
- MPLAB-SIM Software Simulator
- Real-Time In-Circuit Emulators
  - MPLAB-ICE Real-Time Emulator In-Circuit
  - ICEPIC Low-Cost Emulator with Breakpoint debug capabilities
- In-Circuit Debugger
  - MPLAB-ICD for 16F877
- Device Programmers
  - PRO MATE® II Universal Programmer
  - PICSTART® Plus Entry-Level Development Programmer
- Development Boards
  - PICDEM-1 Low-Cost Demonstration Board
  - PICDEM-2 Low-Cost Demonstration Board
  - PICDEM-3 Low-Cost Demonstration Board
  - PICDEM-17 Low-Cost Demonstration Board
  - PICDEM-14A Low-Cost Demonstration Board

The minimum configuration of MPLAB is the Integrated Development Environment (IDE), the assembler (MPASM), and the software simulator (MPLAB-SIM). Other tools are added to MPLAB as they are installed. This gives a common platform for the design activity, from the writing and assembling of the source code, through the simulation/emulation, to the programming of prototype devices.

Note:	The most current version may be downloaded from Microchip's web site for free.

In addition to Microchip, there are many third party vendors. Microchip's Third Party Handbook gives an overview of the manufactures and their tools.

34.2 The Integrated Development Environment (IDE)

The core set of development tools operate under the IDE umbrella. The IDE is called MPLAB. This gives a consistent look and feel to all the development tools so that minimal learning of the new tool interface is required. The MPLAB IDE integrates all the following aspects of development:

- Source code editing
- Project management
- Machine code generation (from assembly or “C”)
- Device simulation
- Device emulation
- Device programming

MPLAB is a PC based Windows® application. It has been extensively tested using Windows 95 and recommended in either of these operating environments:

- Windows 2000
- Windows NT 4.0
- Windows 98
- Windows 95
- Windows 3.X

This comprehensive tool suite allows the complete development of a project without leaving the MPLAB environment.

Windows is a registered trademark of Microsoft Corporation.

34.2.1 MPLAB

The MPLAB IDE Software brings an ease of software development previously unseen in the 8-bit microcontroller market. MPLAB is a Windows based application that contains:

- A full featured editor
- Four operating modes
  - editor
  - simulator
  - emulator
  - programmer
- A project manager
- Customizable tool bar and key mapping
- A status bar with project information
- Extensive on-line help

MPLAB allows you to:

- Edit your source files. This includes:
  - MPASM assembly language
  - MPLAB-CXX 'C' language
- One touch assemble (or compile) and download to PIC16/17 tools (automatically updates all project information)
- Debug using:
  - source files
  - absolute listing file
  - program memory
- Run up to four MPLAB-ICE emulators on the same PC
- Run or Single-step
  - program memory
  - source file
  - absolute listing

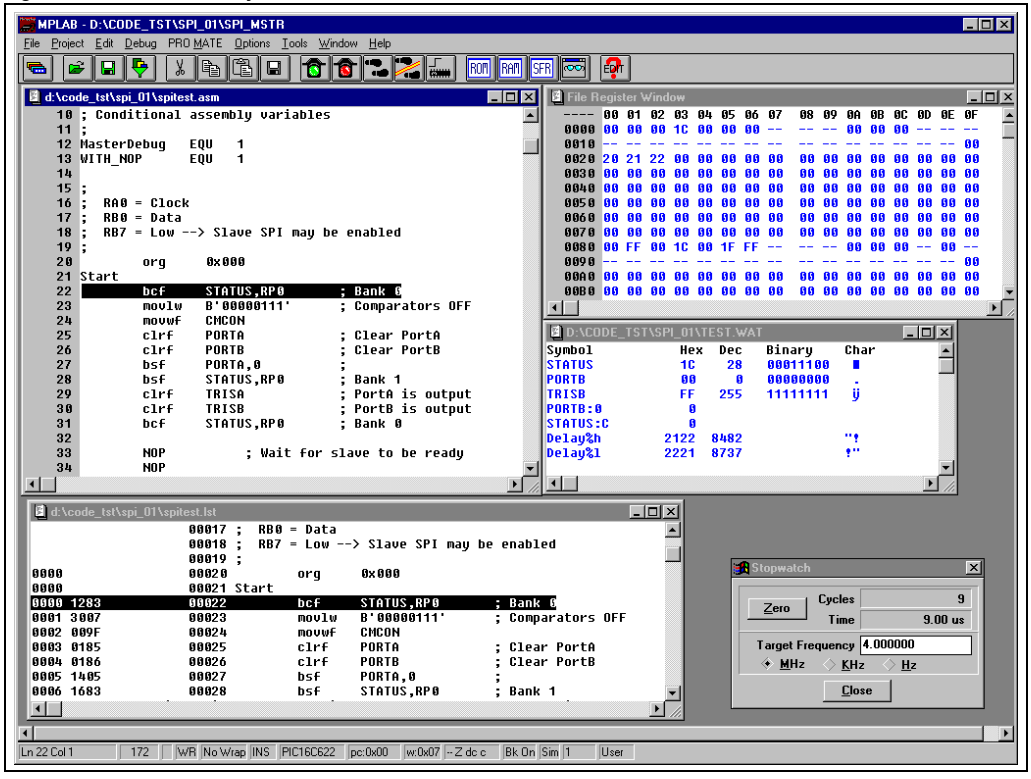
Microchip's simulator, MPLAB-SIM, operates under the same platform as the MPLAB-ICE emulator. This allows the user to learn a single tool set which functions equivalently for both the simulator and the full featured emulator.

Figure 34-1 shows a typical MPLAB desktop with an open project. Some of the highlights are:

- Tool bars, multiple choices and user configurable
- Status, mode information, and button help on footer bar
- Multiple windows, such as
  - Source code
  - Source listing (most useful for 'C' programs)
  - Register file window (RAM)
  - Watch windows (to look at specific register)
  - Stop watch window for time/cycle calculations
- Programmer support (in this case PRO MATE pull down menu)

Note: This screen shot may not look exactly like the currently released MPLAB version.

Figure 34-1: MPLAB Project Window



34.3 MPLAB® Software Language Support

To make the PICmicro device operate as desired in the application, a software program needs to be written for the microcontroller. This software program needs to be written in one of the programming languages for the device. Currently MPLAB supports two of Microchip's language products:

- Microchip Assembler (MPASM)
- Microchip 'C' Compiler (MPLAB-CXX)
- Other language products that support Common Object Description (COD) may also work with MPLAB

34.3.1 Assembler (MPASM)

The MPASM Universal Macro Assembler is a PC-hosted symbolic assembler. It supports all Microchip microcontroller families.

MPASM offers full featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

MPASM allow full symbolic debugging from the Microchip Universal Emulator System (MPLAB-ICE).

MPASM has the following features to assist in developing software for specific use applications.

- Provides translation of Assembler source code to object code for all Microchip microcontrollers.
- Macro assembly capability.
- Produces all the files (Object, Listing, Symbol, and Special) required for symbolic debug with Microchip's emulator systems.
- Supports Hex (default), Decimal and Octal source and listing formats.

MPASM provides a rich directive language to support programming of the PICmicro. Directives are helpful in making the development of your assemble source code shorter and more maintainable.

34.3.2 C Compilers

The MPLAB-CXX is a complete 'C' compiler for Microchip's PICmicro family of microcontrollers. The compiler provides powerful integration capabilities and ease of use not found with other compilers.

For easier source level debugging, the compiler provides symbol information that is compatible with the MPLAB IDE memory display, Watch windows, and File register windows.

34.3.2.1 MPLAB-C17 C Compiler

The MPLAB-C17 Code Development System is a complete ANSI 'C' compiler and integrated development environment for Microchip's **PIC17CXXX** family of microcontrollers. This compiler provides powerful integration capabilities and ease of use not found with other compilers.

For easier source level debugging, the compiler provides symbol information that is compatible with the MPLAB IDE memory display.

34.3.2.2 MPLAB-C18 C Compiler

The MPLAB-C18 Code Development System is a complete ANSI 'C' compiler and integrated development environment for Microchip's **PIC18CXXX** family of microcontrollers. This compiler provides powerful integration capabilities and ease of use not found with other compilers.

For easier source level debugging, the compiler provides symbol information that is compatible with the MPLAB IDE memory display.

34.3.3 MPLINK Linker

MPLINK is a linker for the Microchip C compiler, MPLAB-CXX, and the Microchip relocatable assembler, MPASM. MPLINK is a relocatable linker for MPASM and MPLAB-C17 and MPLAB-C18. It can link relocatable objects from assembly or C source files along with precompiled libraries using directives from a linker script.

MPLINK allows you to produce modular, re-usable code with MPLAB-CXX and MPASM. Control over the linking process is accomplished through a linker "script" file and with command line options. MPLINK ensures that all symbolic references are resolved and that code and data fit into the available PICmicro device.

MPLINK combines multiple input object modules generated by MPLAB-CXX or MPASM, into a single executable file. The actual addresses of data and the location of functions will be assigned when MPLINK is executed. This means that you will instruct MPLINK to place code and data somewhere within the named regions of memory, not to specific physical locations.

Once the linker knows about the ROM and RAM memory regions available in the target PICmicro device and it analyzes all the input files, it will try to fit the application's routines into ROM and assign it's data variables into available RAM. If there is too much code or too many variables to fit, MPLINK will give an error message.

MPLINK also provides flexibility for specifying that certain blocks of data memory are re-usable, so that different routines (which never call each other and don't depend on this data to be retained between execution) can share limited RAM space.

MPLINK features include:

- MPLINK works with MPASM and MPLAB-C17 and MPLAB-C18.
- MPLINK allows all memory areas to be defined as sections to provide link-time flexibility.

34.3.4 MPLIB Librarian

MPLIB is a librarian for use with COFF object modules, created using either MPASM, MPASMWIN, or MPLAB-CXX or later. MPLIB is a librarian for pre-compiled code to be used with MPLINK. When a routine from a library is called from another source file, only the modules that contain that routine will be linked in with the application. This allows large libraries to be used efficiently in many different applications.

MPLIB manages the creation and modification of library files. A library file is a collection of object modules that are stored in a single file. There are several reasons for creating library files:

- Libraries make linking easier. Since library files can contain many object files, the name of a library file can be used instead of the names of many separate object files when linking.
- Libraries help keep code small. Since a linker only uses the required object files contained in a library, not all object files which are contained in the library necessarily wind up in the linker's output module.
- Libraries make projects more maintainable. If a library is included in a project, the addition or removal of calls to that library will not require a change to the link process.
- Libraries help convey the purpose of a group of object modules. Since libraries can group together several related object modules, the purpose of a library file is usually more understandable that the purpose of its individual object modules. For example, the purpose of a file named "math.lib" is more apparent that the purpose of 'power.o', 'ceiling.o', and 'floor.o'.

34.4 MPLAB-SIM Simulator Software

The software simulator is a no-cost tool with which to evaluate Microchip's products and designs. The use of the simulator greatly helps debug software, particularly algorithms. Depending on the complexity of a design project, a time/cost benefit should be looked at comparing the simulator with an emulator.

For projects that have multiple engineers in the development, the simulator in conjunction with an emulator, can keep costs down and will allow speedy debug of the tough problems.

MPLAB-SIM Simulator simulates the PICmicro series microcontrollers on an instruction level. On any given instruction, the user may examine or modify any of the data areas or provide external stimulus to any of the pins. The input/output radix can be set by the user and the execution can be performed in; single step, execute until break, or in a trace mode.

MPLAB-SIM supports symbolic debugging using MPLAB-CXX, and MPASM. The Software Simulator offers the low cost flexibility to develop and debug code outside of the laboratory environment making it an excellent multi-project software development tool.

34.5 MPLAB® Emulator Hardware Support

Microchip offers two emulators, a high-end version (MPLAB-ICE) and a low-cost version (ICEPIC). Both versions offer a very good price/feature value, and the selection of which emulator should depend on the feature set that you wish. For people looking at doing several projects with Microchip devices (or using the high-end devices), the use of MPLAB-ICE may offset the additional investment, through time savings achieved with the sophisticated breakpoint and trace capabilities.

34.6 MPLAB® High Performance Universal In-Circuit Emulator with MPLAB IDE

The MPLAB-ICE Universal In-Circuit Emulator is intended to provide the product development engineer with a complete microcontroller design tool set for PICmicro microcontrollers (MCUs). Software control of MPLAB-ICE is provided by the MPLAB Integrated Development Environment (IDE), which allows editing, “make” and download, and source debugging from a single environment.

Interchangeable target probes allow the system to be easily re-configured for emulation of different processors. The universal architecture of the MPLAB-ICE allows expansion to support all new Microchip microcontrollers.

The MPLAB-ICE Emulator System has been designed as a real-time emulation system with advanced features that are generally found on more expensive development tools.

A CE compliant version of MPLAB-ICE is available for European Union (EU) countries.

34.6.1 ICEPIC: Low-Cost PIC16CXXX In-Circuit Emulator

ICEPIC is a low-cost in-circuit emulator solution for the Microchip Base-line and Mid-Range families of 8-bit OTP microcontrollers.

ICEPIC features real-time emulation. ICEPIC is available under the MPLAB environment.

ICEPIC is designed by Neosoft Inc. and is manufactured under license by RF Solutions. Other emulator solutions may be available directly from RF solutions.

34.7 MPLAB-ICD In-Circuit Debugger

Microchip’s In-Circuit Debugger, MPLAB-ICD, is a powerful, low-cost run-time development tool. This tool is based on the FLASH PIC16F877 and can be used to develop for this and other PICmicro microcontrollers from the PIC16CXXX family (see TB033 for more information). MPLAB-ICD utilizes the In-Circuit Debugging capability built into the PIC16F87X. This feature, along with Microchip’s In-Circuit Serial Programming protocol, offers cost-effective in-circuit FLASH programming and debugging from the graphical user interface of the MPLAB Integrated Development Environment. This enables a designer to develop and debug source code by watching variables, single-stepping and setting break points. Running at full speed enables testing hardware in real-time. The MPLAB-ICD is also a programmer for the flash PIC16F87X family.

34.8 MPLAB Programmer Support

Microchip offers two levels of device programmer support. For most bench setups the PICSTART Plus is sufficient. When true system qualification is done, the PRO MATE II should be the minimum used, due to the validation of program memory at VDD min and VDD max for maximum reliability

34.8.1 PRO MATE® II: Universal Device Programmer

The PRO MATE II Universal Programmer is a full-featured programmer capable of operating in stand-alone mode as well as PC-hosted mode. PRO MATE II operates under MPLAB or as a DOS command driven program.

The PRO MATE II has programmable VDD and VPP supplies which allows it to verify programmed memory at VDD min and VDD max for maximum reliability. It has an LCD display for error messages, keys to enter commands and a modular detachable socket assembly to support various package types. In stand-alone mode, the PRO MATE II can read, verify, or program Base-Line, Mid-Range, and High-End devices. It can also set configuration and code-protect bits in this mode. The PRO MATE II programmer also supports Microchip's Serial EEPROM and KEELOQ® Security devices.

A separate In-Circuit Serial Programming (ICSP) module is available for volume programming in a manufacturing environment. See the Programming module documentation for specific application requirements.

34.8.2 PICSTART® Plus Low-Cost Development Kit

The PICSTART Plus programmer is an easy-to-use, low-cost development programmer. It connects to the PC via one of the COM (RS-232) ports. MPLAB Integrated Development Environment software makes using the programmer simple and efficient. PICSTART Plus is not recommended for production programming, since it does not perform memory verification at VDDMIN and VDDMAX.

PICSTART Plus supports all Base-Line, Mid-Range, and High-End devices. For devices with up to more than 40 pins, an adapter socket is required. DIP packages are the form factor that are directly supported. Other package types may be supported with adapter sockets.

Note:	The use of a PICSTART Plus Programmer is not recommended for ICSP. If ICSP is required, the use of the PRO MATE II Universal Programmer with the associated socket module is the recommended Microchip solution.

34.9 Supplemental Tools

Microchip endeavors to provide a broad range of solutions to our customers. These tools are considered supplemental tools and may be available directly from Microchip or from another vendor. A comprehensive listing of alternate tool providers is contained in the Third Party Guide.

34.9.1 Third Party Guide

Looking for something else? Microchip strongly encourages and supports it's Third Parties. Microchip publishes the “Third Party Guide”. It is an extensive volume that provides:

- Company
- Product
- Contact Information
- Consultants

For over 100 companies and 200 products. These products include Emulators, Device Programmers, Gang Programmers, Language Products, and other tool solutions.

34.10 Development Boards

Development boards give a quick start on a circuit that demonstrates the capabilities of a particular device. The device program can then be modified for your own evaluation of the device functionality and operation.

34.10.1 PICDEM-1 Low-Cost PIC16/17 Demonstration Board

The PICDEM-1 is a simple board which demonstrates the capabilities of several of Microchip's microcontrollers. The microcontrollers supported are:

- PIC16C5X (PIC16C54 to PIC16C58A)
- PIC16C61
- PIC16C62X
- PIC16C71
- PIC16C710
- PIC16C711
- PIC16C8X
- PIC17C42A
- PIC17C43
- PIC17C44

All necessary hardware and software is included to run basic demo programs. The users can program the sample microcontrollers provided with the PICDEM-1 board, on a PRO MATE II or PICSTART Plus programmer, and easily test firmware. The user can also connect the PICDEM-1 board to the MPLAB-ICE emulator and download the firmware to the emulator for testing. Additional prototype area is available to build additional hardware. Some of the features include an RS-232 interface, a potentiometer for simulated analog input, push-button switches and eight LEDs connected to PORTB.

34.10.2 PICDEM-2 Low-Cost PIC16CXXX Demonstration Board

The PICDEM-2 is a simple demonstration board that supports the following microcontrollers:

- PIC16C62
- PIC16C63
- PIC16C64
- PIC16C65
- PIC16C66
- PIC16C67
- PIC16C72
- PIC16C73
- PIC16C74
- PIC16C76
- PIC16C77

All the necessary hardware and software is included to run the basic demonstration programs. The user can program the sample microcontrollers provided with the PICDEM-2 board, on a PRO MATE II programmer or PICSTART Plus, and easily test firmware. The MPLAB-ICE emulator may also be used with the PICDEM-2 board to test firmware. Additional prototype area has been provided for additional hardware. Some of the features include a RS-232 interface, push-button switches, a potentiometer for simulated analog input, a Serial EEPROM to demonstrate usage of the I^2C bus and separate headers for connection to an LCD module and a keypad.

34.10.3 PICDEM-3 Low-Cost PIC16CXXX Demonstration Board

The PICDEM-3 is a simple demonstration board that supports the PIC16C923 and PIC16C924 in the PLCC package. It will also support future 44-pin PLCC microcontrollers that have an LCD Module. All the necessary hardware and software is included to run the basic demonstration programs. The user can program the sample microcontrollers, provided with the PICDEM-3 board, on a PRO MATE II programmer or PICSTART Plus with an adapter socket, and easily test firmware. The MPLAB-ICE emulator may also be used with the PICDEM-3 board to test firmware. Additional prototype area has been provided for adding hardware. Some of the features include an RS-232 interface, push-button switches, a potentiometer for simulated analog input, a thermistor and separate headers for connection to an external LCD module and a keypad. Also provided on the PICDEM-3 board is an LCD panel, with 4 commons and 12 segments, that is capable of displaying time, temperature and day of the week. The PICDEM-3 provides an additional RS-232 interface and Windows 3.1 software for showing the de-multiplexed LCD signals on a PC. A simple serial interface allows the user to construct a hardware de-multiplexer for the LCD signals.

34.10.4 PICDEM-14A Low-Cost PIC14C000 Demonstration Board

The PICDEM-14A demo board is a general purpose platform which is provided to help evaluate the PIC14C000 mixed signal microcontroller. The board runs a PIC14C000 measuring the voltage of a potentiometer and the on-chip temperature sensor. The voltages are then calibrated to the internal bandgap voltage reference. The voltage and temperature data are then transmitted to the RS-232 port. This data can be displayed using a terminal emulation program, such as Windows Terminal. This demo board also includes peripherals that allow users to display data on an LCD panel, read from and write to a serial EEPROM, and prototype custom circuitry to interface to the microcontroller.

34.10.5 PICDEM-17

The PICDEM-17 is an evaluation board that demonstrates the capabilities of several Microchip microcontrollers, including PIC17C752, PIC17C756, PIC17C762, and PIC17C766. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5-inch disk. A programmed sample is included, and the user may erase it and program it with the other sample programs using the PRO MATE II or PICSTART Plus device programmers and easily debug and test sample code. In addition, PICDEM-17 supports down-loading of programs to and executing out of external FLASH memory on board. The PICDEM-17 is also usable with the MPLAB-ICE emulator, and all of the sample programs can be run and modified using that emulator. Additionally, a generous prototype area is available for user hardware.

34.11 Development Tools for Other Microchip Products

34.11.1 SEEVAL® Evaluation and Programming System

The SEEVAL Serial EEPROM Designer's Kit supports all Microchip 2-wire and 3-wire Serial EEPROMs. The kit includes everything necessary to read, write, erase or program special features of any Microchip SEEPROM product including Smart Serials™ and secure serials. The Total Endurance™ Disk is included to aid in trade-off analysis and reliability calculations. The total endurance kit can significantly reduce time-to-market and results in a more optimized system.

34.11.2 KEELOQ® Evaluation and Programming Tools

KEELOQ evaluation and programming tools supports Microchip's HCS Secure Data Products. The HCS evaluation kit includes an LCD display to show changing codes, a decoder to decode transmissions, and a programming interface to program test transmitters.

34.12 Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the Enhanced MCU family (that is they may be written for the Base-Line, Mid-Range, or the High-End), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to Microchip's development tools are:

Title	Application Note #
No related application notes at this time.	

Note: Several of the tools have tutorials which may be helpful in learning the tools. Please refer to the specific tool's documentation for tutorial availability.

Note: Please visit the Microchip Web site for additional software code examples. These code examples are stand alone examples to assist in the understanding of the PIC18CXXX. The web address for these examples is:

http://www.microchip.com/10/faqs/codeex/

34.13 Revision History

Revision A

This is the initial released revision of Microchip's development tools description.

Section 35. Code Development

HIGHLIGHTS

This section of the manual contains the following major topics:

35.1 Overview

This section covers some good programming practice as well as some diagnostic techniques that can be used in both the development stage, and the production release to help diagnose unexpected operation in the field. The advantage of including diagnostic code is that the device continually displays its status and the operational flow of the program.

These suggestions only scratch the surface of the possible techniques and good programming practices. Future revisions of this section will add additional suggestions on good programming practice and diagnostic techniques.

35.2 Good Practice

The following are some recommended programming practices. These will help ensure consistent program operation based on the stimulus supplied to the application software.

35.2.1 Use of Symbolic Code

Microchip supplies header files which use the register and bit name symbols specified in the device data sheets. The use of these symbols aids in several facets of software development. First, symbolic code makes the source file easier to read since the names relate to a function (as opposed to register addresses and bit positions). Example 35-1 shows two implementations. The first, though technically correct, is more difficult to follow then the second implementation. Also, the second allows for easier migration between devices (and PICmicro families) since any remapping of bit positions and register locations can automatically be handled by the software tool.

Example 35-1: Hard Coding vs. Symbolic Code

```
BCF     0xD8, 0, 0          ; Clear the carry bit
:                           ;
BCF     STATUS, C           ; Clear the carry bit
```

35.2.2 Initialization of Data Memory

All Data Memory locations (SFRs and GPRs) should be initialized after a RESET. This will ensure that they are at a known state when the application code accesses each location, and ensures consistent operation

When all data memory is not initialized, there is a possibility that the application software will read a location that is an indeterminate value. This may cause unexpected results from the application software. This issue is sometimes highlighted when development moves from a windowed device (where the window was not covered) to an OTP package.

35.2.3 Trap for Unexpected Conditions

Sometimes the application software will test only for the expected conditions. This is typically seen in the Interrupt Service Routine where only the expected sources for an interrupt are tested for. Example 35-2 shows a typical way that the conditions are tested, while Example 35-3 shows a more robust version. The TrapRoutine may do more then just loop waiting for a RESET. It could output a code on an I/O pin to indicate that this unexpected condition occurred.

Example 35-2: Typical ISR Code

```
        ORG     ISRVectorAddress  ; Address for Interrupt Service
                                  ;          Routine
ISRH    BTFSS   PIR1, ADIF        ; A/D Interrupt?
        GOTO    ADRoutine         ; Yes, do A/D stuff
        BTFSS   PIR1, RCIF        ; AUSART Receive Interrupt?
        GOTO    ReceiveRoutine    ; Yes, do Receive stuff
CCPRoutine                        ; Since not other interrupt sources
        :                         ;   must be CCP interrupt
```

Example 35-3: Recommended ISR Code (with Trap)

```
        ORG     ISRVectorAddress  ; Address for Interrupt Service
                                  ;          Routine
ISRH    BTFSS   PIR1, ADIF        ; A/D Interrupt?
        GOTO    ADRoutine         ; Yes, do A/D stuff
        BTFSS   PIR1, RCIF        ; AUSART Receive Interrupt?
        GOTO    ReceiveRoutine    ; Yes, do Receive stuff
        BTFSS   PIR1, CCPIF       ; CCP Interrupt?
        GOTO    CCPRoutine        ; Yes, do CCP stuff
TrapRoutine                       ; Should NEVER get here
        GOTO    TrapRoutine       ;   If we do, loop forever and
                                  ;   wait for WDT reset
```

35.2.4 Filling Unused Locations

In most application programs, not all program memory locations are used. These unprogrammed locations (0xFFFF) execute as a NOP instruction. Since program execution is never intended to go to one of these locations, the program should trap any occurrence of this. A good way to handle this is to “fill” all unprogrammed locations with a branch which goes to its own program memory location. With this, the WDT must be enabled to cause a device RESET and restart program execution.

Note:	Remember that the CLRWDT instruction should have a minimum number of occurrences in the program and that the maximum time between the CLRWDT instructions must NOT be greater than the minimum WDT time-out period, multiplied by the selected prescaler.

The Microchip Assembler supplies a directive to do this. This directive is the “fill” directive.

35.3 Diagnostic Code Techniques

This section describes diagnostic code that can be embedded into the application code to help track the operation of the application software. Typically some mechanism is required to make the information available to the external world. This can be from a serial port (such as the Addressable USART module) or from extra I/O pins not used by the application.

35.3.1 Function Sequence

Have the diagnostic code output a unique code (value) for each function that the program enters. This allows the flow of the program to be monitored from outside the device (with a logic analyzer) and assist in determining if there is some unexpected condition that is causing code to execute in the observed sequence.

35.3.2 Stack Depth

A counter can be implemented that is incremented in each function that is called and each time that the program counter is at an interrupt vector address. The counter is then decremented at the end of the function or interrupt service routine. The value of the stack can then be output to indicate if the stack depth is out of its normal range (minimum and maximum).

35.3.3 A/D Operation

Visibility into the operation of the A/D can help validate the operation. Monitoring if the acquisition time is the expected time delay, as well as the result generated by the module. Achieving the A/D result accuracy depends on may factors, some internal and many external. The major internal factor is that there is a sufficient amount of time once the input channel is selected, until the conversion is started. This is called the acquisition time (TACQ).

35.3.3.1 A/D Acquisition Time

Use an I/O pin to indicate when the acquisition of the channel starts (toggle high) until the conversion is started (toggle high). If the input channel is changed force the I/O to toggle high. The last two I/O toggles before the A/D result is available, is acquisition time that the A/D had for that selected channel.

35.3.3.2 A/D Result

Output the result so that you can determine if the program execution flow can be attributed to the result from the A/D converter.

35.4 Example Scenario and Implementation

In this example application, one I/O port is not implemented. This means that there are eight I/O pins available to indicate the operational status of the PICmicro device and the flow of the application software. Let's say that the code implements 57 functions, the A/D is used and Addressable USART are used.

Six bits (Code5:Code0) are required to uniquely specify the 57 functions. After each function gets a code, an additional 7 codes are available. Table 35-1 shows how these codes could be defined, while Table 35-2 shows how the diagnostic output port could be defined.

Table 35-1: Definitions of Additional Codes

6-bit Code	Definition
1111 11	Next Byte out is stack depth
1111 10	Next Byte out is A/D Result High Byte
1111 01	Next Byte out is Addressable USART received byte
1111 00	Next Byte out is A/D Result Low Byte
0000 00	Device has exited a device RESET. Rx1:Rx0 can be used to indicate the source of the RESET.

Table 35-2: PORT Function assignment

Pin	Normal Mode	Mode after "Additional Codes" displayed	RESET
Rx7	Code5	Byte value for the indicated function	0
Rx6	Code4		0
Rx5	Code3		0
Rx4	Code2		0
Rx3	Code1		0
Rx2	Code0		0
Rx1	Toggle for Start of acquisition		4 codes available to indicate source of RESET
Rx0	Toggle for Stop of acquisition		

So, as the device exits RESET, the PORT = '1111 11xx' where the xx indicates the source of the device RESET. Then the code for the main function is output on Rx7:Rx2 and Rx1:Rx0 = '00'. As each function gets called, the stack depth is output. This requires that the code Rx7:Rx2 = '1111 11' is output, then the stack depth counter is moved to the PORT. The code for the entered function is then output.

After the A/D is enabled and acquisition of the selected channel begins, the Rx1 pin is toggled high. Each event that causes the input to start a new acquisition time should cause the Rx1 pin to toggle. These events include:

- Selection of a new input channel
- The completion of an A/D conversion
- Disabling and then re-enabling the A/D module

When an A/D conversion completes or the addressable USART receives a byte, the appropriate code is output on the PORT and then the value is output.

35.5 Implications of Using a High Level Language (HLL)

The use of a High Level Language, such as a C compiler, speeds the development cycle but can increase the difficulty in the use of diagnostic code. When writing at the high level, functions may be defined, but how the C compilers will implement those functions is hidden from the user. If the function is used only once, the C complier may put that code in-line or use a GOTO to branch to the function and a GOTO to return from the function. In both these implementations, the stack is not affected. Also, if the function is called many times, but is small, it may be put inline. This again will not affect the stack used. These techniques used by the compiler may lead to efficient code generation, but may add to the difficulty in adding diagnostic code.

35.6 Revision History

Revision A

This is the initial released revision for the Code Development with a PICmicro device description.

Section 36. Appendix

APPENDIX A: I²C™ OVERVIEW

This appendix provides an overview of the Inter-Integrated Circuit (I²C™) bus, with Subsection **A.2 "Addressing I²C Devices"** discussing the operation of the SSP modules in I²C mode.

The I²C bus is a two-wire serial interface. The original specification, or standard mode, is for data transfers of up to 100 Kbps. An enhanced specification, or fast mode (400 Kbps) is supported. Standard and Fast mode devices will operate when attached to the same bus, if the bus operates at the speed of the slower device.

The I²C interface employs a comprehensive protocol to ensure reliable transmission and reception of data. When transmitting data, one device is the "master" which initiates transfer on the bus and generates the clock signals to permit that transfer, while the other device(s) acts as the "slave." All portions of the slave protocol are implemented in the SSP module's hardware, except general call support, while portions of the master protocol need to be addressed in the PIC16CXX software. The MSSP module supports the full implementation of the I²C master protocol, the general call address, and data transfers up to 1 Mbps. The 1 Mbps data transfers are supported by some of Microchips Serial EEPROMs. Table A-1 defines some of the I²C bus terminology.

In the I²C interface protocol each device has an address. When a master wishes to initiate a data transfer, it first transmits the address of the device that it wishes to "talk" to. All devices "listen" to see if this is their address. Within this address, a bit specifies if the master wishes to read-from/write-to the slave device. The master and slave are always in opposite modes (transmitter/receiver) of operation during a data transfer. That is, they can be thought of as operating in either of these two relations:

- Master-transmitter and Slave-receiver
- Slave-transmitter and Master-receiver

In both cases the master generates the clock signal.

The output stages of the clock (SCL) and data (SDA) lines must have an open-drain or open-collector in order to perform the wired-AND function of the bus. External pull-up resistors are used to ensure a high level when no device is pulling the line down. The number of devices that may be attached to the I²C bus is limited only by the maximum bus loading specification of 400 pF and addressing capability.

I²C is a trademark of Philips Corporation.

A.1 Initiating and Terminating Data Transfer

During times of no data transfer (idle time), both the clock line (SCL) and the data line (SDA) are pulled high through the external pull-up resistors. The START and STOP conditions determine the start and stop of data transmission. The START condition is defined as a high to low transition of the SDA when the SCL is high. The STOP condition is defined as a low to high transition of the SDA when the SCL is high. Figure A-1 shows the START and STOP conditions. The master generates these conditions for starting and terminating data transfer. Due to the definition of the START and STOP conditions, when data is being transmitted, the SDA line can only change state when the SCL line is low.

Figure A-1: Start and Stop Conditions

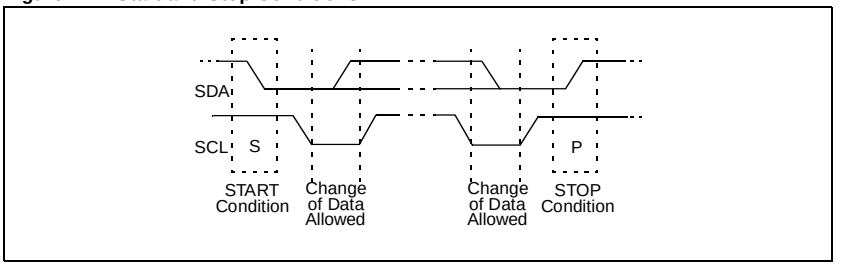


Table A-1: I^2C Bus Terminology

Term	Description
Transmitter	The device that sends the data to the bus.
Receiver	The device that receives the data from the bus.
Master	The device which initiates the transfer, generates the clock and terminates the transfer.
Slave	The device addressed by a master.
Multi-master	More than one master device in a system. These masters can attempt to control the bus at the same time without corrupting the message.
Arbitration	Procedure that ensures that only one of the master devices will control the bus. This ensure that the transfer data does not get corrupted.
Synchronization	Procedure where the clock signals of two or more devices are synchronized.

A.2 Addressing I²C Devices

There are two address formats. The simplest is the 7-bit address format with a R/$\overline{W}$ bit (Figure A-2). The more complex is the 10-bit address with a R/$\overline{W}$ bit (Figure A-3). For 10-bit address format, two bytes must be transmitted. The first five bits specify this to be a 10-bit address format. The 1st transmitted byte has 5 bits which specify a 10-bit address, the two MSbs of the address, and the R/$\overline{W}$ bit. The second byte is the remaining 8 bits of the address.

Figure A-2: 7-bit Address Format

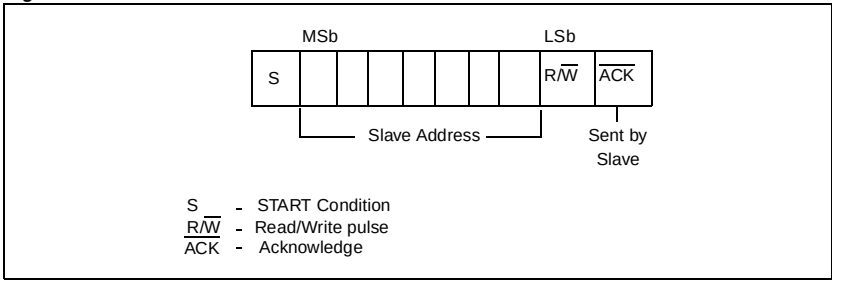
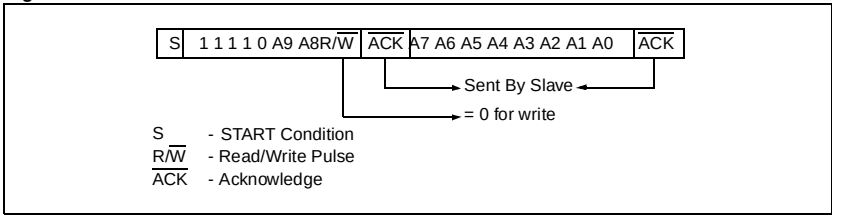


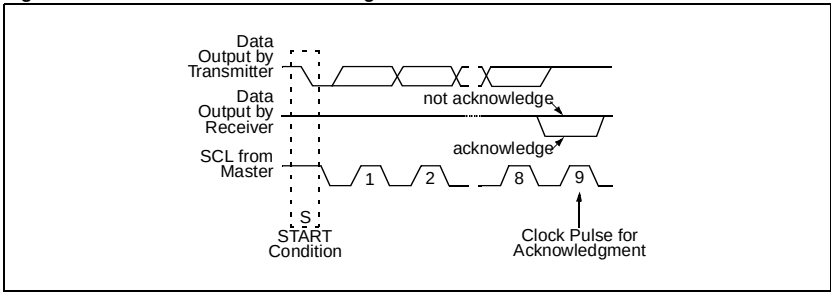
Figure A-3: I²C 10-bit Address Format



A.3 Transfer Acknowledge

All data must be transmitted per byte, with no limit to the number of bytes transmitted per data transfer. After each byte, the slave-receiver generates an acknowledge bit ($\overline{ACK}$) (Figure A-4). When a slave-receiver doesn't acknowledge the slave address or received data, the master must abort the transfer. The slave must leave SDA high so that the master can generate the STOP condition (Figure A-1).

Figure A-4: Slave-Receiver Acknowledge



If the master is receiving the data (master-receiver), it generates an acknowledge signal for each received byte of data, except for the last byte. To signal the end of data to the slave-transmitter, the master does not generate an acknowledge (not acknowledge). The slave then releases the SDA line so the master can generate the STOP condition. The master can also generate the STOP condition during the acknowledge pulse for valid termination of data transfer.

If the slave needs to delay the transmission of the next byte, holding the SCL line low will force the master into a wait state. Data transfer continues when the slave releases the SCL line. This allows the slave to move the received data or fetch the data it needs to transfer before allowing the clock to start. This wait state technique can also be implemented at the bit level, Figure A-5.

Figure A-5: Data Transfer Wait State

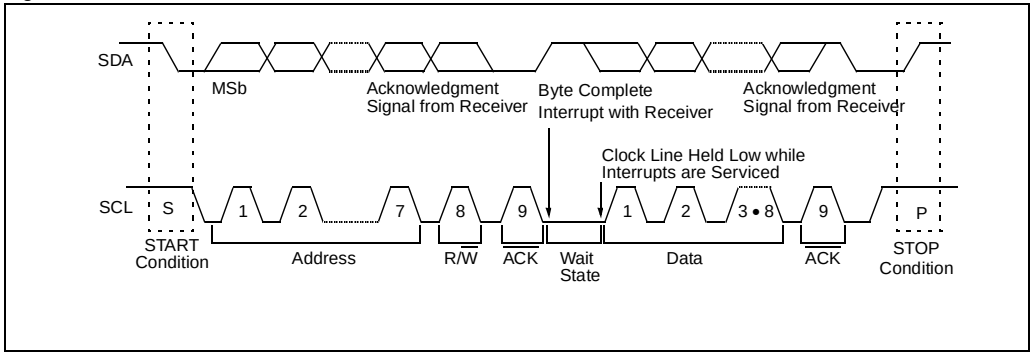


Figure A-6 and Figure A-7 show Master-transmitter and Master-receiver data transfer sequences.

Figure A-6: Master-Transmitter Sequence

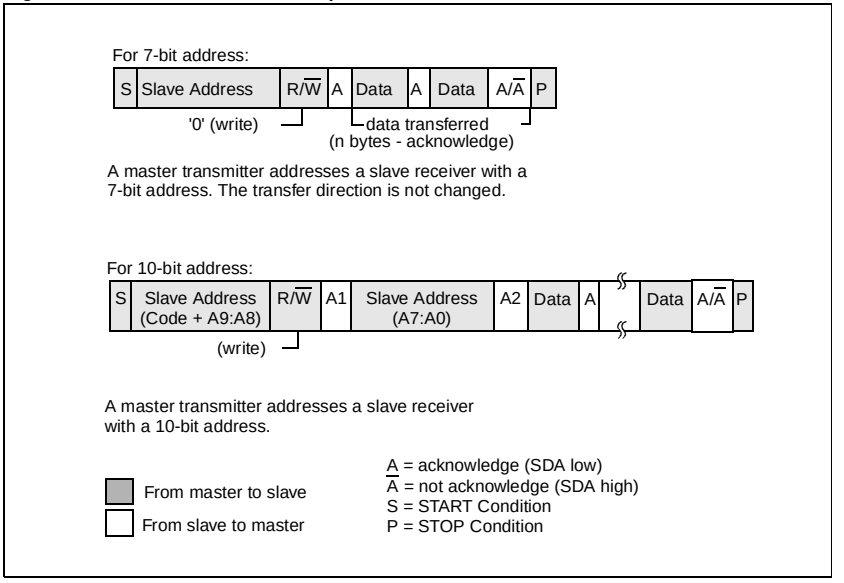
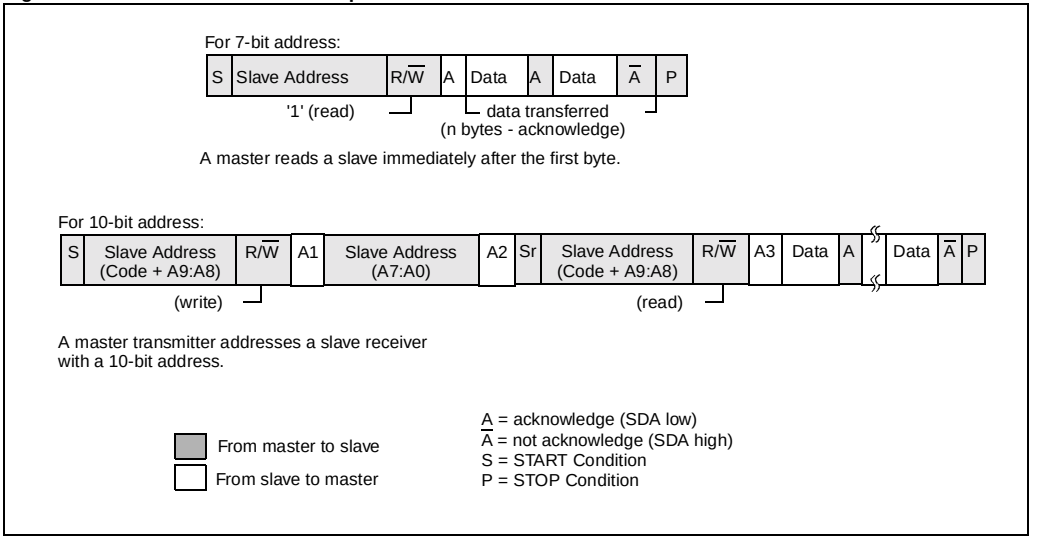
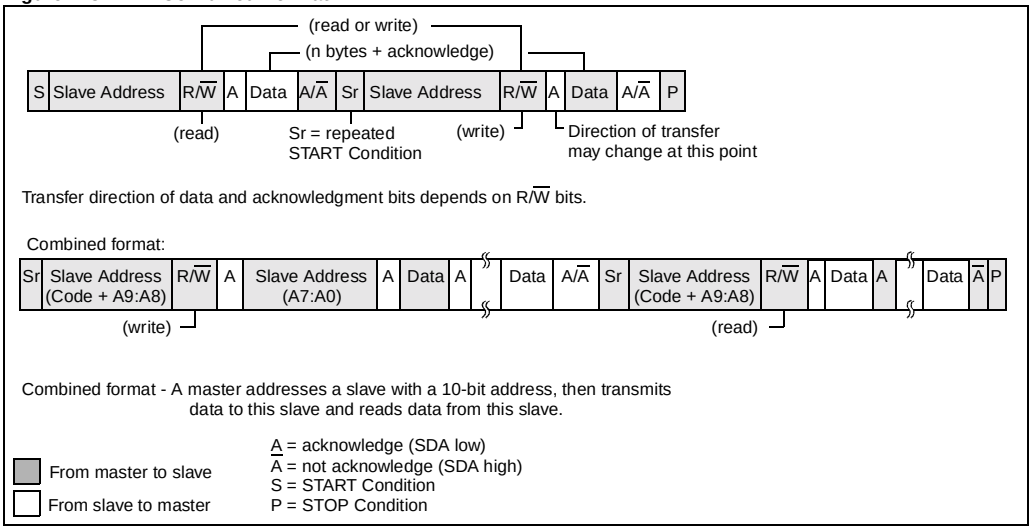


Figure A-7: Master-Receiver Sequence



When a master does not wish to relinquish the bus (which occurs by generating a STOP condition), a repeated START condition (Sr) must be generated. This condition is identical to the START condition (SDA goes high-to-low while SCL is high), but occurs after a data transfer acknowledge pulse (not the bus-free state). This allows a master to send “commands” to the slave and then receive the requested information or to address a different slave device. This sequence is shown in Figure A-8.

Figure A-8: Combined Format



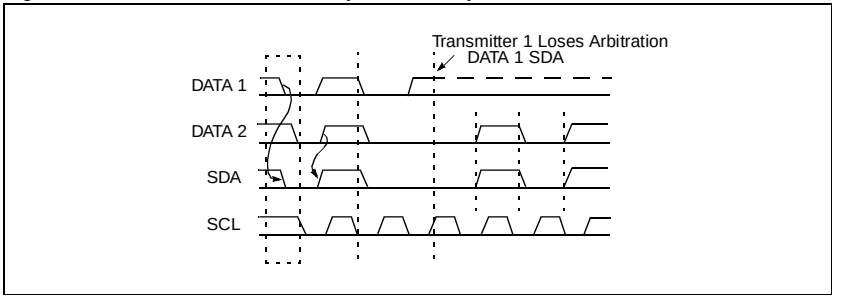
A.4 Multi-master

The I^2C protocol allows a system to have more than one master. This is called a multi-master system. When two or more masters try to transfer data at the same time, arbitration and synchronization occur.

A.4.1 Arbitration

Arbitration takes place on the SDA line, while the SCL line is high. The master which transmits a high when the other master transmits a low, loses arbitration (Figure A-9) and turns off its data output stage. A master which lost arbitration can generate clock pulses until the end of the data byte where it lost arbitration. When the master devices are addressing the same device, arbitration continues into the data.

Figure A-9: Multi-Master Arbitration (Two Masters)



Masters that also incorporate the slave function, and have lost arbitration must immediately switch over to slave-receiver mode. This is because the winning master-transmitter may be addressing it.

Arbitration is not allowed between:

- A repeated START condition
- A STOP condition and a data bit
- A repeated START condition and a STOP condition

Care needs to be taken to ensure that these conditions do not occur.

A.4.2 Clock Synchronization

Clock synchronization occurs after the devices have started arbitration. This is performed using a wired-AND connection to the SCL line. A high to low transition on the SCL line causes the concerned devices to start counting off their low period. Once a device clock has gone low, it will hold the SCL line low until its SCL high state is reached. The low to high transition of this clock may not change the state of the SCL line, if another device clock is still within its low period. The SCL line is held low by the device with the longest low period. Devices with shorter low periods enter a high wait-state, until the SCL line comes high. When the SCL line comes high, all devices start counting off their high periods. The first device to complete its high period will pull the SCL line low. The SCL line high time is determined by the device with the shortest high period, Figure A-10.

Figure A-10:Clock Synchronization

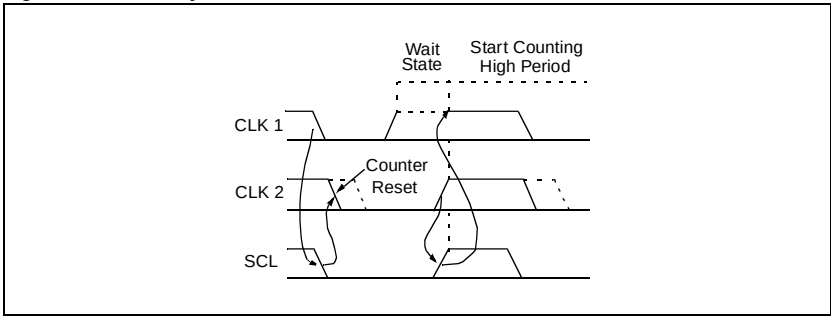


Table A-2 and Table A-3 show the specifications of a compliant I²C bus. The column titled Microchip Parameter No. is provided to ease the user's correlation to the corresponding parameter in the device data sheet. Figure A-11 and Figure A-12 show these times on the appropriate waveforms.

Figure A-11: I²C Bus Start/Stop Bits Timing Specification

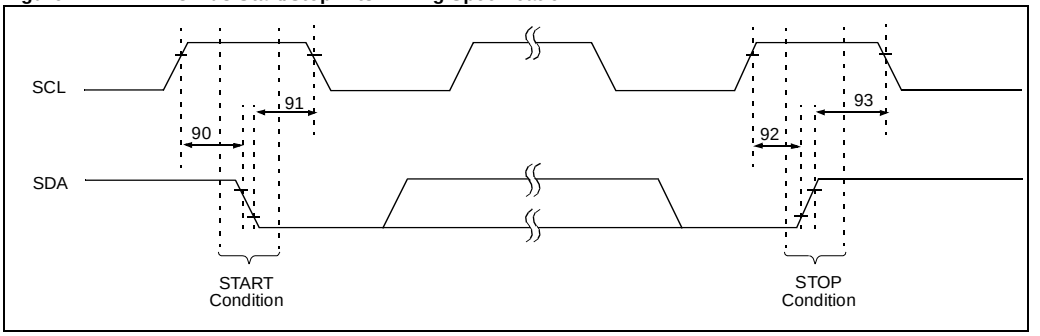


Table A-2: I²C Bus Start/Stop Bits Timing Specification

Microchip Parameter No.	Sym	Characteristic		Min	Typ	Max	Units	Conditions
90	TSU:STA	START condition Setup time	100 kHz mode	4700	—	—	ns	Only relevant for repeated START condition
			400 kHz mode	600	—	—		
91	THD:STA	START condition Hold time	100 kHz mode	4000	—	—	ns	After this period the first clock pulse is generated
			400 kHz mode	600	—	—		
92	TSU:STO	STOP condition Setup time	100 kHz mode	4700	—	—	ns	
			400 kHz mode	600	—	—		
93	THD:STO	STOP condition Hold time	100 kHz mode	4000	—	—	ns	
			400 kHz mode	600	—	—		

Figure A-12: I²C Bus Data Timing Specification

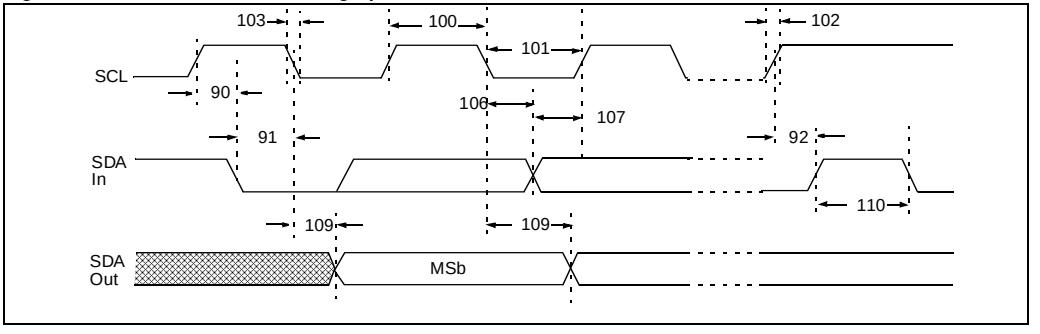


Table A-3: I^2C Bus Data Timing Specification

Microchip Parameter No.	Sym	Characteristic		Min	Max	Units	Conditions
100	THIGH	Clock high time	100 kHz mode	4.0	—	µs	
			400 kHz mode	0.6	—	µs	
101	TLOW	Clock low time	100 kHz mode	4.7	—	µs	
			400 kHz mode	1.3	—	µs	
102	TR	SDA and SCL rise time	100 kHz mode	—	1000	ns	
			400 kHz mode	20 + 0.1Cb	300	ns	Cb is specified to be from 10 to 400 pF
103	TF	SDA and SCL fall time	100 kHz mode	—	300	ns	
			400 kHz mode	20 + 0.1Cb	300	ns	Cb is specified to be from 10 to 400 pF
90	TSU:STA	START condition setup time	100 kHz mode	4.7	—	µs	Only relevant for repeated START condition
			400 kHz mode	0.6	—	µs	
91	THD:STA	START condition hold time	100 kHz mode	4.0	—	µs	After this period the first clock pulse is generated
			400 kHz mode	0.6	—	µs	
106	THD:DAT	Data input hold time	100 kHz mode	0	—	ns	
			400 kHz mode	0	0.9	µs	
107	TSU:DAT	Data input setup time	100 kHz mode	250	—	ns	Note 2
			400 kHz mode	100	—	ns	
92	TSU:STO	STOP condition setup time	100 kHz mode	4.7	—	µs	
			400 kHz mode	0.6	—	µs	
109	TAA	Output valid from clock	100 kHz mode	—	3500	ns	Note 1
			400 kHz mode	—	1000	ns	
110	TBUF	Bus free time	100 kHz mode	4.7	—	µs	Time the bus must be free before a new transmission can start
			400 kHz mode	1.3	—	µs	
D102	Cb	Bus capacitive loading		—	400	pF	

Note 1: As a transmitter, the device must provide this internal minimum delay time to bridge the undefined region (min. 300 ns) of the falling edge of SCL to avoid unintended generation of START or STOP conditions.

2: A fast-mode I^2C-bus device can be used in a standard-mode I^2C-bus system, but the requirement $T_{SU;DAT} \geq 250$ ns must then be met. This will automatically be the case if the device does not stretch the LOW period of the SCL signal. If such a device does stretch the LOW period of the SCL signal, it must output the next data bit to the SDA line
T_R max.+$T_{SU;DAT}$ = 1000 + 250 = 1250 ns (according to the standard-mode I^2C bus specification) before the SCL line is released.

APPENDIX B: CAN OVERVIEW

This appendix provides an overview of the Controller Area Network (CAN) bus. The CAN Section of this reference manual discusses the implementation of the CAN protocol in that hardware module.

Not available at this time.

APPENDIX C: MODULE BLOCK DIAGRAMS AND REGISTERS

This appendix summarizes the block diagrams of the major circuits.

Not available at this time.

APPENDIX D: REGISTER DEFINITIONS

This appendix summarizes the register definitions.

Not available at this time.

APPENDIX E: MIGRATION TIPS

This appendix gives an overview of some things that need to be taken into account when migrating code from other PICmicro families to the PIC18CXXX. For additional information, please also refer to the following application notes:

- AN716
- AN726

Three major changes Timer0 has from the mid-range family are:

1. The Timer0 no longer shares the prescaler with the Watchdog Timer.
2. Timer0 has 16-bit as well as 8-bit capability.
3. Timer0 can now be turned off.

The default state for Timer0 is an 8-bit counter for downward compatibility with the mid-range Timer0.

The prescaler is no longer shared between the Timer0 module and the Watchdog Timer. Each have their own separate prescalers.

Note:	The Timer0 prescaler is not shared with the watchdog as a postscaler. The Watchdog Timer has its own dedicated postscaler. The control bits and prescaler select bits are implemented as EPROM configuration bits.

Thus, a prescaler assignment for the Timer0 has no effect on the Watchdog Timer, and vice-versa.

T1OSCEN bit is now OSCEN.

Note:	To achieve a 1:1 prescaler assignment for the TMR0 register, turn off the TMR0 prescaler (PSA is cleared).

The program space is implemented as a single contiguous block, as compared to Microchip's mid-range and high-end controllers that divided the program memory into pages.

If interrupt priority is not used, all interrupts are treated as high priority, and function the same way as in mid-range and high-end controllers.

If interrupt priority is not used (all interrupt priority bits are set), GIE and PEIE/GIEL function the same as mid-range and high-end controllers.

Look-up tables are implemented two ways in the 18CXXX devices. The Computed Goto is compatible with the PIC16CXXX and PIC17CXXX parts. Code written for those devices will run on the PIC18CXXX devices with little or no change.

Table reads are implemented on the PIC17CXXX and PIC18CXXX devices. However, table operations on the PIC18CXXX work differently than on the PIC17CXXX.

E.1 Differences to the Mid-Range CCP module

The CCP1 and CCP2 modules function exactly as the modules of the mid-range devices with two exceptions:

1. The time base used for capture and compare of the CCP module can come from either Timer1 or Timer3. The default time base for both CCP modules when configured as a capture or compare mode is Timer1. Timer3 can be selected by configuring the Timer3 T3CCPx control bits in the Timer3 control register.
2. A toggle on compare mode has been added to the CCP modules.

E.1.1 Compatibility to PIC16CXX Interrupts

When the IPE bit in the RCON register is clear, the GIE/GIEH bit (INTCON<7>) is the global interrupt enable for all interrupts. The PEIE/GIEL bit (INTCON<6>) is still used to enable peripheral interrupts. The high priority interrupt vector is the same as the PIC16CXX interrupt vector and returns will work the same way. ISR code written for the PIC16CXX devices run unchanged as long as IPE=0.

E.2 Instruction Set Comparison

Table E-1 shows all the PIC18CXXX instructions and if that instruction is available in the other families (PIC17CXXX, PIC16CXX, or PIC16C5XX). Some instructions may be available, but have slightly different characteristics. Attached are notes regarding any major differences.

Table E-1: PIC18CXXX Instruction Set Comparison

BYTE-ORIENTED FILE REGISTER OPERATIONS			
PIC18CXXX Instruction Set	PIC17CXX	PIC16CXX	PIC16C5X
ADDWF	Yes	Yes	Yes
ADDWFC	Yes	—	—
ANDWF	Yes	Yes	Yes
CLRF	Yes (Note 1)	Yes	Yes
COMF	Yes	Yes	Yes
CPFSEQ	Yes	—	—
CPFSGT	Yes	—	—
CPFSLT	Yes	—	—
DAW	Yes (Note 5)	—	—
DCFSNZ	Yes	—	—
DECF	Yes	Yes	Yes
DECFSZ	Yes	Yes	Yes
INCF	Yes	Yes	Yes
INCFSZ	Yes	Yes	Yes
INFSNZ	Yes	—	—
IORWF	Yes	Yes	Yes
MOVF	—	Yes	Yes
MOVFF	—	—	—
MOVWF	Yes	Yes	Yes
MULWF	Yes	—	—
NEGF	Yes (Note 6)	—	—
NOP	Yes	Yes	Yes

Note 1: CLRF and SETF instructions do not have 's' bit that exists in the PIC17CXX instructions.

2: CLRW does not exist as an instruction, but since the WREG is mapped in address space, WREG can be cleared using the CLRF instruction.

3: The RLCF and RRCF instructions are functionally identical to the PIC16CXX instructions RLF and RRF.

4: CALL and GOTO instructions are now 2-word instructions.

5: DAW always has the WREG as the destination. The PIC17CXX can also have a file as the destination.

6: NEGW is replaced by NEGF with the file register always being the destination.

7: The mnemonics for 17CXX instructions TABLRD and TABLWT are changed to TBLRD and TBLWT, respectively, and these instructions are special instructions which only exchange data between the TABLAT and the program memory.

8: The lower nibble of the Bank Select Register (BSR) now specifies which bank of RAM is selected.

BYTE-ORIENTED FILE REGISTER OPERATIONS (Continued)			
Instruction Set	PIC17CXX	PIC16CXX	PIC16C5X
RLCF	Yes	Yes (Note 3)	Yes (Note 3)
RLNCF	Yes	—	—
RRCF	Yes	Yes (Note 3)	Yes (Note 3)
RRNCF	Yes	—	—
SETF	Yes (Note 1)	—	—
SUBFWB	—	—	—
SUBWF	Yes	Yes	Yes
SUBWFB	Yes	—	—
SWAPF	Yes	Yes	Yes
TABLRD	Yes (Note 7)	—	—
TABLWT	Yes (Note 7)	—	—
TSTFSZ	Yes	—	—
XORWF	Yes	Yes	Yes
BIT-ORIENTED FILE REGISTER OPERATIONS			
Instruction Set	PIC17CXX	PIC16CXX	PIC16C5X
BCF	Yes	Yes	Yes
BSF	Yes	Yes	Yes
BTFSC	Yes	Yes	Yes
BTFSS	Yes	Yes	Yes
BTG	Yes	—	—

Note 1: CLRF and SETF instructions do not have 's' bit that exists in the PIC17CXX instructions.

2: CLRW does not exist as an instruction, but since the WREG is mapped in address space, WREG can be cleared using the CLRF instruction.

3: The RLCF and RRCF instructions are functionally identical to the PIC16CXX instructions RLF and RRF.

4: CALL and GOTO instructions are now 2-word instructions.

5: DAW always has the WREG as the destination. The PIC17CXX can also have a file as the destination.

6: NEGW is replaced by NEGF with the file register always being the destination.

7: The mnemonics for 17CXX instructions TABLRD and TABLWT are changed to TBLRD and TBLWT, respectively, and these instructions are special instructions which only exchange data between the TABLAT and the program memory.

8: The lower nibble of the Bank Select Register (BSR) now specifies which bank of RAM is selected.

CONTROL OPERATIONS			
Instruction Set	PIC17CXX	PIC16CXX	PIC16C5X
BC	—	—	—
BN	—	—	—
BNC	—	—	—
BNN	—	—	—
BNV	—	—	—
BNZ	—	—	—
BRA	—	—	—
BV	—	—	—
BZ	—	—	—
CALL (Note 4)	Yes	Yes	Yes
CLRWDT	Yes	Yes	Yes
GOTO (Note 4)	Yes	Yes	Yes
POP	—	—	—
PUSH	—	—	—
RCALL	—	—	—
RESET	—	—	—
RETFIE	Yes	Yes	
RETURN	Yes	Yes	Yes
SLEEP	Yes	Yes	Yes
LITERAL OPERATIONS			
Instruction Set	PIC17CXX	PIC16CXX	PIC16C5X
ADDLW	Yes	Yes	Yes
ANDLW	Yes	Yes	Yes
IORLW	Yes	Yes	Yes
LFSR	—	—	—
MOVLB	Yes (Note 8)	—	—
MOVLW	Yes	Yes	Yes
MULLW	Yes	—	—
RETLW	Yes	Yes	Yes
SUBLW	Yes	Yes	Yes
XORLW	Yes	Yes	Yes

Note 1: CLRF and SETF instructions do not have 's' bit that exists in the PIC17CXX instructions.

2: CLRW does not exist as an instruction, but since the WREG is mapped in address space, WREG can be cleared using the CLRF instruction.

3: The RLCF and RRCF instructions are functionally identical to the PIC16CXX instructions RLF and RRF.

4: CALL and GOTO instructions are now 2-word instructions.

5: DAW always has the WREG as the destination. The PIC17CXX can also have a file as the destination.

6: NEGW is replaced by NEGF with the file register always being the destination.

7: The mnemonics for 17CXX instructions TABLRD and TABLWT are changed to TBLRD and TBLWT, respectively, and these instructions are special instructions which only exchange data between the TABLAT and the program memory.

8: The lower nibble of the Bank Select Register (BSR) now specifies which bank of RAM is selected.

Table E-2 shows the instructions that are used in the other PICmicro families, but are not used in the PIC18CXXX instruction set.

Table E-2: Instructions Not Implemented or Modified From PIC18CXXX

BYTE-ORIENTED FILE REGISTER OPERATIONS				
Instruction	PIC17CXX	PIC16CXX	PIC16C5X	Comment
CLRW	—	Yes (Note 1)	—	= CLRF WREG
DAW	Yes (Note 6)	—	—	
MOVFP	Yes (Note 7)	—	—	= MOVFF REG1, REG2
MOVPF	Yes (Note 7)	—	—	= MOVFF REG2, REG1
NEGW	Yes (Note 3)	—	—	
RLF	—	Yes (Note 4)	Yes (Note 4)	
RRF	—	Yes (Note 4)	Yes (Note 4)	
TLRD	Yes	—	—	
TLWT	Yes	—	—	
CONTROL OPERATIONS				
Instruction	PIC17CXX	PIC16CXX	PIC16C5X	
CALL	Yes (Note 2)	Yes (Note 2)	Yes (Note 2)	
GOTO	Yes (Note 2)	Yes (Note 2)	Yes (Note 2)	
LCALL	Yes	—	—	
MOVLR	Yes	—	—	
OPTION	—	— (Note 5)	Yes	
TRIS	—	— (Note 5)	Yes	
LITERAL OPERATIONS				
Instruction	PIC17CXX	PIC16CXX	PIC16C5X	
MOVLR	Yes	—	—	

Note 1: CLRW does not exist as an instruction, but since the WREG is mapped in address space, WREG can be cleared using the CLRF instruction.

2: CALL and GOTO instructions are now 2-word instructions.

3: NEGW is replaced by NEGF with the file register always being the destination.

4: The mnemonics for RLF and RRF have been changed to RLCF and RRCF, respectively, but the functionality is identical.

5: This instruction is not recommended in PIC16CXX devices.

6: The PIC17CXX may also specify the file as the destination.

7: When migrating software the MOVFF instruction can be used.

APPENDIX F: ASCII CHARACTER SET

The PICmicro assembler recognizes the ASCII characters shown in Table F-1.

Table F-1: ASCII Character Set (7-bit Code)

Least Significant nibble (LSn)	Most Significant nibble (MSn)							
	0 (0)	1 (16)	2 (32)	3 (48)	4 (64)	5 (80)	6 (96)	7 (112)
0	NUL	DLE	SP	0	@	P	’	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	—	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Note: To obtain the decimal value, add the decimal most significant nibble (MSn) to the decimal least significant nibble (LSn).

APPENDIX G: COMMON SOCKET PINOUTS

This appendix shows the pinouts for some of the standard sockets that are commonly used in prototype and production applications. You may use these pinouts when you wiretap your breadboard with a socket. These diagrams will make constructing, debugging, and troubleshooting with the Picmicro families quicker and easier.

These figures shown in this appendix are for sockets that correspond to the following package types:

PLCC-to-PGA sockets

- 28-pin PLCC/CLCC
- 44-pin PLCC/CLCC
- 68-pin PLCC/CLCC

DIP sockets

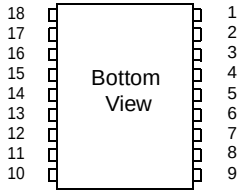
- 18-pin
- 28-pin
- 40-pin

SOIC sockets

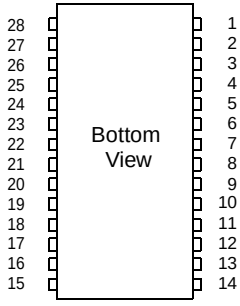
- 18-pin
- 28-pin

Figure G-1: DIP Packages Pinouts (Bottom View)

18-pin



28-pin



40-pin

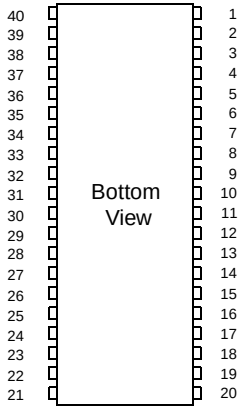
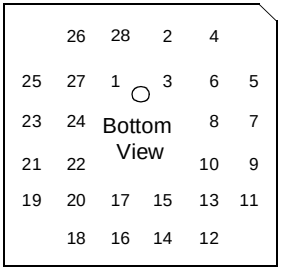


Figure G-2: PGA Package Pinout (Bottom View) for LCC Packages

28-pin



44-pin

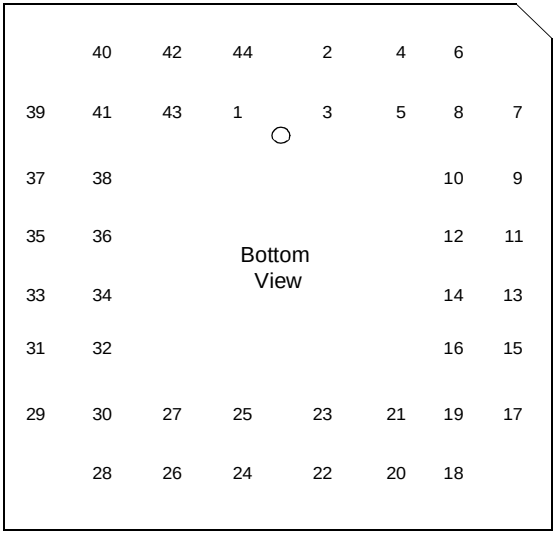


Figure G-2: PGA Package Pinout (Bottom View) for LCC Packages (Continued)

68-pin

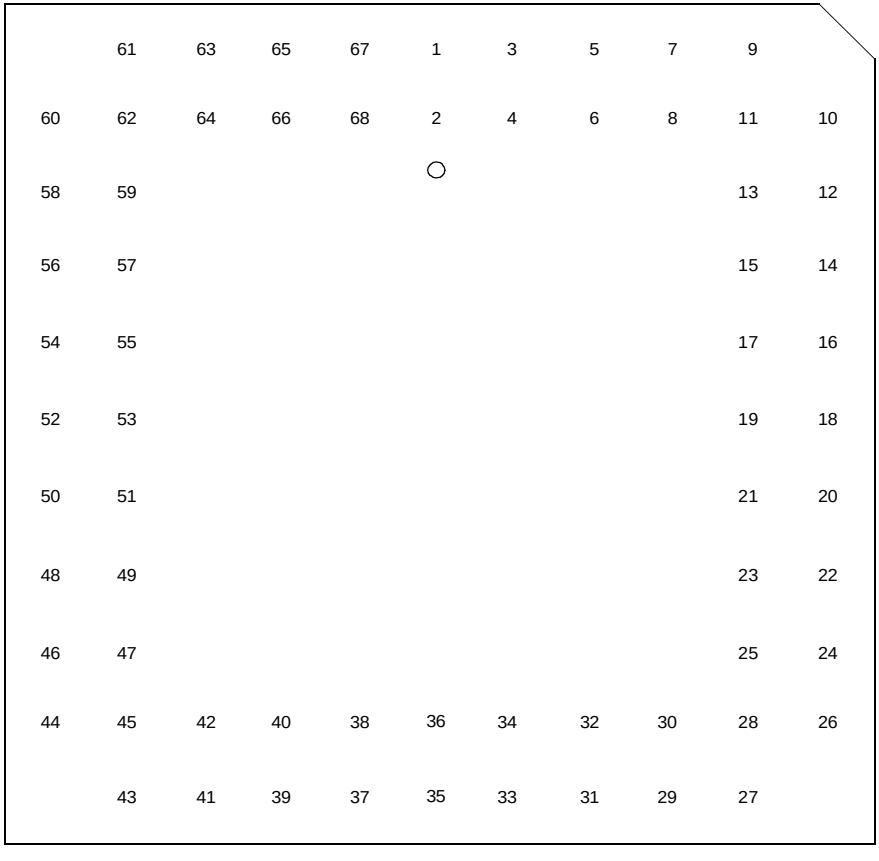


Figure G-2: PGA Package Pinout (Bottom View) for LCC Packages (Continued)

84-pin

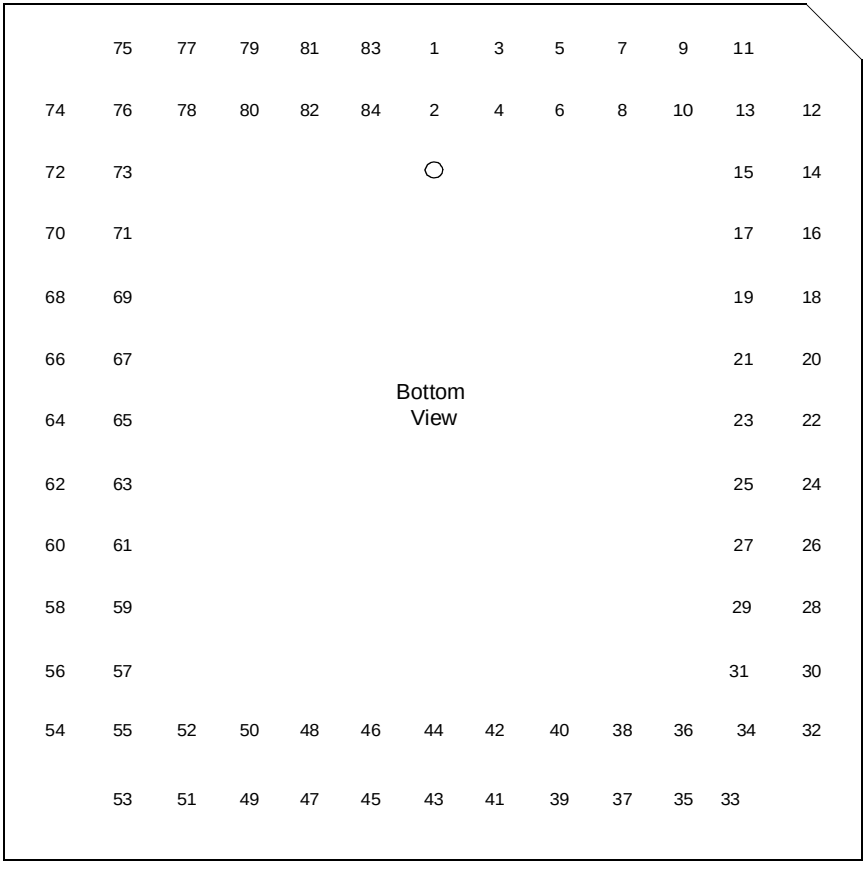
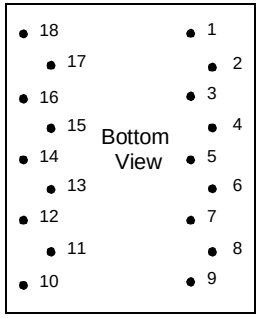
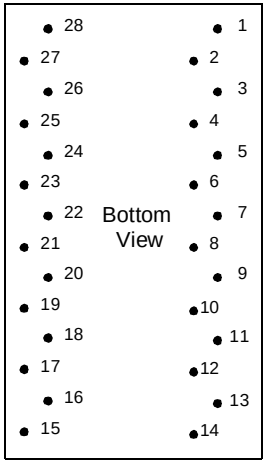


Figure G-3: SOIC Socket Pinouts (Bottom View)

18-pin



28-pin



APPENDIX H: REVISION HISTORY

Revision A

This is the initial released revision of the Enhanced MCU Reference Guide Appendix.

Section 37. Glossary

A

A/D

See description under **"Analog to Digital (A/D)"**.

Access RAM

This is a region of data memory RAM that can be accessed regardless of the currently selected bank. This allows special function registers to be accessed by the instruction without changing the currently selected bank. Access RAM also contains some General Purpose Registers (GPRs). This is useful for the saving of required variables during context switching (such as during an interrupt).

Acquisition Time (TACQ)

This is related to Analog to Digital (A/D) converters. This is the time that the PIC18CXXX A/D's holding capacitor acquires the analog input voltage level connected to it. When the GO bit is set, the analog input is disconnected from the holding capacitor and the A/D conversion is started.

ALU

Arithmetical Logical Unit. Device logic that is responsible for the mathematical (add, subtract, ...), logical (and, or, ...), and shifting operation.

Analog to Digital (A/D)

The conversion of an analog input voltage to a ratiometric digital equivalent value.

Assembly Language

A symbolic language that describes the binary machine code in a readable form.

AUSART

Addressable Universal Synchronous Asynchronous Receiver Transmitter. This module can either operate as a full duplex asynchronous communications port, or a half duplex synchronous communications port. When operating in the asynchronous mode, the USART can be interfaced to a PC's serial port.

B

Bank

This is a method of addressing Data Memory. Since enhanced devices have 8-bits for direct addressing, instructions can address up to 256 bytes. To allow more data memory to be present on a device, data memory is partitioned into contiguous banks of 256 bytes each. To select the desired bank, the bank selection register (BSR) needs to be appropriately configured. 16 banks can be implemented.

Baud

Generally this is how the communication speed of serial ports is described. Equivalent to bits per second (bps).

BCD

See description under **"Binary Coded Decimal (BCD)"**.

Binary Coded Decimal (BCD)

Each 4-bit nibble expresses a digit from 0-9. Usually two digits are contained in a byte yielding a range of 0 - 99.

BOR

See description under **"Brown-out Reset (BOR)"**.

Brown-out

A condition where the supply voltage of the device temporarily falls below the specified minimum operation point. This can occur when a load is switched on and causes the system/device voltage to drop.

Brown-out Reset (BOR)

Circuitry which will force the device to the RESET state if the device's power supply voltage falls below a specified voltage level. Some devices have an internal BOR circuit, while other devices would require an external circuit to be created.

Bus width

This is the number of bits of information that a bus carries. For the Data Memory, the bus width is 8-bits. For enhanced devices the Program Memory bus width is 16-bits.

C

Capture

A function of the CCP module in which the value of a timer/counter is “captured” into a holding register module when a predetermined event occurs.

CCP

Capture, Compare, Pulse Width Modulation (PWM). The CCP module can be configured to operate as an input capture, or a timer compare, or a PWM output.

Compare

A function of the CCP module in which the device will perform an action when a timer’s register value matches the value in the compare register.

Compare Register

A 16-bit register that contains a value that is compared to the 16-bit TMR1 register. The compare function triggers when the counter matches the contents of the compare register.

Capture Register

A 16-bit register that is loaded with the value of the 16-bit TMR1 register when a capture event occurs.

Configuration Word

This is a non-volatile memory location that specifies the characteristics that the device will have for operation (such as oscillator mode, WDT enable, start-up timer enables). These characteristics can be specified at the time of device programming. For EPROM memory devices, as long as the bit is a '1', it may at a later time be programmed as a '0'. The device must be erased for a '0' to be returned to a '1'.

Conversion Time (Tconv)

This is related to Analog to Digital (A/D) converters. This is the time that the PIC18CXXX A/D’s converter requires to convert the analog voltage level on the holding capacitor to a digital value.

CPU

Central Processing Unit. Decodes the instructions, and determines the operands and operations that are needed for program execution. Arithmetic, logical, or shift operations are passed to the ALU.

D

D/A

See description under **"Digital to Analog"**.

DAC

Digital to analog converter.

Data Bus

The bus which is used to transfer data to and from the data memory.

Data EEPROM

Data Electrically Erasable Programmable Read Only Memory. This memory is capable of being programmed and re-programmed by the CPU to ensure that in the case of a power loss, critical values/variables are retained in the non-volatile memory.

Data Memory

The memory that is on the Data Bus. This memory is volatile (SRAM) and contains both the Special Function Registers and General Purpose Registers.

Direct Addressing

When the Data Memory Address is contained in the Instruction. The execution of this type of instruction will always access the data at the embedded address.

Digital to Analog

The conversion of a digital value to an equivalent ratiometric analog voltage.

E

EEPROM

Electrically Erasable Programmable Read Only Memory. This memory has the capability to be programmed and erased in-circuit.

EPROM

Electrically Programmable Read Only Memory. This memory has the capability to be programmed in-circuit. Erasing requires that the program memory be exposed to UV light.

EXTRC

External Resistor-Capacitor (RC). Some devices have a device oscillator option that allows the clock to come from an external RC. This is the same as RC mode on some devices.

F

FLASH Memory

This memory has the capability to be programmed and erased in-circuit. Program Memory technology that is almost functionally equivalent to Program EEPROM Memory.

FOSC

Frequency of the device oscillator.

G

GIO

General Input/Output.

GPIO

General Purpose Input/Output.

GPR

General Purpose Register (RAM). A portion of the data memory that can be used to store the program's dynamic variables.

H

Harvard Architecture

In this architecture, the Program Memory and Data Memory buses are separated. This allows concurrent accesses to Data Memory and Program Memory, which increases the performance of the device. All PICmicro devices implement a Harvard Architecture.

Holding Capacitor

This is a capacitor in the Analog to Digital (A/D) module which "holds" an analog input level once a conversion is started. During acquisition, the holding capacitor is charged/discharged by the voltage level on the analog input pin. Once the conversion is started, the holding capacitor is disconnected from the analog input and "holds" this voltage for the A/D conversion.

HS

High Speed. One of the device oscillator modes. The oscillator circuit is tuned to support the high frequency operation. Currently this allows for operation from 4 MHz to 25 MHz.

I

I^2C

Inter-Integrated Circuit. This is a two wire communication interface. This feature is one of the modes of the **"SSP"** and **"MSSP"** modules.

Indirect Addressing

When the Data Memory Address is not contained in the Instruction, the instruction operates on the INDF address, which causes the Data Memory Address to be the value in the FSR register. The execution of the instruction will always access the data at the address pointed to by the FSR register.

Instruction Bus

The bus which is used to transfer instruction words from the program memory to the CPU.

Instruction Fetch

Due to the Harvard architecture, when one instruction is to be executed, the next location in program memory is “fetched” and ready to be decoded as soon as the currently executing instruction is completed.

Instruction Cycle

The events for an instruction to execute. There are four events which can generally be described as: Decode, Read, Execute, and Write. Not all events will be done by all instructions. To see the operations during the instruction cycle, please look at the description of each instruction. Four external clocks (Tosc) make one instruction cycle (T_{CY}).

Interrupt

A signal to the CPU that causes the program flow to be forced to the Interrupt Vector Address (04h in program memory). Before the program flow is changed, the contents of the Program Counter (PC) are forced onto the hardware stack, so that program execution may return to the interrupted point.

INTRC

Internal Resistor-Capacitor (RC). Some devices have a device oscillator option that allows the clock to come from an internal RC combination.

L

LCD

Liquid Crystal Display. Useful for giving visual status of a system. This may require the specification of custom LCD glass.

LED

Light Emitting Diode. Useful for giving visual status of a system.

Literal

This is a constant value that is embedded in an instruction word.

Long Word Instruction

An instruction word that embeds all the required information (opcode and data) into a single word. This ensures that every instruction is accessed and executed in a single instruction cycle.

LP

One of the device oscillator modes. Used for low frequency operation which allows the oscillator to be tuned for low power consumption. Operation is up to 200 kHz.

LSb

Least Significant Bit.

LSB

Least Significant Byte.

M

Machine cycle

This is a concept where the device clock is divided down to a unit time. For PICmicro devices, this unit time is 4 times the device oscillator (4TOSC), also known as TCY.

MSb

Most Significant Bit.

MSB

Most Significant Byte.

MSSP

Master Synchronous Serial Port. The MSSP has two operational functions. The first is a **"Serial Peripheral Interface (SPI)"** and the second is the Inter-Integrated Circuit (**"I^2C"**). The I^2C function supports both master and slave functions in hardware.

N

Non-Return to Zero (NRZ)

Two-level encoding used to transmit data over a communications medium. A bit value of '1' indicates a high voltage signal. A bit value of '0' indicates a low voltage signal. The data line defaults to a high level.

NRZ

See description under **"Non-Return to Zero (NRZ)"**.

O

Opcode

The portion of the 16-bit instruction word that specifies the operation that needs to occur. The opcode is of variable length depending on the instruction that needs to be executed. The opcode varies from 4-bits to 8-bits. The remainder of the instruction word contains program or data memory information.

Oscillator Start-up Timer (OST)

This timer counts 1024 crystal/resonator oscillator clock cycles before releasing the internal RESET signal.

OST

See description under **"Oscillator Start-up Timer (OST)"**.

P

Pages

Method of addressing the Program Memory. Mid-range devices have 11-bit addressing for CALL and GOTO instructions, which gives these instructions a 2-Kword reach. To allow more program memory to be present on a device, program memory is partitioned into contiguous pages, where each page is 2-Kwords. To select the desired page, the page selection bits (PCLATCH<5:4>) need to be appropriately configured. Since there are presently 2 page selection bits, 4 pages can be implemented. The enhanced devices do not have paging. PIC16CXXX code migrates to the PIC18CXXX without modification (with respect to paging). Optimization may be implemented.

Parallel Slave Port (PSP)

A parallel communication port which is used to interface to a microprocessor's 8-bit data bus.

POP

A term used to refer to the action of restoring information from a stack (software and/or hardware). See **"Serial Peripheral Interface (SPI)"**.

Postscaler

A circuit that slows the rate of the interrupt generation (or WDT Reset) from a counter/timer by dividing it down.

Power-on Reset (POR)

Circuitry which determines if the device power supply voltage rose from a powered down level (0V). If the device power supply voltage is rising from ground, a device RESET occurs and the PWRT is started.

Power-up Timer (PWRT)

A timer which holds the internal RESET signal low for a timed delay to allow the device voltage to reach the valid operating voltage range. Once the timer times out, the OST circuitry is enabled (for all crystal/resonator device oscillator modes).

Prescaler

A circuit that slows the rate of a clocking source to a counter/timer.

Program Bus

The bus used to transfer instruction words from the program memory to the CPU.

Program Counter

A register which specifies the address in program memory that contains the next instruction to execute.

Program Memory

Any memory that is on the program memory bus. Static variables may be contained in program memory, such as tables.

PSP

See description under **"Parallel Slave Port (PSP)"**.

Pulse Width Modulation (PWM)

A serial signal in which the information is contained in the width of a (high) pulse of a constant frequency signal. A PWM output, from the CCP module, of the same duty cycle requires no software overhead.

PUSH

A term used to refer to the action of saving information onto a stack (software and/or hardware). See **"Serial Peripheral Interface (SPI)"**.

PWM

See description under **"Pulse Width Modulation (PWM)"**.

Q

Q-cycles

This is the same as a device oscillator cycle. There are 4 Q-cycles for each instruction cycle.

R

RC

Resistor-Capacitor. The default configuration for the device oscillator. This allows a "**R**eal-**C**heap" implementation for the device clock source. This clock source does not supply an accurate time-base.

Read-Modify-Write

This is where a register is read, then modified, and then written back to the original register. This may be done in one instruction cycle or multiple instruction cycles.

Register File

This is the Data Memory. Contains the SFRs and GPRs.

ROM

Read Only Memory. Memory that is fixed and cannot be modified.

S

Sampling Time

Sampling time is the complete time to get an A/D result. It includes the acquisition time and the conversion time.

Serial Peripheral Interface (SPI)

This is one of the modes of the **"SSP"** and **"MSSP"** modules. This is typically a 3-wire interface, with a data out line, a data in line, and a clock line. Since the clock is present, this is a synchronous interface.

SFR

Special Function Register. These registers contain the control bits and status information for the device.

Single Cycle Instruction

An instruction that executes in a “single” machine cycle (TCY).

SLEEP

This is a low power mode of the device, where the device’s oscillator circuitry is disabled. This reduces the current the device consumes. Certain peripherals may be placed into modes where they continue to operate.

Special Function Registers (SFR)

These registers contain the control bits and status information for the device.

SPI

See description under **"Serial Peripheral Interface (SPI)"**.

SSP

Synchronous Serial Port. The SSP has two operational functions. The first is a **"Serial Peripheral Interface (SPI)"**and the second is the Inter-Integrated Circuit (**"I^2C"**). The I^2C function supports the slave function in hardware and has additional status information to support a software implemented master.

Stack

A portion of the CPU that retains the return address for program execution. The stack gets loaded with the value in the Program Counter when a CALL instruction is executed or if an interrupt occurs.

T

TAD

In the A/D Converter, the time for a single bit of the analog voltage to be converted to a digital value.

TCY

The time for an instruction to complete. This time is equal to Fosc/4 and is divided into four Q-cycles.

TOSC

The time for the single period of the device oscillator.

U

USART

Universal Synchronous Asynchronous Receiver Transmitter. This module can either operate as a full duplex asynchronous communications port, or a half duplex synchronous communications port. When operating in the asynchronous mode, the USART can be interfaced to a PC's serial port.

V

Voltage Reference (VREF)

A voltage level that can be used as a reference point for A/D conversions (AVDD and AVSS) or the trip point for comparators.

von Neumann Architecture

In this architecture the Program Memory and Data Memory are contained in the same area and use the same bus. This means that accesses to the program memory and data memory must occur sequentially, which affects the performance of the device.

W

W Register

See description under **"Working Register (WREG)"**.

Watchdog Timer (WDT)

Used to increase the robustness of a design by recovering from software flows that were not expected in the design of the product or from other system related issues. The Watchdog Timer causes a RESET if it is not cleared prior to overflow. The clock source for a PICmicro device is an on-chip RC oscillator which enhances system reliability.

WDT

Watchdog Timer.

Working Register (WREG)

Can also be thought of as the accumulator of the device. Also used as an operand in conjunction with the ALU during two operand instructions.

X

XT

One of the device oscillator modes. Used for operation from 100 kHz to 4 MHz.

37.1 Revision History

Revision A

This is the initial released revision of the Glossary.

APPENDIX I: SOURCE CODE

```
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the "Company") for its PICmicro® Microcontroller is intended and
; supplied to you, the Company's customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
;  Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;*********************************************************
;   PIC18CXX2 EXAMPLE CODE FOR PICDEM-2
;
;   TITLE: USART Demo Demonstration
;   FILENAME: usart.asm
;   REVISION HISTORY:  A 5/13/00 jb   format change
;   HARDWARE: PICDEM-2 board
;   FREQUENCY: 4MHz
:
;*********************************************************
```


Source Code

```
; This program demonstrates basic functionality of the USART.
;
; Port B is connected to 8 LEDs.
; When the PIC18C452 receives a word of data from
; the USART, the value is displayed on the LEDs and
; is retransmitted to the host computer.
;
; Set terminal program to 9600 baud, 1 stop bit, no parity

    list p=18c452; set processor type
    list n=0; supress page breaks in list file
    include <P18c452.INC>

;*************************************************************
; Reset and Interrupt Vectors

    org 00000h; Reset Vector
    gotoStart

    org 00008h; Interrupt vector
    gotoIntVector

;*************************************************************
; Program begins here

    org 00020h; Beginning of program EPROM
Start
    clrfLATB; Clear PORTB output latches
    clrfTRISB ; Config PORTB as all outputs
    bcf TRISC,6; Make RC6 an output

    movlw19h; 9600 baud @4MHz
    movwfSPBRG

    bsf TXSTA,TXEN; Enable transmit
    bsf TXSTA,BRGH; Select high baud rate

    bsf RCSTA,SPEN; Enable Serial Port
    bsf RCSTA,CREN; Enable continuous reception

    bcf PIR1,RCIF; Clear RCIF Interrupt Flag
    bsf PIE1,RCIE; Set RCIE Interrupt Enable
    bsf INTCON,PEIE; Enable peripheral interrupts
    bsf INTCON,GIE; Enable global interrupts

;*************************************************************
; Main loop
```

```
Main
    gotoMain; loop to self doing nothing

;*********************************************************
; Interrupt Service Routine

IntVector
    ; save context (WREG and STATUS registers) if needed.

    btfssPIR1,RCIF; Did USART cause interrupt?
    gotoOtherInt; No, some other interrupt

    movlw06h; Mask out unwanted bits
    andwfRCSTA,W; Check for errors
    btfssSTATUS,Z; Was either error status bit set?
    gotoRcvError; Found error, flag it

    movfRCREG,W; Get input data
    movwfLATB; Display on LEDs
    movwfTXREG; Echo character back
    gotoISREnd; go to end of ISR, restore context, return

RcvError
    bcf RCSTA,CREN; Clear receiver status
    bsf RCSTA,CREN
    movlw0FFh; Light all LEDs
    movwfPORTB
    gotoISREnd; go to end of ISR, restore context, return

OtherInt
    goto$   ; Find cause of interrupt and service it before returning from
            ; interrupt. If not, the same interrupt will re-occur as soon
            ; as execution returns to interrupted program.

ISREnd
    ; Restore context if needed.
    retfie

    end
```


Source Code

```
;
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the “Company”) for its PICmicro® Microcontroller is intended and
; supplied to you, the Company’s customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
;  Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;*************************************************************
;   PIC18CXX2 EXAMPLE CODE FOR PICDEM-2
;
;   TITLE:Button Press Demonstration
;   FILENAME:  bttn.asm
;   REVISION HISTORY:  A 5/13/00 jb   format change
;   HARDWARE: PICDEM-2 board
;   FREQUENCY: 4MHz
:
;*************************************************************
; This program demonstrates how to read a push-button and control LED's.
;
; Port B is connected to 8 LEDs.
; RA4 is connected to a switch (S2).
; This program increments a file register count every time S2 is pressed.
; The value of count is displayed on the LEDs connected to Port B.
; The LEDs should increment in a binary manner each time S2 is pressed.

    list p=18c452
    #include<P18C452.INC>

;*************************************************************
; variables

Countequ0x000
```

```
;*************************************************************
; reset vectors

    org 00000h; Reset Vector
    gotoStart

;*************************************************************
;program code starts here

    org 00020h; Beginning of program EPROM
Start
    clrfLATB; Clear PORTB output latch
    clrfTRISB; Make PORTB pins all outputs
    clrfCount; Clear Count
Loop
    btfscPORTA,4; Has S2 been pressed? (Normally high, goes low when pressed.)
    gotoLoop; No, check again

IncCount
    incfCount,F; Increment Count
    movffCount,LATB; move Count to PORTB

Debounce
    btfssPORTA,4; Has key been released?
    gotoDebounce; No, wait some more
    gotoLoop; yes, wait for next key press


    END     ; directive indicates end of code
```


Source Code

```
;
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the "Company") for its PICmicro® Microcontroller is intended and
; supplied to you, the Company's customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
;  Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;*********************************************************
;   PIC18CXX2 EXAMPLE CODE FOR PICDEM-2
;
;   TITLE:I2C hardware interface PICmicro to serial EEPROM
;   FILENAME:  i2c.asm
;   REVISION HISTORY:  A 5/13/00 jb   format change
;   HARDWARE: PICDEM-2 board
;   FREQUENCY: 4MHz
;
;*********************************************************
; This program uses the advanced features of the PIC18C452, specifically
; full hardware support for master mode I2C
;
; This program loads EEPROM address 0x00 to 0xFF with 0x00 to 0xFF
; (each location contains its own address).
;
; Each location is then read out, and compared to what is expected.
; The data is displayed on the LEDS. If the data is wrong,
; the TX LED will flash briefly before proceeding to the next address.
;
; Revised Version(05-05-99).
; Note:  1) All timing is based on a reference crystal frequency of 4MHz
;   which is equivalent to an instruction cycle time of 1 usec.
;        2) Address and literal values are read in hexidecimal unless
;   otherwise specified.
;        3) The PIC18C452 MSSP module offers full hardware support for
```

```
;   master mode I2C.
;*************************************************************

    LIST P=18C452
    #include <P18C452.INC>
    listn=0 ; suppress list file page breaks
    listST=off; suppress list file symbol table

;*************************************************************
; Register File Assignment

EEADDRequ0x000; Address register
EEDATAequ0x001; data to read/write
EESLAVE equ 0x002; Device address (1010xxxy)
DelayCtr1equ0x003; delay routine counter
DelayCtr2equ0x004; delay routine counter

SlaveAddrequ0xA0; slave address literal

;*************************************************************
; Vector Assignment
    ORG 0x00000
    gotoStart; Reset Vector

;*************************************************************
; Main Program

    ORG 0x00020; Start of Program space
Start
    ; initialize PORTB
    clrfLATB; Clear PORTB output latch
    clrfTRISB; Set PORTB as all outputs

    ; configure SSP for hardware master mode I2C
    bsf SSPSTAT,SMP; I2C slew rate control disabled
    bsf SSPCON1,SSPM3; I2C master mode in hardware
    bsf SSPCON1,SSPEN; enable SSP module

    movlw0x09; set I2C clock rate to 100kHz
    movwfSSPADD

    bsf TRISC,3; I2C SCL pin is input
    bsf PORTC,3; (will be controlled by SSP)
    bsf TRISC,4; I2C SDA pin is input
    bsf PORTC,4; (will be controlled by SSP)

    movlwSlaveAddr; EEPROM I2C address
```


Source Code

```
    movwfEESLAVE

Main
    clrfPORTB; initialize variables
    clrfEEDATA
    clrfEEADDR

WrEEPROM
    bsf PORTB,7; indicate write, light TX LED
    rcallDelay

    bcf EESLAVE,0; write mode
    rcallWakeSlave; gets slave attention
    rcallWrADDR; sends EEPROM address
    rcallWrDATA; sends data to slave
    rcallStop; send stop bit

    incfPORTB,F; increment count
    incfEEDATA,F; increment data
    incfEEADDR,F; Point to next address

    btfssEEADDR,7; at end of EEPROM?
    gotoWrEEPROM; no, write more data

RdLoop
    clrfPORTB; initialize variables
    clrfEEDATA
    clrfEEADDR

RdEEPROM
    rcallDelay

    bcf EESLAVE,0; write mode
    rcallWakeSlave; gets slave attention
    rcallWrADDR; sends EEPROM address
    rcallStop; send stop bit

    bsf EESLAVE,0; read mode
    rcallWakeSlave; gets slave attention
    rcallRdDATA; receive one data byte, leaves idle

    movfEEDATA,W; get data
    movwfPORTB; move received data to PORTB
    xorwfEEADDR,W; compare data with address
    bz  GoodData; branch if DATA = ADDR
    rcallErrorloop; DATA is wrong, indicate error
```

```
GoodDataincfEEADDR,F; Point to next address
    btfssEEADDR,7; at end of EEPROM?
    gotoRdEEPROM; no, read more data

    gotoMain; do it all over again

;*****************************************************
; TX LED flashes to indicate error while displaying received data.

Errorloop
    rcallDelay
    btg PORTB,7; Toggle TX LED
    rcallDelay
    btg PORTB,7; Toggle TX LED
    rcallDelay
    btg PORTB,7; Toggle TX LED
    rcallDelay
    btg PORTB,7; Toggle TX LED
    rcallDelay
    btg PORTB,7; Toggle TX LED
    rcallDelay
    btg PORTB,7; Toggle sTX LED
    return

;*****************************************************
; sends start bit, slave address
; if ACK not recieved, sends restart, tries again
; execution can get stuck in this loop if slave not present
; can be used to poll slave status (retries until slave responds)

WakeSlave
    bsf SSPCON2,SEN; Send start bit
    btfscSSPCON2,SEN; Has SEN cleared yet?
    goto$-2 ; No, loop back to test.

rWakeSlave
    bcf PIR1,SSPIF; clear interrupt flag
    nop
    movfEESLAVE,W
    movwfSSPBUF; move slave address to SSPBUF
    btfssPIR1,SSPIF; has SSP completed sending SLAVE Address?
    goto$-2 ; no, loop back to test

    btfssSSPCON2,ACKSTAT; was ACK received from slave?
    return  ; yes, return to calling routine

    bsf SSPCON2,RSEN; send repeated start bit
```


Source Code

```
    btfscSSPCON2,RSEN; has repeated start been sent yet?
    goto$-2 ; no, loop back to test

    bra rWakeSlave; send slave address again

;*************************************************************
; writes EEPROM memory address, hangs if no ACK

WrADDR
    bcf PIR1,SSPIF; clear interrupt flag

    movffEEADDR,SSPBUF; move EEPROM address to SSPBUF
    btfssPIR1,SSPIF; has SSP completed sending EEPROM Address?
    goto$-2 ; no, loop back to test

    btfscSSPCON2,ACKSTAT; has slave sent ACK?
    goto$-2 ; no, try again

    return

;*************************************************************
; Sends one byte of data to slave, hangs if no ACK

WrDATA
    bcf PIR1,SSPIF; clear interrupt flag

    movffEEDATA,SSPBUF; move data to SSPBUF
    btfssPIR1,SSPIF; has SSP completed sending data to EEPROM?
    goto$-2 ; no, loop back to test

    btfscSSPCON2,ACKSTAT; has slave sent ACK?
    goto$-2 ; no, try again

    return

;*************************************************************
; receive one byte from slave
; do not send ACK, send stop bit instead

RdDATA
    bcf PIR1,SSPIF; clear interrupt flag

    bsf SSPCON2,RCEN; enable receive mode
    btfssPIR1,SSPIF; has SSP received a data byte?
    goto$-2 ; no, loop back to test

    bsf SSPCON2,ACKDT; no ACK
```

```
    bsf SSPCON2,ACKEN; send ACKDT bit

    btfscSSPCON2,ACKEN; has ACKDT bit been sent yet?
    goto$-2 ; no, loop back to test

    bsf SSPCON2,PEN; send stop bit
    btfscSSPCON2,PEN; has stop bit been sent?
    goto$-2 ; no, loop back to test

    movffSSPBUF,EEDATA; save data to RAM
    bcf SSPCON2,RCEN; disable receive mode

    return


;*********************************************************
; Sends stop bit, waits until sent

Stop
    bsf SSPCON2,PEN; send stop bit
    btfscSSPCON2,PEN; has stop bit been sent?
    goto$-2 ; no, loop back to test

    return

;*********************************************************
; a delay of 98.57mS

Delay
    movlw0x80
    movwfDelayCtr2; preset
    clrfDelayCtr1; clear counter

Delay1
    decfszDelayCtr1; decrement counter
    bra Delay1; back to top of loop

    decfszDelayCtr2; decrement counter
    bra Delay1; back to top of loop

    return


    END
```


Source Code

```
;
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the “Company”) for its PICmicro® Microcontroller is intended and
; supplied to you, the Company’s customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
;  Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;*********************************************************
;   PIC18CXX2 EXAMPLE CODE FOR PICDEM-2
;
;   TITLE: Interrupt Priority Demonstration
;   FILENAME: intrp_ex .asm
;   REVISION HISTORY:  A 5/13/00 jb   format change
;   HARDWARE: PICDEM-2 board
;   FREQUENCY: 4MHz
;
;*********************************************************
; This program uses Timer1 and Timer3 to demonstrate the use of
; interrupt priority.
;
; Timer1 is configured for high-priority interrupts
; and Timer3 is configured for low-priority interrupts.  By writing
; to the PORTB LEDS, it is shown that a high-priority interrupts
; override low-priority interrupts.

    list p=18c452, n=48, t=ON, st=OFF
    #include "p18c452.inc"

;-----------------BIT DEFINITIONS-----------------------------------

F                EQU      0x0001

;-----------------VECTORS-------------------------------------------
```

```
org 0x000000; reset vector
bra START

org 0x000008; high priority interrupt vector
bra TMR1_ISR

org 0x000018; low priority interrupt vector
bra TMR3_ISR

;--------------------PROGRAM-------------------------------------
START

   rcall    INIT

;Set up priority interrupts.
    bsf RCON,IPEN;enable priority interrupts.
    bsf IPR1,TMR1IP;set Timer1 as a high priority interrupt source
    bcf IPR2,TMR3IP;set Timer3 as a low priority interrupt source
    bcf PIR1,TMR1IF;clear the Timer1 interrupt flag
    bcf PIR2,TMR3IF;clear the Timer3 interrupt flag
    bsf PIE1,TMR1IE;enable Timer1 interrupts
    bsf PIE2,TMR3IE;enable Timer3 interrupts
    bsf INTCON,GIEH;set the global interrupt enable bits
    bsf INTCON,GIEL;"

;Timer1 setup
    clrfT1CON
    clrfTMR1H;clear Timer1 high
    clrfTMR1L;clear Timer1 low
    bsf T1CON,TMR1ON;turn on Timer1

;Timer3 setup
    clrfT3CON
    movlw0xF0
    movwfTMR3H;write 0xf000 to Timer3
    clrfTMR3L
    bsf T3CON,TMR3ON;turn on Timer3

MLOOP
    gotoMLOOP

;-------------------------------SUBROUTINES---------------------------------
TMR1_ISR   ; high priority isr

    bcf PIR1,TMR1IF;Clear the Timer1 interrupt flag.
    bcf PORTB,0;Turn off PORTB<0> to indicate high priority
```


Source Code

```
            ; interrupt has overridden low priority.
    bsf PORTB,7;Turn on PORTB<7> to indicate high priority
            ; interrupt is occuring.
T1POLL
    btfssPIR1,TMR1IF;Poll TMR11 interrupt flag to wait for another
            ; TMR1 overflow.
    bra T1POLL
    bcf PIR1,TMR1IF;Clear the Timer1 interrupt flag again.
    bcf PORTB,7;Turn off PORTB<7> to indicate the
            ; high-priority ISR is over.
    retfie


TMR3_ISR   ;low priority isr
    bcf PIR2,TMR3IF;Clear the TMR3 interrupt flag.
    movlw0xF0;Load TMR3 with the value 0xF000
    movwfTMR3H
    clrfTMR3L
    bsf PORTB,0;Turn on PORTB<0> to indicate low priority
            ; interrupt is occurring.
T3POLL
    btfssPIR2,TMR3IF;Poll TMR3 interrupt flag to wait for another TMR3 overflow.
    bra T3POLL
    movlw0xF0;Load TMR3 with the value 0xF000 again.
    movwfTMR3H
    clrfTMR3L
    bcf PIR2,TMR3IF;Clear the Timer3 interrupt flag again.
    bcf PORTB,0;Turn off PORTB<0> to indicate the low-priority ISR is over.

    retfie


INIT
    clrfPORTB; setup portb for outputs
    clrfDDRB

    return

END
```

```
;
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the “Company”) for its PICmicro® Microcontroller is intended and
; supplied to you, the Company’s customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
;  Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;*************************************************************
;   PIC18CXX2 EXAMPLE CODE FOR PICDEM-2
;
;   TITLE: Oscillator Switching Demonstration
;   FILENAME: osc.asm
;   REVISION HISTORY:  A 5/13/00 jb   format change
;   HARDWARE: PICDEM-2 board
;   FREQUENCY: 4MHz
:
;*************************************************************
; This program demonstrates the use of oscillator switching.
;
; The data held in MYDAT is periodically flashed on the PORTB
; LEDs.  Each time a keypress is detected on RA4, the data is
; incremented and the oscillator source is changed.

    list p=18c452, n=48, t=ON, st=OFF
    #include "p18c452.inc"

KEY EQU   4

;-------------------18C452 RAM LOCATIONS-----------------------------


COUNT0EQU      0x0000      ; used for software timing loop
COUNT1EQU      0x0001      ;            "
```


Source Code

```
MYDAT   EQU      0x0002      ; data storage register

;-----------------BIT DEFINITIONS-----------------------------------

F              EQU      0x0001

;-----------------VECTORS-------------------------------------------

ORG 0x000000; reset vector
BRA START

;--------------------PROGRAM-----------------------------------

START
   rcall    INIT; setup ports, etc.
   bsf      T1CON, T1OSCEN; setup the LP oscillator

MLOOP

   btfss    PORTA,KEY
   rcall    KEYPRESS; call keypress routine if
           ; button is pressed
   movff    MYDAT,PORTB; move data to portb
   rcall    WAIT; wait a while
   clrf     PORTB; clear the port
   rcall    WAIT; wait a while

   bra      MLOOP

;------------------------------SUBROUTINES--------------------------------

KEYPRESS

   btfss    PORTA,KEY
   bra      KEYPRESS
   incf     MYDAT

; Oscillator source changes every time the subroutine is called.

   btg      OSCCON,SCS

   return

INIT

   clrf     PORTA
   clrf     PORTB
```

```
   bsf      DDRA,4
   clrf     DDRB

   return

WAIT                                    ; software time delay
   clrf     COUNT0
   movlw    0x08
   movwf    COUNT1
WLOOP
   decfsz   COUNT0,F
   bra      WLOOP
   decfsz   COUNT1,F
   bra      WLOOP

   return

    end
```


Source Code

```
;
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the “Company”) for its PICmicro® Microcontroller is intended and
; supplied to you, the Company’s customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
;  Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;*************************************************************
;   PIC18CXX2 EXAMPLE CODE FOR PICDEM-2
;
;   TITLE: Pulse Width Modulation Demonstration
;   FILENAME: pwm_ex .asm
;   REVISION HISTORY:  A 5/13/00 jb   format change
;   HARDWARE: PICDEM-2 board
;   FREQUENCY: 4MHz
:
;*************************************************************
; This program demonstrates pulse width modulation using CCP2.
;
; The PWM period is fixed and then the duty cycle is varied by
; looking up a new value in a table.  When the final value of
; the table is read, the table is read in reverse. The PWM output
; is set to RC1.  Parts Y3, C6 & C7 are are not installed the
; on the PICDEM 2 board.  PWM signal can be observed on RC1 pin.
;
    list p=18c452, n=48, t=ON, st=OFF
    #include "p18c452.inc"

;*************************************************************
; bit definitions

F                EQU      0x0001
```

```
;*************************************************************
; 18C452 RAM LOCATIONS

DIRFLAG           EQU      0x0000

;*************************************************************
; 18C452 ROM LOCATIONS

TABLADDR          EQU      0x0003000

;*************************************************************
; vectors

    org 0x000000; reset vector
    bra START

    org 0x000008; high priority interrupt vector
    bra TMR1_ISR

;*************************************************************
; program
START

; Set up PWM module
; Set PWM period by writing to PR2
; Set PWM duty cycle by writing to the CCPR2L register
; and the CCP2CON<5:4>>bits
; Make the CCP2 pin an output by clearing the TRISC<2> bit.
    clrfCCP2CON;CCP module is off
    bsf CCP2CON, CCP2M3;select PWM mode
    bsf CCP2CON, CCP2M2;select PWM mode
    movlw0x3F;Set PWM frequency to 78.12kHz
    movwfPR2;
    bcf TRISC, 1;make channel 1 an output
    movlw0x00
    movwfCCPR2L

;Set the TMR2 prescale value and enable Timer2 by writing to T2CON
;Configure the CCP2 module for PWM operation
    clrfT2CON;clear T2CON
    clrfTMR2;clear Timer2
    bsf T2CON,TMR2ON;turn on Timer2

;initialize direction flag for table
    clrfDIRFLAG

; Initialize the table pointer registers
```


Source Code

```
; to the first location of the data stored in program memory.

    movlwUPPER(TABLADDR)
    movwfTBLPTRU
    movlwHIGH(TABLADDR)
    movwfTBLPTRH
    movlwLOW(TABLADDR)
    movwfTBLPTRL

;setup interrupt
    bsf RCON,IPEN;enable priority interrupts.
    bsf IPR1,TMR1IP;set Timer1 as a high priority interrupt source
    bcf PIR1,TMR1IF;clear the Timer1 interrupt flag
    bsf PIE1,TMR1IE;enable Timer1 interrupts
    bsf INTCON,GIEH;set the global interrupt enable bits
    bsf INTCON,GIEL;"

;Timer1 setup
    clrfT1CON
    clrfTMR1H;clear Timer1 high
    clrfTMR1L;clear Timer1 low
    bsf T1CON,TMR1ON;turn on Timer1

MLOOP
    gotoMLOOP

;***********************************************************
; subroutines

TMR1_ISR   ; high priority isr

    bcf PIR1,TMR1IF;Clear the Timer1 interrupt flag.
    rcall    CHECK_ADDR

    btfsc    DIRFLAG,0
    bra RD_DOWN



RD_UP

; Code here does a table read, post-increments, and writes the
; value to the CCPR2L PWM duty-cycle register.

    tblrd*+
    movffTABLAT,CCPR2L
```

```
    bra T1POLL

RD_DOWN

; Code here does a table read, post-decrements, and writes the
; value to the CCPR2L PWM duty-cycle register.

    tblrd*-
    movffTABLAT,CCPR2L

    bra T1POLL

T1POLL
    btfssPIR1,TMR1IF;Poll TMR11 interrupt flag to wait for another
           ; TMR1 overflow.
    bra T1POLL
    bcf PIR1,TMR1IF;Clear the Timer1 interrupt flag again.

    retfie

CHECK_ADDR
   movlw    LOW(TABEND) - 1
   subwf    TBLPTRL,W
   bnz      CHECK_LOW
   setf     DIRFLAG
   return

CHECK_LOW
   movlw    LOW(TABLADDR)
   subwf    TBLPTRL,W
   bnz      DONE_CHECK
   clrf     DIRFLAG
DONE_CHECK
   return


;------------------------------DATA-----------------------------------

   org TABLADDR

   DB  0x00,0x06,0x0C,0x12,0x18,0x1E,0x23,0x28
   DB  0x2D,0x31,0x35,0x38,0x3A,0x3D,0x3E,0x3F

TABEND

    END
```


Source Code

```
;
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the “Company”) for its PICmicro® Microcontroller is intended and
; supplied to you, the Company’s customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
;  Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;*********************************************************
;   PIC18CXX2 EXAMPLE CODE FOR PICDEM-2
;
;   TITLE: Table Read Demonstration
;   FILENAME: table.asm
;   REVISION HISTORY:  A 5/13/00 jb   format change
;   HARDWARE: PICDEM-2 board
;   FREQUENCY: 4MHz
:
;*********************************************************
; This program demonstrates the use of the TBLRD instruction
; to read program memory.
;
; A pre-defined sequence of data
; bytes is included in program memory.  Each time the button
; on pin RA4 is pressed, the next byte of data is accessed
; and displayed on the PORTB LEDS.  When the beginning or end of
; the data is reached, the direction of access is reversed.

    list p=18c452, n=48, t=ON, st=OFF
    #include "p18c452.inc"

;-------------------18C452 RAM LOCATIONS-----------------------------

DIRFLAG           EQU      0x0000
```

```
;------------------18C452 ROM LOCATIONS-----------------------------

TABLADDR          EQU      0x0003000

;-----------------BIT DEFINITIONS-----------------------------------

KEY               EQU      4

;-----------------VECTORS-------------------------------------------

ORG 0x000000; rest vector
GOTOSTART

ORG 0x000008; high priority interrupt vector
GOTOSTART

ORG 0x000018; low priority interrupt vector
GOTOSTART


;--------------------PROGRAM-----------------------------------


START

   bsfDDRA,4; porta button input

   clrf PORTB; setup portb for outputs
   clrf DDRB

   clrf DIRFLAG

; Code should be written here to intialize the table pointer registers
; to the first location of the data stored in program memory.  Use
; the appropriate assembler directives to accomplish this.
; (Refer to the 'DATA' statements in this source code.)

   movlwUPPER(TABLADDR)
   movwfTBLPTRU
   movlwHIGH(TABLADDR)
   movwfTBLPTRH
   movlwLOW(TABLADDR)
   movwfTBLPTRL

MLOOP
   btfsc    PORTA,KEY; keypress routine
   bra      $ - 2; decrenment PC by 2 because
```


Source Code

```
   btfss    PORTA,KEY; of byte addressing!
   bra      $ - 2

   rcall    CHECK_ADDR

   btfsc    DIRFLAG,0
   bra      RD_DOWN

RD_UP

; Code here does a table read, post-increments, and writes the
; value to the PORTB LEDS

   tblrd*+
   movff    TABLAT,PORTB

   bra      MLOOP

RD_DOWN

; Code here does a table read, post-decrements, and writes the
; value to the PORTB LEDS

   tblrd*-
   movff    TABLAT,PORTB

   bra      MLOOP

;-----------------------------SUBROUTINES-------------------------

CHECK_ADDR
   movlw    LOW(TABEND) - 1
   subwf    TBLPTRL,W
   bnz      CHECK_LOW
   setf     DIRFLAG
   return

CHECK_LOW
   movlw    LOW(TABLADDR)
   subwf    TBLPTRL,W
   bnz      DONE_CHECK
   clrf     DIRFLAG
DONE_CHECK
   return


;-------------------------------DATA-----------------------------------------
```

```
    org TABLADDR

    DATA  0x0201,0x0804,0x2010,0x8040      ; 0x003000 - 0x003007
    DATA  0x4281,0x1824,0x2400,0x8142      ; 0x003008 - 0x00300F
    DATA  0x1211,0x1814,0x2818,0x8848      ; 0x003010 - 0x003017
    DATA  0xaa55,0xaa55,0x0100,0x0703      ; 0x003018 - 0x00301F
    DATA  0x1F0F,0x7F3F,0xFFFF,0xFFFF      ; 0x003020 - 0x003027

TABEND

    END
```

Source Code

```
;
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the “Company”) for its PICmicro® Microcontroller is intended and
; supplied to you, the Company’s customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
;  Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;*********************************************************
;   PIC18CXX2 EXAMPLE CODE FOR PICDEM-2
;
;   TITLE: Timer Read/Write Demonstration
;   FILENAME: tmrrw.asm
;   REVISION HISTORY:  A 5/13/00 jb   format change
;   HARDWARE: PICDEM-2 board
;   FREQUENCY: 4MHz
:
;*********************************************************
; This program uses Timer1 and Timer3 to demonstrate the use of
; 8 and 16 bit write modes.
;
; The counters are used to maintain
; overflow count registers (similar to a RTCC).  The 16 bit write to
; Timer3 will introduce an error between the overflow registers.
; This error is calculated in the main program loop and displayed on
; the PORTB LEDS.

    list p=18c452, n=48, t=ON, st=OFF
    #include "p18c452.inc"

;-------------------18C452 RAM LOCATIONS-----------------------------

T1COUNT            EQU      0x0000
T3COUNT            EQU      0x0001
```

```
;------------------BIT DEFINITIONS-----------------------------------

F                EQU      0x0001

;------------------VECTORS-------------------------------------------

    org 0x000000; reset vector
    bra START

    org 0x000008; high priority interrupt vector
    bra TMR_ISR

    org 0x000018; low priority interrupt vector
    bra START


;--------------------PROGRAM-----------------------------------

START

    rcallINIT

; Setup Timer1

    clrf     T1CON
    bsf  T1CON,T1CKPS1; set prescaler 1:8
    bsf  T1CON,T1CKPS0; set prescaler 1:8
    bsf      T1CON,RD16; enable 16 bit read/write mode
    movlw    0x80; initialize TMR1 with 8000
    movwf    TMR1H;
    clrf     TMR1L;
    bsf      T1CON,TMR1ON; turn on TMR1

; Setup Timer3

    clrf     T3CON
    bsf      T3CON,TMR3CS
    bsf      T3CON,RD16
    bsf T3CON,T3CKPS1; set prescaler 1:8
    bsf T3CON,T3CKPS0; set prescaler 1:8
    bcf      T3CON,RD16; enable 8 bit read/write mode
    movlw    0x80; initialize TMR3 with 8000
    movwf    TMR3H
    clrf     TMR3L
    bsf      T3CON,TMR3ON
```


Source Code

```
MLOOP

; Subtract T3COUNT from T1COUNT
; and write the result to PORTB LEDS.

    movf     T3COUNT,W
    subwf    T1COUNT,W
    movwf    PORTB

    bra      MLOOP

;------------------------------SUBROUTINES--------------------------------

TMR_ISR

    btfsc    PIR1,TMR1IF; check which timer caused the interrupt
    bra      T1_HANDLER

    btfsc    PIR2,TMR3IF
    bra      T3_HANDLER

    retfie

T1_HANDLER
; Load Timer1 with 0x8000 by writing to the high byte only.
; Increment the T1COUNT register.
; Clear the Timer1 interrupt flag.

    movlw    0x80
    movwf    TMR1H
    incf     T1COUNT
    bcf      PIR1,TMR1IF

   retfie

T3_HANDLER

; Load Timer3 with 0x8000 by performing a 16-bit timer write.
; Increment the T3COUNT register.
; Clear the Timer3 interrupt flag.

    movlw    0x80
    movwf    TMR3H
    clrf     TMR3L
    incf     T3COUNT
```

```
    bcf      PIR2,TMR3IF

   retfie


;-----------------------------SUBROUTINES---------------------------------

INIT
   clrf     T1COUNT
   clrf     T3COUNT

   bsf      DDRA,4; porta

   clrf     PORTB; setup portb for outputs
   clrf     DDRB

   bsf      PIE1,TMR1IE; enable interrupts
   bsf      PIE2,TMR3IE
   bsf      INTCON,PEIE; enable peripheral interrupts
   bsf      INTCON,GIE; enable global interrupts
   return

    END
```


Source Code

```
;
; Software License Agreement
;
; The software supplied herewith by Microchip Technology Incorporated
; (the “Company”) for its PICmicro® Microcontroller is intended and
; supplied to you, the Company’s customer, for use solely and
; exclusively on Microchip PICmicro Microcontroller products. The
; software is owned by the Company and/or its supplier, and is
; protected under applicable copyright laws. All rights are reserved.
; Any use in violation of the foregoing restrictions may subject the
; user to criminal sanctions under applicable laws, as well as to
; civil liability for the breach of the terms and conditions of this
; license.
;
; THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES,
; WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
; TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
; PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
; IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
; CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;
;*************************************************************
;   PIC18CXX2 EXAMPLE CODE FOR PICDEM-2
;
;   TITLE: A/D Converter Demonstration
;   FILENAME: a2d.asm
;   REVISION HISTORY:  A 5/13/00 jb   format change
;   HARDWARE: PICDEM-2 board
;   FREQUENCY: 4MHz
:
;*************************************************************
; This program is a simple implementation of the
; PIC18C452's A/D.
;
; One Channel is selected (AN0).
; The hardware for this program is the PICDEM-2 board. The program
; converts the potentiometer value on RA0 and displays it as
; an 8 bit binary value on Port B.
;
; The A/D is configured as follows:
; Vref = +5V internal
; A/D Osc. = internal RC
; A/D Channel = AN0 (RA0)

    LIST P=18C452
    #include <P18C452.INC>; File contains addresses for register and bit names
```

```
;*************************************************************
; reset and interrupt vectors

    org 0x00000; Reset Vector Address
    gotoStart

    org 0x00008; Interrupt Vector Address
    gotoISR ; goto Interrupt Service Routine

;*************************************************************
; program code starts here

    org 0x00020
Start
    clrfPORTB; clear all bits of PORTB
    clrfTRISB; Set PORTB as outputs

    callInitializeAD ; configure A/D module

    callSetupDelay; delay for 15 instruction cycles

    bsf ADCON0,GO; Start first A/D conversion

MaingotoMain; do nothing loop

;*************************************************************
; Service A/D interrupt
; Get value and display on LEDs

ISR
    ; Save context (WREG and STATUS) if required.

    btfssPIR1,ADIF; Did A/D cause interrupt?
    gotoOtherInt; No, check other sources

    movfADRESH,W; Get A/D value
    movwfLATB; Display on LEDs
    bcf PIR1,ADIF; Reset A/D int flag

    callSetupDelay; Delay for 15 cycles

    bsf ADCON0,GO; Start A/D conversion

    gotoEndISR; return from ISR

OtherInt
    ; This would be replaced by code to check and service other interrupt sources
```


Source Code

```
    goto$   ; trap here, loops to self

EndISR
    ; Restore context if saved.

    retfie; Return, enables GIE

;*************************************************************
; InitializeAD - initializes and sets up the A/D hardware.
; Select AN0 to AN3 as analog inputs, RC clock, and read AN0.

InitializeAD
    movlwB'00000100'; Make RA0,RA1,RA4 analog inputs
    movwfADCON1

    movlwB'11000001'; Select RC osc, AN0 selected,
    movwfADCON0; A/D enabled

    bcf PIR1,ADIF; Clear A/D interrupt flag
    bsf PIE1,ADIE; Enable A/D interrupt

    bsf INTCON,PEIE; Enable peripheral interrupts
    bsf INTCON,GIE; Enable Global interrupts

    return

;*************************************************************
; This is used to allow the A/D time to sample the input
; (acquisition time).
;
; This routine requires 11 cycles to complete.
; The call and return add another 4 cycles.
;
; 15 cycles with Fosc=4MHz means this delay consumes 15us.

SetupDelay
    movlw.3 ; Load Temp with decimal 3
    movwfTEMP
SD
    decfszTEMP, F; Delay loop
    gotoSD
    return

    END
```

Index

NOTES:

NOTES:

NOTES:

NOTES:

NOTES:

NOTES:

NOTES:

NOTES:

NOTES:

NOTES:

NOTES:

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-786-7200 Fax: 480-786-7277
Technical Support: 480-786-7627
Web Address: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
2 LAN Drive, Suite 120
Westford, MA 01886
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Detroit
Microchip Technology Inc.
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 202
Hauppauge, NY 11788
Tel: 631-273-5305 Fax: 631-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

AMERICAS (continued)

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

China - Beijing
Microchip Technology, Beijing
Unit 915, 6 Chaoyangmen Bei Dajie
Dong Erhuan Road, Dongcheng District
New China Hong Kong Manhattan Building
Beijing, 100027, P.R.C.
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Shanghai
Microchip Technology
Unit B701, Far East International Plaza,
No. 317, Xianxia Road
Shanghai, 200051, P.R.C.
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

Hong Kong
Microchip Asia Pacific
Unit 2101, Tower 2
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore, 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Japan
Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa, 222-0033, Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

ASIA/PACIFIC (continued)

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Denmark
Microchip Technology Denmark ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
Arizona Microchip Technology SARL
Parc d'Activite du Moulin de Massy
43 Rue du Saule Trapu
Batiment A - ler Etage
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-039-65791-1 Fax: 39-039-6899883

United Kingdom
Arizona Microchip Technology Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5858 Fax: 44-118 921-5835

05/16/00

DNV Certification, Inc. USA — ANSI-RAB QMS ACCREDITED — DNV — DNV MSC The Netherlands Accredited by the RvA
ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.