MICROCHIP

PIC18FXX39 Data Sheet

Enhanced FLASH Microcontrollers
with Single Phase Induction
Motor Control Kernel

PIC18FXX39

Enhanced FLASH Microcontrollers with Single Phase Induction Motor Control Kernel

High Performance RISC CPU:

- Linear program memory addressing to 24 Kbytes
- Linear data memory addressing to 1.4 Kbytes
- 20 MHz operation (5 MIPs):
  - 20 MHz oscillator/clock input
  - 5 MHz oscillator/clock input with PLL active
- 16-bit wide instructions, 8-bit wide data path
- 8 x 8 Single Cycle Hardware Multiplier

Special Microcontroller Features:

- 100,000 erase/write cycle Enhanced FLASH program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory
- FLASH/Data EEPROM Retention: > 100 years
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Programmable code protection
- Power saving SLEEP mode
- Single supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins

Analog Features:

- Compatible 10-bit Analog-to-Digital Converter module (A/D) with:
  - Fast sampling rate
  - Conversion available during SLEEP
  - DNL = ±1 LSb, INL = ±1 LSb
- Programmable Low Voltage Detection (PLVD)
  - Supports interrupt on Low Voltage Detection
- Programmable Brown-out Reset (BOR)

Peripheral Features:

- High current sink/source 25 mA/25 mA
- Three external interrupt pins
- Timer0 module: 8-bit/16-bit timer/counter with 8-bit programmable prescaler
- Timer1 module: 16-bit timer/counter
- Timer3 module: 16-bit timer/counter
- Secondary oscillator clock option - Timer1/Timer3
- Two PWM modules:
  - Resolution is 1- to 10-bit,
    Max. PWM freq. @ 8-bit resolution = 156 kHz
    10-bit resolution = 39 kHz
- Single Phase Induction Motor Control kernel
  - Programmable Motor Control Technology (ProMPT™) provides open loop Variable Frequency (VF) control
  - User programmable Voltage vs. Frequency curve
  - Most suitable for shaded pole and permanent split capacitor type motors
- Master Synchronous Serial Port (MSSP) module with two modes of operation:
  - 3-wire SPI™ (supports all 4 SPI modes)
  - I^2C™ Master and Slave mode
- Addressable USART module:
  - Supports RS-485 and RS-232
- Parallel Slave Port (PSP) module

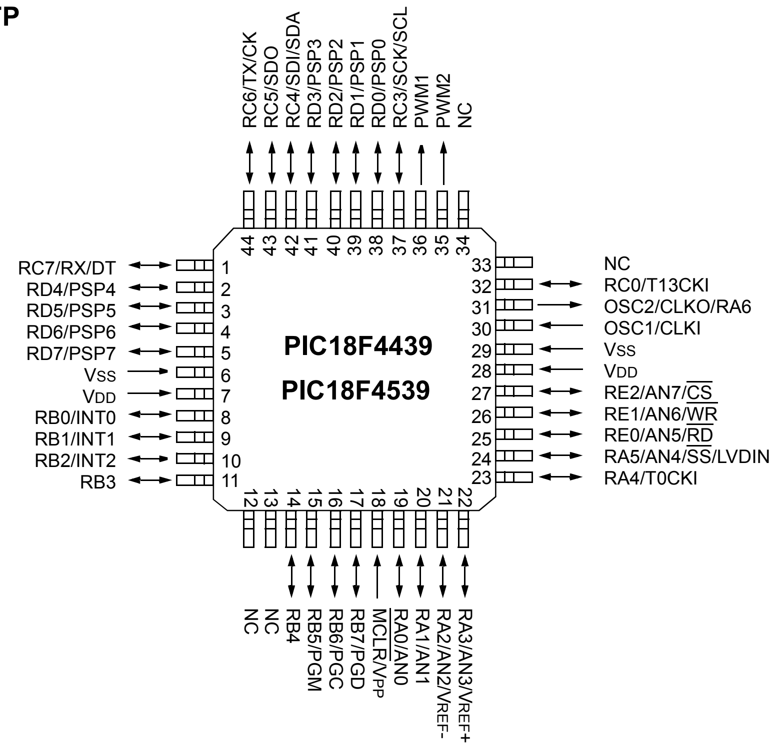
CMOS Technology:

- Low power, high speed FLASH/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Industrial and Extended temperature ranges

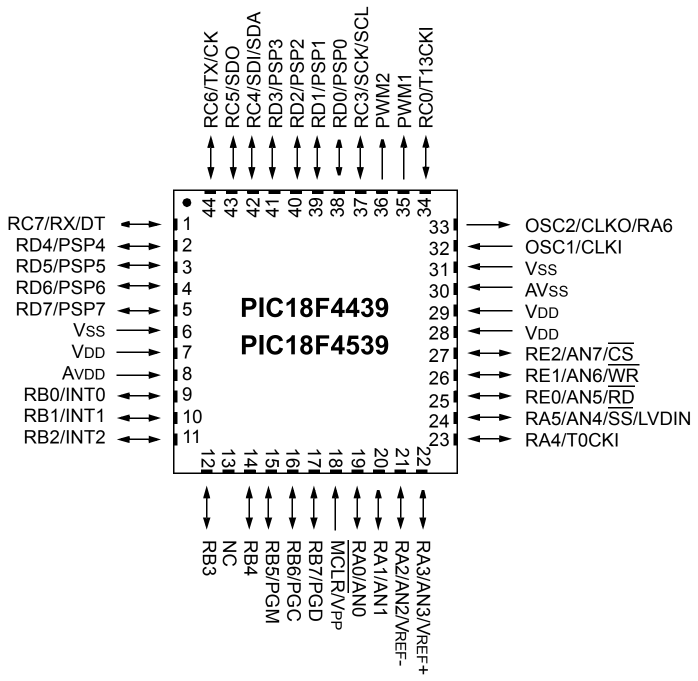
Device	Program Memory		Data Memory		I/O Pins	10-bit A/D (ch)	PWM 10-bit	MSSP		AUSART	Timers 16-bit/WDT
	Bytes	Words	SRAM (Bytes)	EEPROM (Bytes)				SPI	Master I^2C		
PIC18F2439	12K	6144	640	256	21	5	2	Yes	Yes	Yes	3/1
PIC18F2539	24K	12288	1408	256	21	5	2	Yes	Yes	Yes	3/1
PIC18F4439	12K	6144	640	256	32	8	2	Yes	Yes	Yes	3/1
PIC18F4539	24K	12288	1408	256	32	8	2	Yes	Yes	Yes	3/1

Pin Diagrams

44-Pin TQFP

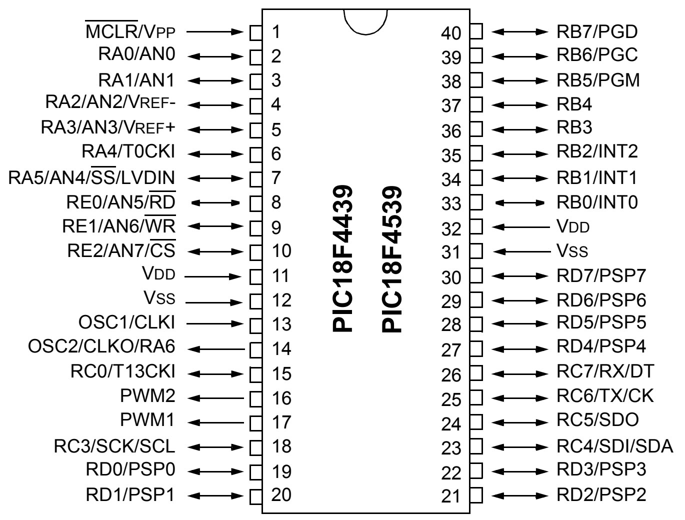


44-Pin QFN



Pin Diagrams (Cont.'d)

40-Pin DIP



28-Pin DIP, SOIC

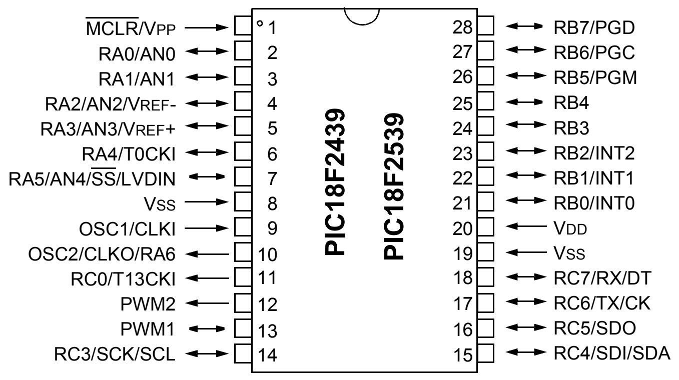


Table of Contents

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via E-mail at **docerrors@mail.microchip.com** or fax the **Reader Response Form** in the back of this data sheet to (480) 792-4150. We welcome your feedback.

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., DS30000A is version A of document DS30000).

Errata

An errata sheet, describing minor operational differences from the data sheet and recommended workarounds, may exist for current devices. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)
- The Microchip Corporate Literature Center; U.S. FAX: (480) 792-7277

When contacting a sales office or the literature center, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Customer Notification System

Register on our web site at **www.microchip.com/cn** to receive the most current information on all of our products.

NOTES:

1.0 DEVICE OVERVIEW

This document contains device specific information for the following devices:

- PIC18F2439
- PIC18F2539
- PIC18F4439
- PIC18F4539

This family offers the advantages of all PIC18 microcontrollers - namely, high computational performance at an economical price - with the addition of high-endurance Enhanced FLASH program memory. The PIC18FXX39 family also provides an off-the-shelf solution for simple motor control applications, allowing users to create speed control solutions with small part counts and short development times.

1.1 Key Features

1.1.1 PROGRAMMABLE MOTOR PROCESSOR TECHNOLOGY (ProMPT™) MOTOR CONTROL

The integrated motor control kernel uses on-chip Pulse Width Modulation (PWM) to provide speed control for single phase induction motors. Through a convenient set of Application Program Interfaces (APIs) and variable frequency technology for open loop control, users can develop applications with little or no previous experience in motor control techniques. ProMPT motor control provides modulated output over a range of 0 to 127 Hz, and has a pre-defined V/F curve that can be reprogrammed to suit the application.

1.1.2 OTHER PIC18FXX39 FEATURES

- **Memory Endurance:** The Enhanced FLASH cells for both program memory and data EEPROM are rated to last for many thousands of erase/write cycles - up to 100,000 for program memory, and 1,000,000 for EEPROM. Data retention without refresh is conservatively estimated to be greater than 100 years at 25°C.
- **Self-programmability:** These devices can write to their own program memory spaces under internal software control. By using a bootloader routine located in the protected Boot Block at the top of program memory, it becomes possible to create an application that can update itself in the field.
- **Addressable USART:** This serial communication module is capable of standard RS-232 operation using the internal oscillator block, removing the need for an external crystal (and its accompanying power requirement) in applications that talk to the outside world.
- **10-bit A/D Converter:** This module offers up to 8 conversion channels for flexibility in sensor monitoring and control, as well as the ability to do conversions while the device is in SLEEP mode.

1.2 Details on Individual Family Members

Devices in the PIC18FXX39 family are available in 28-pin (PIC18F2X39) and 40/44-pin (PIC18F4X39) packages. Block diagrams for the two groups are shown in Figure 1-1 and Figure 1-2.

The devices are differentiated from each other in four ways:

1. FLASH program memory and data RAM (12 Kbytes and 640 bytes for PIC18FX439 devices, 24 Kbytes and 1408 bytes for PIC18FX539)
2. A/D channels (5 for PIC18F2X39 devices, 8 for PIC18F4X39)
3. I/O ports (3 ports on PIC18F2X39, 5 ports on PIC18F4X39 devices)
4. Parallel Slave Port (present only on PIC18F4X39 devices)

All other features for devices in this family are identical. These are summarized in Table 1-1.

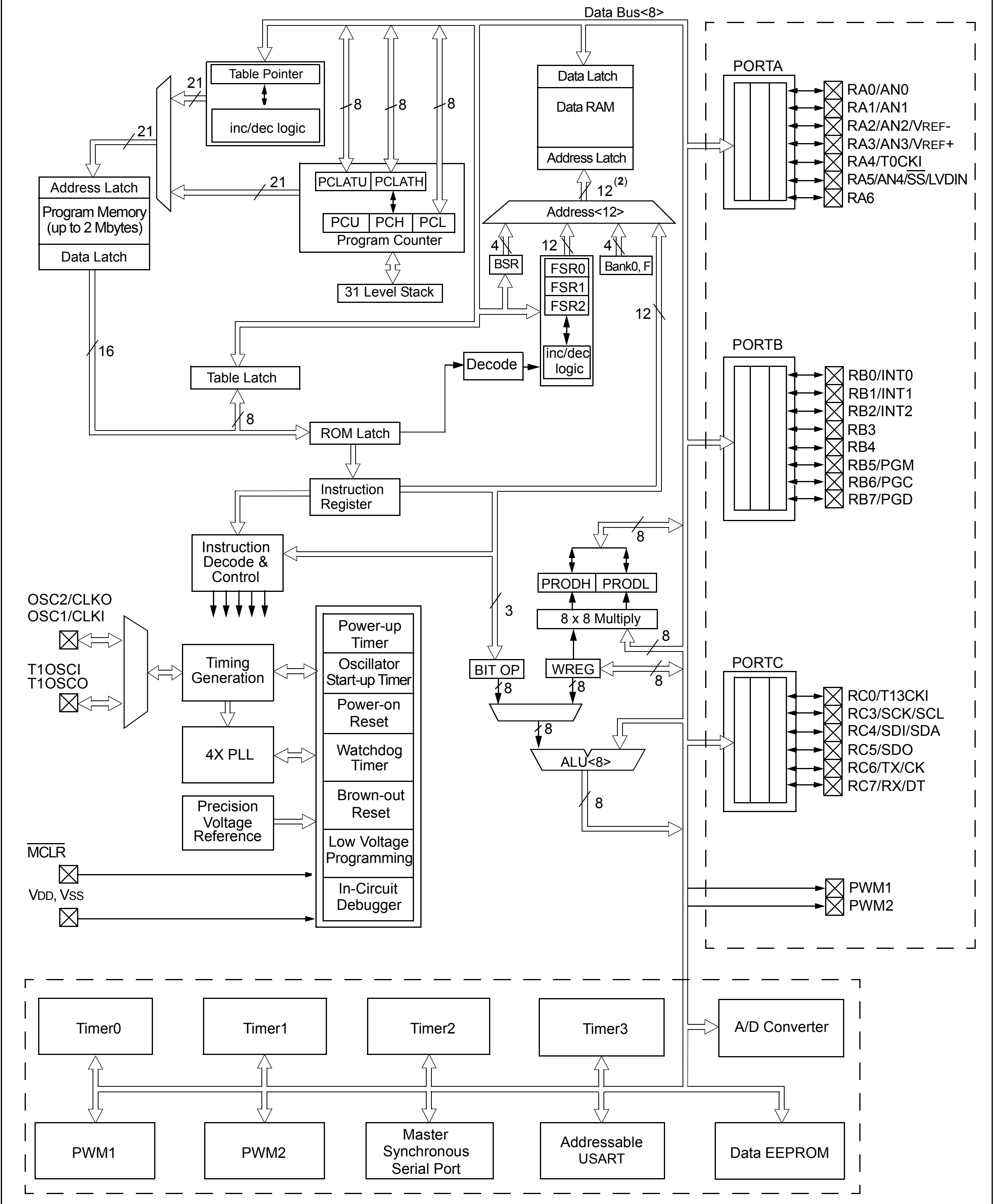
The pinouts for all devices are listed in Table 1-2 and Table 1-3.

TABLE 1-1: PIC18FXX39 DEVICE FEATURES

Features	PIC18F2439	PIC18F2539	PIC18F4439	PIC18F4539
Operating Frequency	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz
Program Memory (Bytes)	12K	24K	12K	24K
Program Memory (Instructions)	6144	12288	6144	12288
Data Memory (Bytes)	640	1408	640	1408
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	15	15	16	16
I/O Ports	Ports A, B, C	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	3	3	3	3
PWM Modules(1)	2	2	2	2
Single Phase Induction Motor Control	Yes	Yes	Yes	Yes
Serial Communications	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART
Parallel Communications	—	—	PSP	PSP
10-bit Analog-to-Digital Module	5 input channels	5 input channels	8 input channels	8 input channels
RESETS (and Delays)	POR, BOR, `RESET` Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, `RESET` Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, `RESET` Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, `RESET` Instruction, Stack Full, Stack Underflow (PWRT, OST)
Programmable Low Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions	75 Instructions	75 Instructions	75 Instructions
Packages	28-pin DIP 28-pin SOIC	28-pin DIP 28-pin SOIC	40-pin DIP 44-pin TQFP 44-pin QFN	40-pin DIP 44-pin TQFP 44-pin QFN

Note 1: PWM modules are used exclusively in conjunction with the motor control kernel, and are not available for other applications.

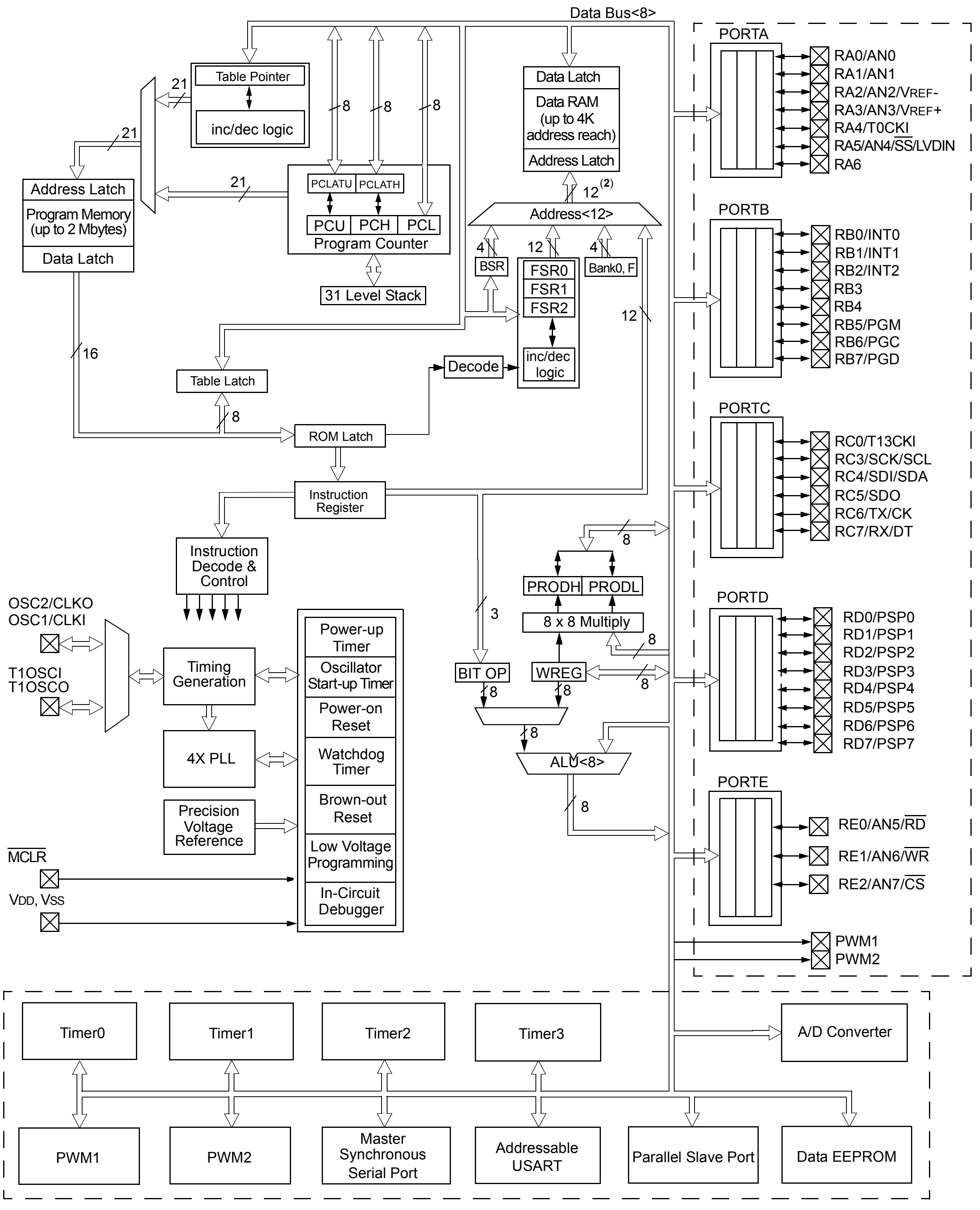
FIGURE 1-1: PIC18F2X39 BLOCK DIAGRAM



Note 1: The high order bits of the Direct Address for the RAM are from the BSR register (except for the `MOVFF` instruction).

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

FIGURE 1-2: PIC18F4X39 BLOCK DIAGRAM



Note 1: The high order bits of the Direct Address for the RAM are from the BSR register (except for the `MOVFF` instruction).

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

TABLE 1-2: PIC18F2X39 PINOUT I/O DESCRIPTIONS

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	DIP	SOIC			
$\overline{\text{MCLR}}$/VPP	1	1			Master Clear (input) or high voltage ICSP programming enable pin.
$\overline{\text{MCLR}}$			I	ST	Master Clear (Reset) input. This pin is an active low RESET to the device.
VPP			I	ST	High voltage ICSP programming enable pin.
NC	—	—	—	—	These pins should be left unconnected.
OSC1/CLKI	9	9			Oscillator crystal or external clock input.
OSC1			I	CMOS	Oscillator crystal input or external clock source input.
CLKI			I	CMOS	External clock source input. Always associated with pin function OSC1. (See related OSC1/CLKI, OSC2/CLKO pins.)
OSC2/CLKO/RA6	10	10			Oscillator crystal or clock output.
OSC2			O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO			O	—	In EC mode, OSC2 pin outputs CLKO which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
RA6			I/O	TTL	General purpose I/O pin.
					PORTA is a bi-directional I/O port.
RA0/AN0	2	2			
RA0			I/O	TTL	Digital I/O.
AN0			I	Analog	Analog input 0.
RA1/AN1	3	3			
RA1			I/O	TTL	Digital I/O.
AN1			I	Analog	Analog input 1.
RA2/AN2/VREF-	4	4			
RA2			I/O	TTL	Digital I/O.
AN2			I	Analog	Analog input 2.
VREF-			I	Analog	A/D Reference Voltage (Low) input.
RA3/AN3/VREF+	5	5			
RA3			I/O	TTL	Digital I/O.
AN3			I	Analog	Analog input 3.
VREF+			I	Analog	A/D Reference Voltage (High) input.
RA4/T0CKI	6	6			
RA4			I/O	ST/OD	Digital I/O. Open drain when configured as output.
T0CKI			I	ST	Timer0 external clock input.
RA5/AN4/$\overline{\text{SS}}$/LVDIN	7	7			
RA5			I/O	TTL	Digital I/O.
AN4			I	Analog	Analog input 4.
$\overline{\text{SS}}$			I	ST	SPI Slave Select input.
LVDIN			I	Analog	Low Voltage Detect input.
RA6					See the OSC2/CLKO/RA6 pin.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels; I = Input
O = Output; P = Power
OD = Open Drain (no P diode to VDD)

TABLE 1-2: PIC18F2X39 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	DIP	SOIC			
					PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs.
RB0/INT0	21	21			
RB0			I/O	TTL	Digital I/O.
INT0			I	ST	External interrupt 0.
RB1/INT1	22	22			
RB1			I/O	TTL	Digital I/O.
INT1			I	ST	External interrupt 1.
RB2/INT2	23	23			
RB2			I/O	TTL	Digital I/O.
INT2			I	ST	External interrupt 2.
RB3	24	24	I/O	TTL	Digital I/O.
RB4	25	25	I/O	TTL	Digital I/O. Interrupt-on-change pin.
RB5/PGM	26	26			
RB5			I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGM			I/O	ST	Low Voltage ICSP programming enable pin.
RB6/PGC	27	27			
RB6			I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGC			I/O	ST	In-Circuit Debugger and ICSP programming clock pin.
RB7/PGD	28	28			
RB7			I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGD			I/O	ST	In-Circuit Debugger and ICSP programming data pin.

Legend:

TTL	= TTL compatible input	CMOS	= CMOS compatible input or output
ST	= Schmitt Trigger input with CMOS levels	I	= Input
O	= Output	P	= Power
OD	= Open Drain (no P diode to V_{DD})		

TABLE 1-2: PIC18F2X39 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number		Pin Type	Buffer Type	Description
	DIP	SOIC			
					PORTC is a bi-directional I/O port.
RC0/T13CKI	11	11			
RC0			I/O	ST	Digital I/O.
T13CKI			I	ST	Timer1/Timer3 external clock input.
RC3/SCK/SCL	14	14			
RC3			I/O	ST	Digital I/O.
SCK			I/O	ST	Synchronous serial clock input/output for SPI mode.
SCL			I/O	ST	Synchronous serial clock input/output for I^{2}C mode.
RC4/SDI/SDA	15	15			
RC4			I/O	ST	Digital I/O.
SDI			I	ST	SPI Data in.
SDA			I/O	ST	I^{2}C Data I/O.
RC5/SDO	16	16			
RC5			I/O	ST	Digital I/O.
SDO			O	—	SPI Data out.
RC6/TX/CK	17	17			
RC6			I/O	ST	Digital I/O.
TX			O	—	USART Asynchronous Transmit.
CK			I/O	ST	USART Synchronous Clock (see related RX/DT).
RC7/RX/DT	18	18			
RC7			I/O	ST	Digital I/O.
RX			I	ST	USART Asynchronous Receive.
DT			I/O	ST	USART Synchronous Data (see related TX/CK).
PWM1	13	13	O	—	PWM Channel 1 (motor control) output.
PWM2	12	12	O	—	PWM Channel 2 (motor control) output.
VSS	8, 19	8, 19	P	—	Ground reference for logic and I/O pins.
VDD	20	20	P	—	Positive supply for logic and I/O pins.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output; ST = Schmitt Trigger input with CMOS levels; I = Input; O = Output; P = Power; OD = Open Drain (no P diode to VDD)

TABLE 1-3: PIC18F4X39 PINOUT I/O DESCRIPTIONS

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	DIP	QFN	TQFP			
$\overline{\text{MCLR}}$/VPP	1	18	18			Master Clear (input) or high voltage ICSP programming enable pin.
$\overline{\text{MCLR}}$				I	ST	Master Clear (Reset) input. This pin is an active low RESET to the device.
VPP				I	ST	High voltage ICSP programming enable pin.
OSC1/CLKI	13	32	30			Oscillator crystal or external clock input.
OSC1				I	CMOS	Oscillator crystal input or external clock source input.
CLKI				I	CMOS	External clock source input. Always associated with pin function OSC1. (See related OSC1/CLKI, OSC2/CLKO pins.)
OSC2/CLKO/RA6	14	33	31			Oscillator crystal or clock output.
OSC2				O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO				O	—	In EC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
RA6				I/O	TTL	General purpose I/O pin.
						PORTA is a bi-directional I/O port.
RA0/AN0	2	19	19			
RA0				I/O	TTL	Digital I/O.
AN0				I	Analog	Analog input 0.
RA1/AN1	3	20	20			
RA1				I/O	TTL	Digital I/O.
AN1				I	Analog	Analog input 1.
RA2/AN2/VREF-	4	21	21			
RA2				I/O	TTL	Digital I/O.
AN2				I	Analog	Analog input 2.
VREF-				I	Analog	A/D Reference Voltage (Low) input.
RA3/AN3/VREF+	5	22	22			
RA3				I/O	TTL	Digital I/O.
AN3				I	Analog	Analog input 3.
VREF+				I	Analog	A/D Reference Voltage (High) input.
RA4/T0CKI	6	23	23			
RA4				I/O	ST/OD	Digital I/O. Open drain when configured as output.
T0CKI				I	ST	Timer0 external clock input.
RA5/AN4/$\overline{\text{SS}}$/LVDIN	7	24	24			
RA5				I/O	TTL	Digital I/O.
AN4				I	Analog	Analog input 4.
$\overline{\text{SS}}$				I	ST	SPI Slave Select input.
LVDIN				I	Analog	Low Voltage Detect input.
RA6						(See the OSC2/CLKO/RA6 pin.)

Legend:

TTL	= TTL compatible input	CMOS	= CMOS compatible input or output
ST	= Schmitt Trigger input with CMOS levels	I	= Input
O	= Output	P	= Power
OD	= Open Drain (no P diode to VDD)		

TABLE 1-3: PIC18F4X39 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	DIP	QFN	TQFP			
						PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs.
RB0/INT0	33	9	8			
RB0				I/O	TTL	Digital I/O.
INT0				I	ST	External interrupt 0.
RB1/INT1	34	10	9			
RB1				I/O	TTL	Digital I/O.
INT1				I	ST	External interrupt 1.
RB2/INT2	35	11	10			
RB2				I/O	TTL	Digital I/O.
INT2				I	ST	External interrupt 2.
RB3	36	12	11	I/O	TTL	Digital I/O.
RB4	37	14	14	I/O	TTL	Digital I/O. Interrupt-on-change pin.
RB5/PGM	38	15	15			
RB5				I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGM				I/O	ST	Low Voltage ICSP programming enable pin.
RB6/PGC	39	16	16			
RB6				I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGC				I/O	ST	In-Circuit Debugger and ICSP programming clock pin.
RB7/PGD	40	17	17			
RB7				I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGD				I/O	ST	In-Circuit Debugger and ICSP programming data pin.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels; I = Input
O = Output; P = Power
OD = Open Drain (no P diode to V_{DD})

TABLE 1-3: PIC18F4X39 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	DIP	QFN	TQFP			
						PORTC is a bi-directional I/O port.
RC0/T13CKI	15	34	32			
RC0				I/O	ST	Digital I/O.
T13CKI				I	ST	Timer1/Timer3 external clock input.
RC3/SCK/SCL	18	37	37			
RC3				I/O	ST	Digital I/O.
SCK				I/O	ST	Synchronous serial clock input/output for SPI mode.
SCL				I/O	ST	Synchronous serial clock input/output for I^{2}C mode.
RC4/SDI/SDA	23	42	42			
RC4				I/O	ST	Digital I/O.
SDI				I	ST	SPI Data in.
SDA				I/O	ST	I^{2}C Data I/O.
RC5/SDO	24	43	43			
RC5				I/O	ST	Digital I/O.
SDO				O	—	SPI Data out.
RC6/TX/CK	25	44	44			
RC6				I/O	ST	Digital I/O.
TX				O	—	USART Asynchronous Transmit.
CK				I/O	ST	USART Synchronous Clock (see related RX/DT).
RC7/RX/DT	26	1	1			
RC7				I/O	ST	Digital I/O.
RX				I	ST	USART Asynchronous Receive.
DT				I/O	ST	USART Synchronous Data (see related TX/CK).
PWM1	17	35	36	O	—	PWM Channel 1 (motor control) output.
PWM2	16	36	35	O	—	PWM Channel 2 (motor control) output.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels; I = Input
O = Output; P = Power
OD = Open Drain (no P diode to VDD)

TABLE 1-3: PIC18F4X39 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	DIP	QFN	TQFP			
						PORTD is a bi-directional I/O port, or a Parallel Slave Port (PSP) for interfacing to a microprocessor port. These pins have TTL input buffers when PSP module is enabled.
RD0/PSP0	19	38	38	I/O		
RD0					ST	Digital I/O.
PSP0					TTL	Parallel Slave Port Data.
RD1/PSP1	20	39	39	I/O		
RD1					ST	Digital I/O.
PSP1					TTL	Parallel Slave Port Data.
RD2/PSP2	21	40	40	I/O		
RD2					ST	Digital I/O.
PSP2					TTL	Parallel Slave Port Data.
RD3/PSP3	22	41	41	I/O		
RD3					ST	Digital I/O.
PSP3					TTL	Parallel Slave Port Data.
RD4/PSP4	27	2	2	I/O		
RD4					ST	Digital I/O.
PSP4					TTL	Parallel Slave Port Data.
RD5/PSP5	28	3	3	I/O		
RD5					ST	Digital I/O.
PSP5					TTL	Parallel Slave Port Data.
RD6/PSP6	29	4	4	I/O		
RD6					ST	Digital I/O.
PSP6					TTL	Parallel Slave Port Data.
RD7/PSP7	30	5	5	I/O		
RD7					ST	Digital I/O.
PSP7					TTL	Parallel Slave Port Data.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels; I = Input
O = Output; P = Power
OD = Open Drain (no P diode to V_{DD})

TABLE 1-3: PIC18F4X39 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	DIP	QFN	TQFP			
						PORTE is a bi-directional I/O port.
RE0/$\overline{\text{RD}}$/AN5	8	25	25	I/O		
RE0					ST	Digital I/O.
$\overline{\text{RD}}$					TTL	Read control for parallel slave port (see also $\overline{\text{WR}}$ and $\overline{\text{CS}}$ pins).
AN5					Analog	Analog input 5.
RE1/$\overline{\text{WR}}$/AN6	9	26	26	I/O		
RE1					ST	Digital I/O.
$\overline{\text{WR}}$					TTL	Write control for parallel slave port (see $\overline{\text{CS}}$ and $\overline{\text{RD}}$ pins).
AN6					Analog	Analog input 6.
RE2/$\overline{\text{CS}}$/AN7	10	27	27	I/O		
RE2					ST	Digital I/O.
$\overline{\text{CS}}$					TTL	Chip Select control for parallel slave port (see related $\overline{\text{RD}}$ and $\overline{\text{WR}}$).
AN7					Analog	Analog input 7.
VSS	12, 31	6, 31	6, 29	P	—	Ground reference for logic and I/O pins.
VDD	11, 32	7, 28, 29	7, 28	P	—	Positive supply for logic and I/O pins.
AVSS	—	30	—	P	—	Ground reference for analog modules.
AVDD	—	8	—	P	—	Positive supply for analog modules.
NC	—	13	12, 13, 33, 34	—	—	These pins should be left unconnected.

Legend: TTL = TTL compatible input — CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels — I = Input
O = Output — P = Power
OD = Open Drain (no P diode to VDD)

2.0 OSCILLATOR CONFIGURATIONS

2.1 Oscillator Types

The PIC18FXX39 can be operated in four different Oscillator modes at a frequency of 20 MHz. The user can program three configuration bits (FOSC2, FOSC1, and FOSC0) to select one of these four modes:

1.	HS	High Speed Crystal/Resonator
2.	HS + PLL	High Speed Crystal/Resonator with PLL enabled using 5 MHz crystal
3.	EC	External Clock
4.	ECIO	External Clock with I/O pin enabled

Note: The operation of the Motor Control kernel and its APIs (Section 14.0) is based on an assumed clock frequency of 20 MHz. Changing the oscillator frequency will change the timing used in the Motor Control kernel accordingly. To achieve the best results in motor control applications, a clock frequency of 20 MHz is highly recommended.

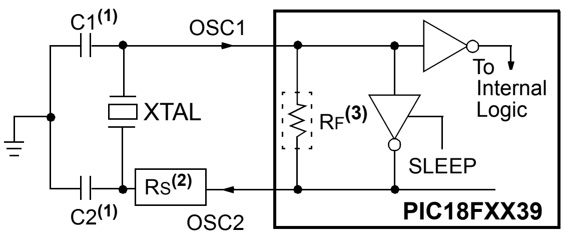
2.2 Crystal Oscillator/Ceramic Resonators

In HS or HS+PLL Oscillator modes, a crystal or ceramic resonator is connected to the OSC1 and OSC2 pins to establish oscillation. Figure 2-1 shows the pin connections.

The PIC18FXX39 oscillator design requires the use of a parallel cut crystal.

Note: Use of a series cut crystal may give a frequency out of the crystal manufacturers specifications.

FIGURE 2-1: CRYSTAL/CERAMIC RESONATOR OPERATION (HS CONFIGURATION)



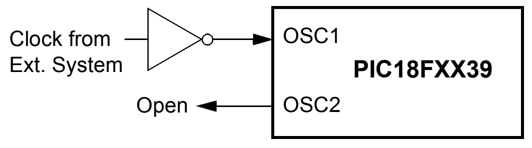
Note 1: See Table 2-1 for recommended values of C1 and C2.

2: A series resistor (RS) may be required for AT strip cut crystals.

3: RF varies with the Oscillator mode chosen.

An external clock source may also be connected to the OSC1 pin in the HS mode, as shown in Figure 2-2.

FIGURE 2-2: EXTERNAL CLOCK INPUT OPERATION (HS OSC CONFIGURATION)



Note 1: Higher capacitance increases the stability of the 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, or verify oscillator performance.

TABLE 2-1: CAPACITOR SELECTION FOR CRYSTAL OSCILLATOR

Ranges Tested:			
Mode	**Freq**	**C1**	**C2**
HS	20.0 MHz	15-33 pF	15-33 pF
These values are for design guidance only. See notes following this table.			
Crystals Used			
20.0 MHz	Epson CA-301 20.000M-C		± 30 PPM

Note 1: Higher capacitance increases the stability of the oscillator, but also increases the start-up time.

2: Rs may be required in HS mode to avoid overdriving crystals with low drive level specification.

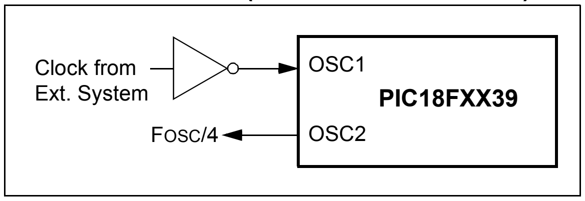
3: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components, or verify oscillator performance.

2.3 External Clock Input

The EC and ECIO Oscillator modes require an external clock source to be connected to the OSC1 pin. The feedback device between OSC1 and OSC2 is turned off in these modes to save current. There is no oscillator start-up time required after a Power-on Reset or after a recovery from SLEEP mode.

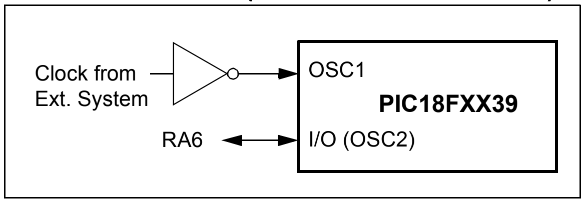
In the EC Oscillator mode, the oscillator frequency divided by 4 is available on the OSC2 pin. This signal may be used for test purposes or to synchronize other logic. Figure 2-3 shows the pin connections for the EC Oscillator mode.

FIGURE 2-3: EXTERNAL CLOCK INPUT OPERATION (EC CONFIGURATION)



The ECIO Oscillator mode functions like the EC mode, except that the OSC2 pin becomes an additional general purpose I/O pin. The I/O pin becomes bit 6 of PORTA (RA6). Figure 2-4 shows the pin connections for the ECIO Oscillator mode.

FIGURE 2-4: EXTERNAL CLOCK INPUT OPERATION (ECIO CONFIGURATION)



2.4 HS/PLL

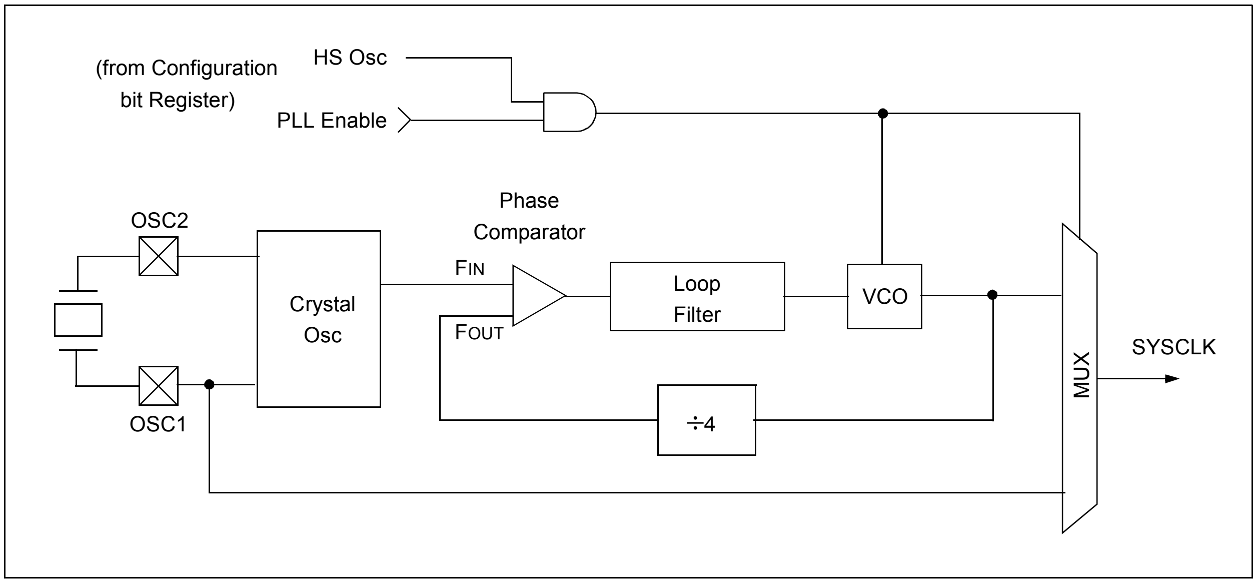
A Phase Locked Loop circuit is provided as a programmable option for users that want to multiply the frequency of the incoming crystal oscillator signal by 4. For an input clock frequency of 5 MHz, the internal clock frequency will be multiplied to 20 MHz. This is useful for customers who are concerned with EMI due to high frequency crystals.

The PLL can only be enabled when the oscillator configuration bits are programmed for HS mode. If they are programmed for any other mode, the PLL is not enabled and the system clock will come directly from OSC1.

The PLL is one of the modes specified by the FOSC<2:0> configuration bits. The Oscillator mode is specified during device programming.

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.

FIGURE 2-5: PLL BLOCK DIAGRAM



2.5 Effects of SLEEP Mode on the On-Chip Oscillator

When the device executes a `SLEEP` instruction, the oscillator is 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 during SLEEP. The user can wake from SLEEP through external RESET, Watchdog Timer Reset, or through an interrupt.

2.6 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 Section 3.0.

The first timer is the Power-up Timer (PWRT), which optionally provides a fixed delay of 72 ms (nominal) on power-up only (POR and BOR). The second timer is the Oscillator Start-up Timer (OST), intended to keep the chip in RESET until the crystal oscillator is stable.

With the PLL enabled (HS/PLL Oscillator mode), the time-out sequence following a Power-on Reset is different from other Oscillator modes. The time-out sequence is as follows:

1. The PWRT time-out is invoked after a POR time delay has expired.
2. The Oscillator Start-up Timer (OST) is invoked. However, this is still not a sufficient amount of time to allow the PLL to lock at high frequencies.
3. The PWRT timer is used to provide an additional fixed 2 ms (nominal) time-out to allow the PLL ample time to lock to the incoming clock frequency.

TABLE 2-2: OSC1 AND OSC2 PIN STATES IN SLEEP MODE

OSC Mode	OSC1 Pin	OSC2 Pin
ECIO	Floating	Configured as PORTA, bit 6
EC	Floating	At logic low
HS	Feedback inverter disabled, at quiescent voltage level	Feedback inverter disabled, at quiescent voltage level

Note: See Table 3-1 in the **“Reset”** section, for time-outs due to SLEEP and $\overline{\text{MCLR}}$ Reset.

NOTES:

3.0 RESET

The PIC18FXX39 differentiates 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) Watchdog Timer (WDT) Reset (during normal operation)
e) Programmable Brown-out Reset (BOR)
f) `RESET` Instruction
g) Stack Full 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.

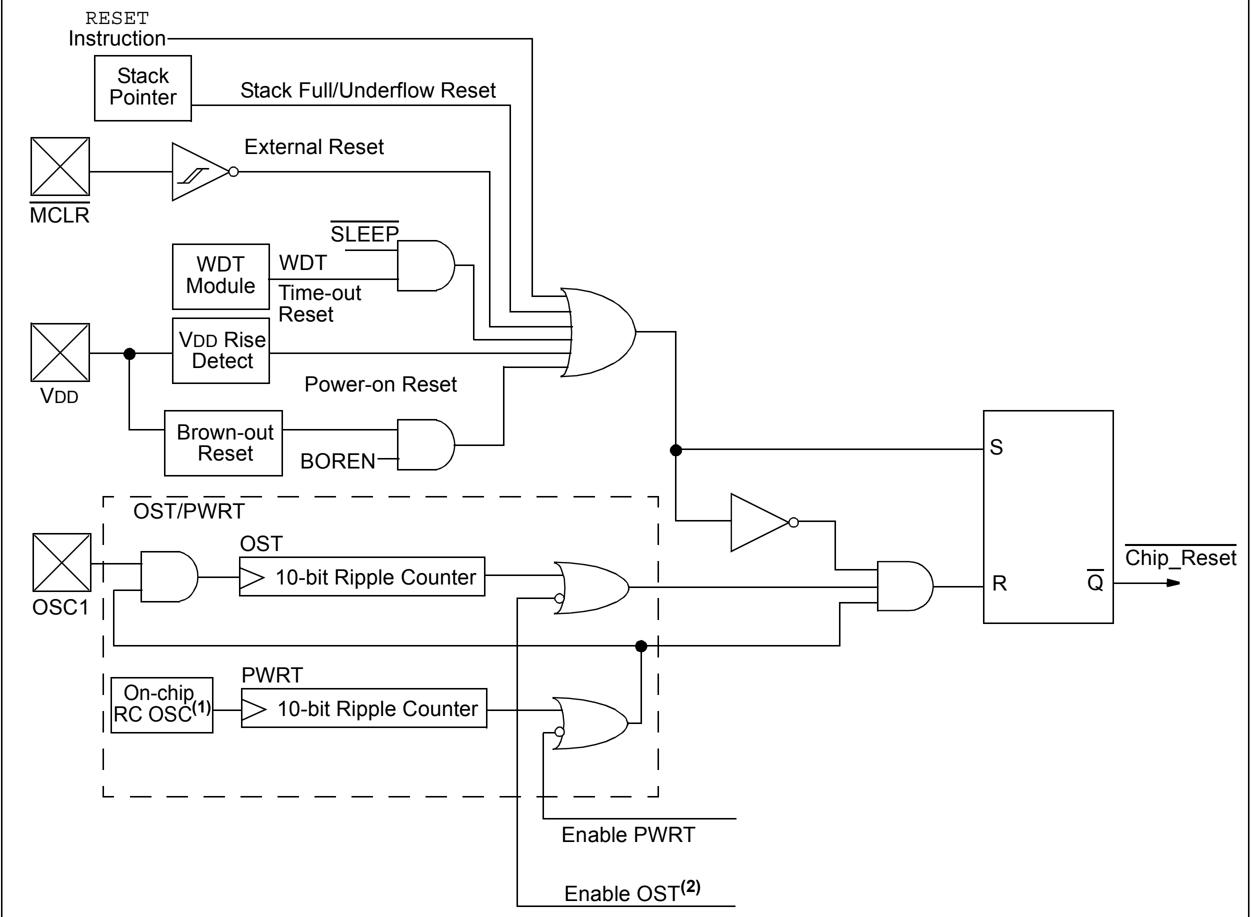
Most registers are not affected by a WDT wake-up, since this is viewed as the resumption of normal operation. Status bits from the RCON register, $\overline{\text{RI}}$, $\overline{\text{TO}}$, $\overline{\text{PD}}$, $\overline{\text{POR}}$ and $\overline{\text{BOR}}$, are set or cleared differently in different RESET situations, as indicated in Table 3-2. These bits are used in software to determine the nature of the RESET. See Table 3-3 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.

The Enhanced MCU devices have a $\overline{\text{MCLR}}$ noise filter in the $\overline{\text{MCLR}}$ Reset path. The filter will detect and ignore small pulses.

The $\overline{\text{MCLR}}$ pin is not driven low by any internal RESETS, including the WDT.

FIGURE 3-1: SIMPLIFIED BLOCK DIAGRAM OF ON-CHIP RESET CIRCUIT



Note 1: This is a separate oscillator from the RC oscillator of the CLKI pin.

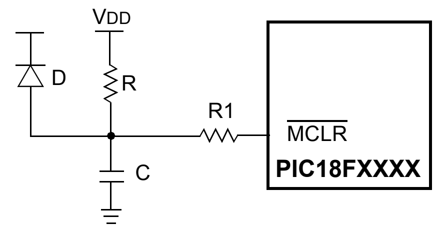
2: See Table 3-1 for time-out situations.

3.1 Power-on Reset (POR)

A Power-on Reset pulse is generated on-chip when VDD rise is detected. To take advantage of the POR circuitry, just tie the $\overline{\text{MCLR}}$ pin directly (or through a resistor) to VDD. This will eliminate external RC components usually needed to create a Power-on Reset delay. A minimum rise rate for VDD is specified (parameter D004). For a slow rise time, see Figure 3-2.

When the device starts normal operation (i.e., exits the RESET condition), device operating parameters (voltage, frequency, temperature, etc.) must be met to ensure operation. If these conditions are not met, the device must be held in RESET until the operating conditions are met.

FIGURE 3-2: EXTERNAL POWER-ON RESET CIRCUIT (FOR SLOW VDD POWER-UP)



Note 1: External Power-on Reset circuit is required only if the VDD power-up slope is too slow. The diode D helps discharge the capacitor quickly when VDD powers down.

2: R < 40 kΩ is recommended to make sure that the voltage drop across R does not violate the device's electrical specification.

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

3.2 Power-up Timer (PWRT)

The Power-up Timer provides a fixed nominal time-out (parameter 33) only on power-up from the POR. The Power-up Timer operates on an internal RC oscillator. The chip is kept in RESET as long as the PWRT is active. The PWRT's time delay allows VDD to rise to an acceptable level. A configuration bit is provided to enable/disable the PWRT.

The power-up time delay will vary from chip-to-chip due to VDD, temperature and process variation. See DC parameter D033 for details.

3.3 Oscillator Start-up Timer (OST)

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

The OST time-out is invoked only for XT, LP and HS modes and only on Power-on Reset or wake-up from SLEEP.

3.4 PLL Lock Time-out

With the PLL 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) is typically 2 ms and follows the Oscillator Start-up Time-out (OST).

3.5 Brown-out Reset (BOR)

A configuration bit, BOREN, can disable (if clear/programmed), or enable (if set) the Brown-out Reset circuitry. If VDD falls below parameter D005 for greater than parameter 35, the brown-out situation will reset the chip. A RESET may not occur if VDD falls below parameter D005 for less than parameter 35. The chip will remain in Brown-out Reset until VDD rises above BVDD. If the Power-up Timer is enabled, it will be invoked after VDD rises above BVDD; it then will keep the chip in RESET for an additional time delay (parameter 33). If VDD drops below BVDD while the Power-up Timer is running, the chip will go back into a Brown-out Reset and the Power-up Timer will be initialized. Once VDD rises above BVDD, the Power-up Timer will execute the additional time delay.

3.6 Time-out Sequence

On power-up, the time-out sequence is as follows: First, PWRT time-out is invoked after the POR time delay has expired. Then, OST is activated. The total time-out will vary based on oscillator configuration and the status of the PWRT. For example, in RC mode with the PWRT disabled, there will be no time-out at all. Figure 3-3, Figure 3-4, Figure 3-5, Figure 3-6 and Figure 3-7 depict time-out sequences on power-up.

Since the time-outs occur from the 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 3-5). This is useful for testing purposes or to synchronize more than one PIC18FXXX device operating in parallel.

Table 3-2 shows the RESET conditions for some Special Function Registers, while Table 3-3 shows the RESET conditions for all the registers.

TABLE 3-1: TIME-OUT IN VARIOUS SITUATIONS

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

Note 1: 2 ms is the nominal time required for the 4x PLL to lock.
2: 72 ms is the nominal power-up timer delay, if implemented.

REGISTER 3-1: RCON REGISTER BITS AND POSITIONS

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

Note 1: Refer to Section 4.14 (page 50) for bit definitions.

TABLE 3-2: STATUS BITS, THEIR SIGNIFICANCE AND THE INITIALIZATION CONDITION FOR RCON REGISTER

Condition	Program Counter	RCON Register	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$	STKFUL	STKUNF
Power-on Reset	0000h	`0--1 1100`	1	1	1	0	0	u	u
$\overline{\text{MCLR}}$ Reset during normal operation	0000h	`0--u uuuu`	u	u	u	u	u	u	u
Software Reset during normal operation	0000h	`0--0 uuuu`	0	u	u	u	u	u	u
Stack Full Reset during normal operation	0000h	`0--u uu11`	u	u	u	u	u	u	1
Stack Underflow Reset during normal operation	0000h	`0--u uu11`	u	u	u	u	u	1	u
$\overline{\text{MCLR}}$ Reset during SLEEP	0000h	`0--u 10uu`	u	1	0	u	u	u	u
WDT Reset	0000h	`0--u 01uu`	1	0	1	u	u	u	u
WDT Wake-up	PC + 2	`u--u 00uu`	u	0	0	u	u	u	u
Brown-out Reset	0000h	`0--1 11u0`	1	1	1	1	0	u	u
Interrupt wake-up from SLEEP	PC + 2[(1)]	`u--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 (0x000008h or 0x000018h).

TABLE 3-3: INITIALIZATION CONDITIONS FOR ALL REGISTERS

Register	Applicable Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets WDT Reset `RESET` Instruction Stack Resets	Wake-up via WDT or Interrupt
TOSU	2439	4439	2539	4539	`---0 0000`	`---0 0000`	`---0 uuuu`(1)
TOSH	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`(1)
TOSL	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`(1)
STKPTR	2439	4439	2539	4539	`00-0 0000`	`uu-0 0000`	`uu-u uuuu`(1)
PCLATU	2439	4439	2539	4539	`---0 0000`	`---0 0000`	`---u uuuu`
PCLATH	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
PCL	2439	4439	2539	4539	`0000 0000`	`0000 0000`	PC + 2(2)
TBLPTRU	2439	4439	2539	4539	`--00 0000`	`--00 0000`	`--uu uuuu`
TBLPTRH	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
TBLPTRL	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
TABLAT	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
PRODH	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
PRODL	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
INTCON	2439	4439	2539	4539	`0000 000x`	`0000 000u`	`uuuu uuuu`(3)
INTCON2	2439	4439	2539	4539	`1111 -1-1`	`1111 -1-1`	`uuuu -u-u`(3)
INTCON3	2439	4439	2539	4539	`11-0 0-00`	`11-0 0-00`	`uu-u u-uu`(3)
INDF0	2439	4439	2539	4539	N/A	N/A	N/A
POSTINC0	2439	4439	2539	4539	N/A	N/A	N/A
POSTDEC0	2439	4439	2539	4539	N/A	N/A	N/A
PREINC0	2439	4439	2539	4539	N/A	N/A	N/A
PLUSW0	2439	4439	2539	4539	N/A	N/A	N/A
FSR0H	2439	4439	2539	4539	`---- xxxx`	`---- uuuu`	`---- uuuu`
FSR0L	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
WREG	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
INDF1	2439	4439	2539	4539	N/A	N/A	N/A
POSTINC1	2439	4439	2539	4539	N/A	N/A	N/A
POSTDEC1	2439	4439	2539	4539	N/A	N/A	N/A
PREINC1	2439	4439	2539	4539	N/A	N/A	N/A
PLUSW1	2439	4439	2539	4539	N/A	N/A	N/A

Legend: `u` = unchanged, `x` = unknown, `-` = unimplemented bit, read as '0', `q` = value depends on condition. Shaded cells indicate conditions do not apply for the designated device.

* These registers are retained to maintain compatibility with PIC18FXX2 devices; however, one or more bits are reserved. Users should not modify the value of these bits. See Section 4.9.2 for details.

Note 1: 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.

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

4: See Table 3-2 for RESET value for specific condition.

5: Bit 6 of PORTA, LATA, and TRISA are enabled in ECIO and RCIO Oscillator modes only. In all other Oscillator modes, they are disabled and read '0'.

6: Bit 6 of PORTA, LATA and TRISA are not available on all devices. When unimplemented, they are read '0'.

TABLE 3-3: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Applicable Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets WDT Reset `RESET` Instruction Stack Resets	Wake-up via WDT or Interrupt
FSR1H	2439	4439	2539	4539	`---- xxxx`	`---- uuuu`	`---- uuuu`
FSR1L	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
BSR	2439	4439	2539	4539	`---- 0000`	`---- 0000`	`---- uuuu`
INDF2	2439	4439	2539	4539	N/A	N/A	N/A
POSTINC2	2439	4439	2539	4539	N/A	N/A	N/A
POSTDEC2	2439	4439	2539	4539	N/A	N/A	N/A
PREINC2	2439	4439	2539	4539	N/A	N/A	N/A
PLUSW2	2439	4439	2539	4539	N/A	N/A	N/A
FSR2H	2439	4439	2539	4539	`---- xxxx`	`---- uuuu`	`---- uuuu`
FSR2L	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
STATUS	2439	4439	2539	4539	`---x xxxx`	`---u uuuu`	`---u uuuu`
TMR0H	2439	4439	2539	4539	`0000 0000`	`uuuu uuuu`	`uuuu uuuu`
TMR0L	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
T0CON	2439	4439	2539	4539	`1111 1111`	`1111 1111`	`uuuu uuuu`
OSCCON*	2439	4439	2539	4539	`---- ---0`	`---- ---0`	`---- ---u`
LVDCON	2439	4439	2539	4539	`--00 0101`	`--00 0101`	`--uu uuuu`
WDTCON	2439	4439	2539	4539	`---- ---0`	`---- ---0`	`---- ---u`
RCON(4)	2439	4439	2539	4539	`0--q 11qq`	`0--q qquu`	`u--u qquu`
TMR1H	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
TMR1L	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
T1CON	2439	4439	2539	4539	`0-00 0000`	`u-uu uuuu`	`u-uu uuuu`
TMR2*	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
PR2*	2439	4439	2539	4539	`1111 1111`	`1111 1111`	`1111 1111`
T2CON*	2439	4439	2539	4539	`-000 0000`	`-000 0000`	`-uuu uuuu`
SSPBUF	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
SSPADD	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
SSPSTAT	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
SSPCON1	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
SSPCON2	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`

Legend: `u` = unchanged, `x` = unknown, `-` = unimplemented bit, read as '0', `q` = value depends on condition. Shaded cells indicate conditions do not apply for the designated device.

* These registers are retained to maintain compatibility with PIC18FXX2 devices; however, one or more bits are reserved. Users should not modify the value of these bits. See Section 4.9.2 for details.

Note 1: 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.

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

4: See Table 3-2 for RESET value for specific condition.

5: Bit 6 of PORTA, LATA, and TRISA are enabled in ECIO and RCIO Oscillator modes only. In all other Oscillator modes, they are disabled and read '0'.

6: Bit 6 of PORTA, LATA and TRISA are not available on all devices. When unimplemented, they are read '0'.

TABLE 3-3: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Applicable Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets WDT Reset `RESET` Instruction Stack Resets	Wake-up via WDT or Interrupt
ADRESH	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
ADRESL	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
ADCON0	2439	4439	2539	4539	`0000 00-0`	`0000 00-0`	`uuuu uu-u`
ADCON1	2439	4439	2539	4539	`00-- 0000`	`00-- 0000`	`uu-- uuuu`
CCPR1H	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
CCPR1L*	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
CCP1CON*	2439	4439	2539	4539	`--00 0000`	`--00 0000`	`--uu uuuu`
CCPR2H	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
CCPR2L*	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
CCP2CON*	2439	4439	2539	4539	`--00 0000`	`--00 0000`	`--uu uuuu`
TMR3H	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
TMR3L	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
T3CON	2439	4439	2539	4539	`0000 0000`	`uuuu uuuu`	`uuuu uuuu`
SPBRG	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
RCREG	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
TXREG	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
TXSTA	2439	4439	2539	4539	`0000 -010`	`0000 -010`	`uuuu -uuu`
RCSTA	2439	4439	2539	4539	`0000 000x`	`0000 000x`	`uuuu uuuu`
EEADR	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
EEDATA	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
EECON1	2439	4439	2539	4539	`xx-0 x000`	`uu-0 u000`	`uu-0 u000`
EECON2	2439	4439	2539	4539	`---- ----`	`---- ----`	`---- ----`

Legend: `u` = unchanged, `x` = unknown, `-` = unimplemented bit, read as '0', `q` = value depends on condition.
Shaded cells indicate conditions do not apply for the designated device.
* These registers are retained to maintain compatibility with PIC18FXX2 devices; however, one or more bits are reserved. Users should not modify the value of these bits. See Section 4.9.2 for details.

Note 1: 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.
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: One or more bits in the INTCONx or PIRx registers will be affected (to cause wake-up).
4: See Table 3-2 for RESET value for specific condition.
5: Bit 6 of PORTA, LATA, and TRISA are enabled in ECIO and RCIO Oscillator modes only. In all other Oscillator modes, they are disabled and read ‘0’.
6: Bit 6 of PORTA, LATA and TRISA are not available on all devices. When unimplemented, they are read ‘0’.

TABLE 3-3: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Applicable Devices				Power-on Reset, Brown-out Reset	$\overline{\text{MCLR}}$ Resets WDT Reset `RESET` Instruction Stack Resets	Wake-up via WDT or Interrupt
IPR2	2439	4439	2539	4539	`---1 1111`	`---1 1111`	`---u uuuu`
PIR2	2439	4439	2539	4539	`---0 0000`	`---0 0000`	`---u uuuu`(3)
PIE2	2439	4439	2539	4539	`---0 0000`	`---0 0000`	`---u uuuu`
IPR1	2439	4439	2539	4539	`1111 1111`	`1111 1111`	`uuuu uuuu`
	2439	4439	2539	4539	`-111 1111`	`-111 1111`	`-uuu uuuu`
PIR1	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`(3)
	2439	4439	2539	4539	`-000 0000`	`-000 0000`	`-uuu uuuu`(3)
PIE1	2439	4439	2539	4539	`0000 0000`	`0000 0000`	`uuuu uuuu`
	2439	4439	2539	4539	`-000 0000`	`-000 0000`	`-uuu uuuu`
TRISE	2439	4439	2539	4539	`0000 -111`	`0000 -111`	`uuuu -uuu`
TRISD	2439	4439	2539	4539	`1111 1111`	`1111 1111`	`uuuu uuuu`
TRISC*	2439	4439	2539	4539	`1111 1111`	`1111 1111`	`uuuu uuuu`
TRISB	2439	4439	2539	4539	`1111 1111`	`1111 1111`	`uuuu uuuu`
TRISA(5,6)	2439	4439	2539	4539	`-111 1111`(5)	`-111 1111`(5)	`-uuu uuuu`(5)
LATE	2439	4439	2539	4539	`---- -xxx`	`---- -uuu`	`---- -uuu`
LATD	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
LATC*	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
LATB	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
LATA(5,6)	2439	4439	2539	4539	`-xxx xxxx`(5)	`-uuu uuuu`(5)	`-uuu uuuu`(5)
PORTE	2439	4439	2539	4539	`---- -000`	`---- -000`	`---- -uuu`
PORTD	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
PORTC*	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
PORTB	2439	4439	2539	4539	`xxxx xxxx`	`uuuu uuuu`	`uuuu uuuu`
PORTA(5,6)	2439	4439	2539	4539	`-x0x 0000`(5)	`-u0u 0000`(5)	`-uuu uuuu`(5)

Legend: `u` = unchanged, `x` = unknown, `-` = unimplemented bit, read as '0', `q` = value depends on condition. Shaded cells indicate conditions do not apply for the designated device.

* These registers are retained to maintain compatibility with PIC18FXX2 devices; however, one or more bits are reserved. Users should not modify the value of these bits. See Section 4.9.2 for details.

Note 1: 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.

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

4: See Table 3-2 for RESET value for specific condition.

5: Bit 6 of PORTA, LATA, and TRISA are enabled in ECIO and RCIO Oscillator modes only. In all other Oscillator modes, they are disabled and read ‘0’.

6: Bit 6 of PORTA, LATA and TRISA are not available on all devices. When unimplemented, they are read ‘0’.

FIGURE 3-3: TIME-OUT SEQUENCE ON POWER-UP ($\overline{\text{MCLR}}$ TIED TO VDD)

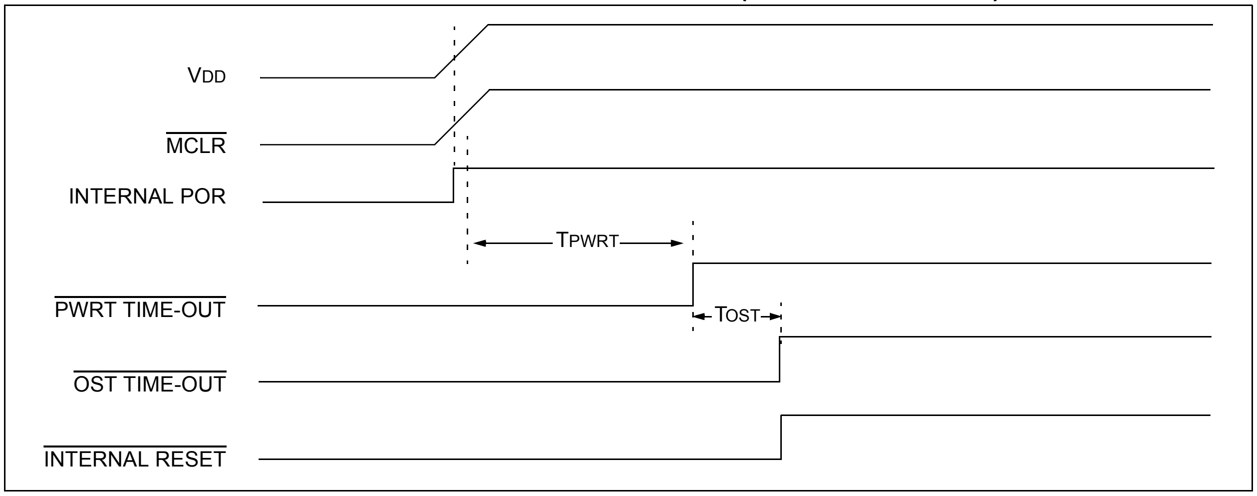


FIGURE 3-4: TIME-OUT SEQUENCE ON POWER-UP ($\overline{\text{MCLR}}$ NOT TIED TO VDD): CASE 1

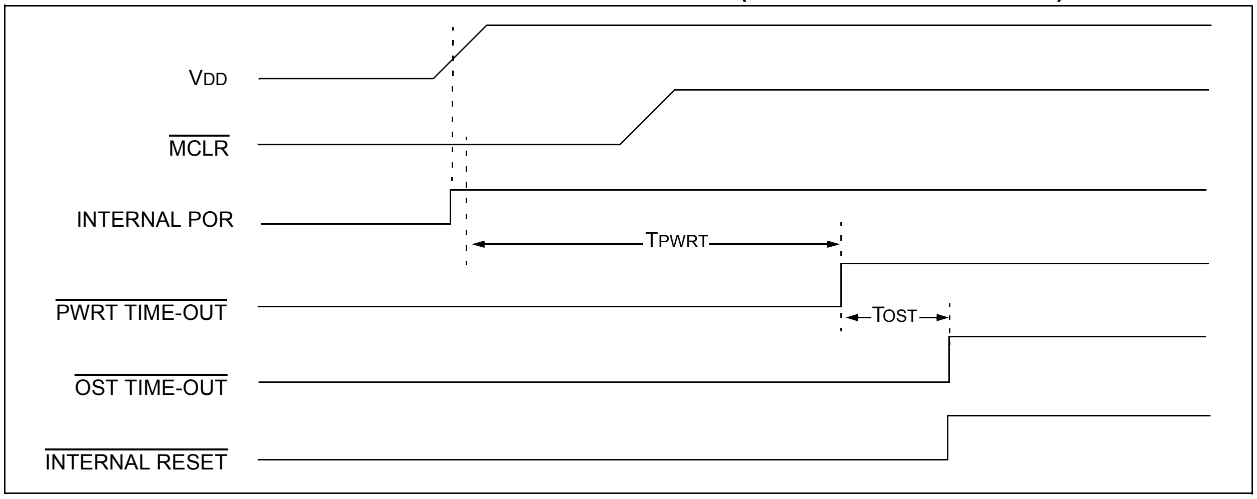


FIGURE 3-5: TIME-OUT SEQUENCE ON POWER-UP ($\overline{\text{MCLR}}$ NOT TIED TO VDD): CASE 2

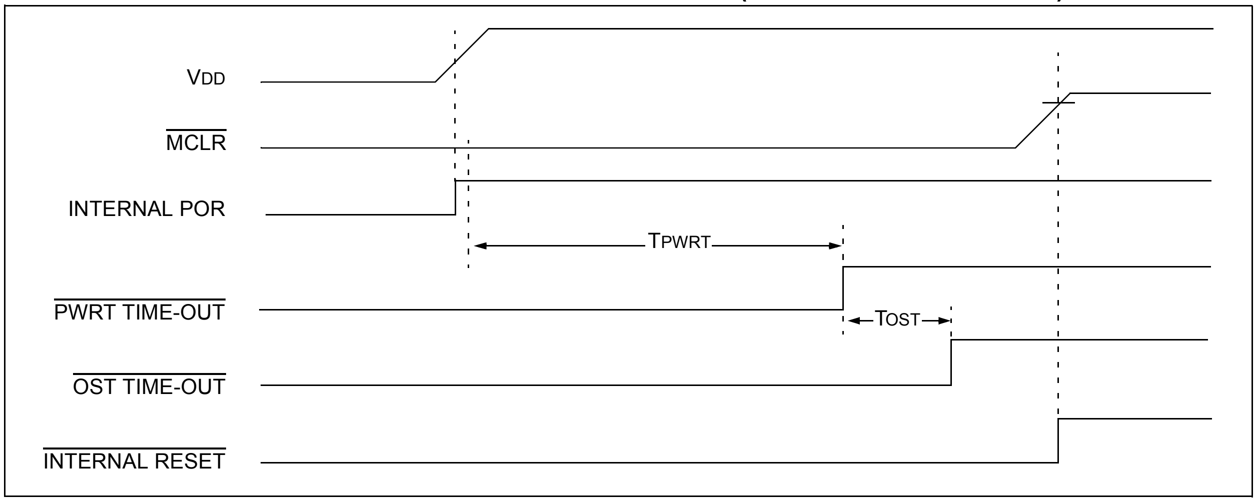


FIGURE 3-6: SLOW RISE TIME ($\overline{\text{MCLR}}$ TIED TO VDD)

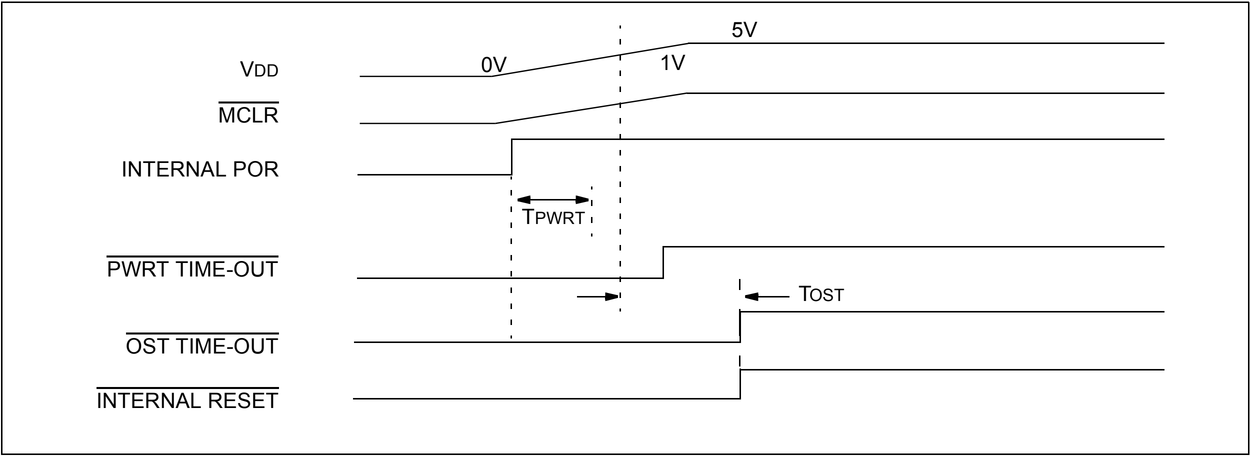
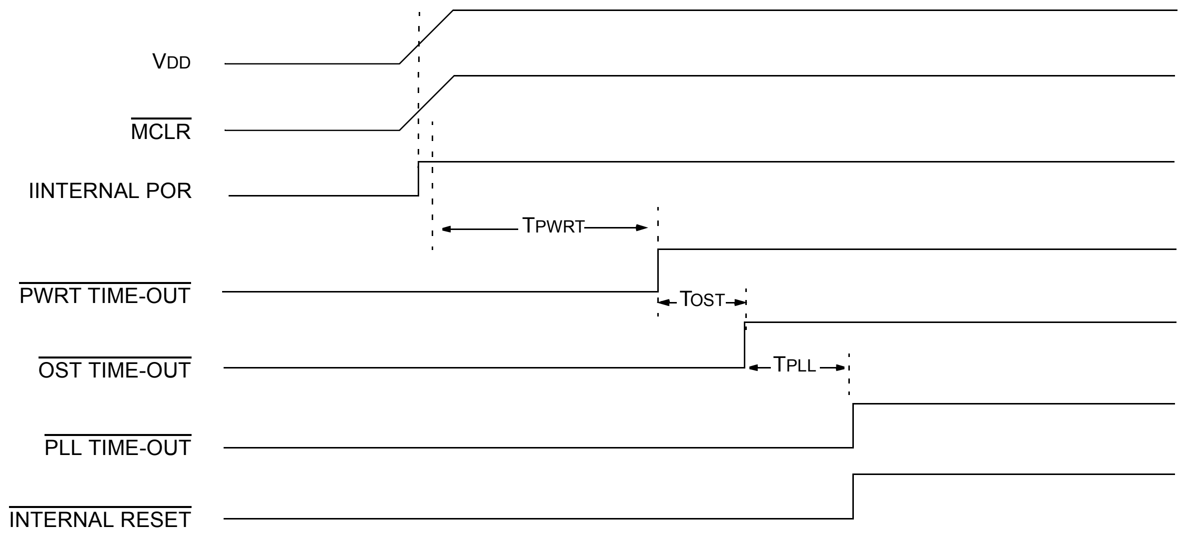

VDD
$\overline{\text{MCLR}}$
INTERNAL POR
$\overline{\text{PWRT TIME-OUT}}$
$\overline{\text{OST TIME-OUT}}$
$\overline{\text{INTERNAL RESET}}$
0V
1V
5V
TPWRT
TOST

FIGURE 3-7: TIME-OUT SEQUENCE ON POR W/ PLL ENABLED ($\overline{\text{MCLR}}$ TIED TO VDD)


VDD
$\overline{\text{MCLR}}$
IINTERNAL POR
$\overline{\text{PWRT TIME-OUT}}$
$\overline{\text{OST TIME-OUT}}$
$\overline{\text{PLL TIME-OUT}}$
$\overline{\text{INTERNAL RESET}}$
TPWRT
TOST
TPLL

Note: TOST = 1024 clock cycles.
TPLL ≈ 2 ms max. First three stages of the PWRT timer.

NOTES:

4.0 MEMORY ORGANIZATION

There are three memory blocks in Enhanced MCU devices. These memory blocks are:

- Program Memory
- Data RAM
- Data EEPROM

Data and program memory use separate busses, which allows for concurrent access of these blocks.

Additional detailed information for FLASH program memory and Data EEPROM is provided in Section 5.0 and Section 6.0, respectively.

4.1 Program Memory Organization

A 21-bit program counter is capable of addressing the 2-Mbyte program memory space. Accessing a location between the physically implemented memory and the top of the 2-MByte range will cause a read of all ‘0’s (a `NOP` instruction).

The PIC18F2539 and PIC18F4539 each have a total of 24 Kbytes, or 12K of single word instructions of FLASH memory, from addresses 0000h to 5FFFh. The next 8 Kbytes beyond this space (from 6000h to 7FFFh) are reserved for the Motor Control kernel; accessing locations in this range will return random information.

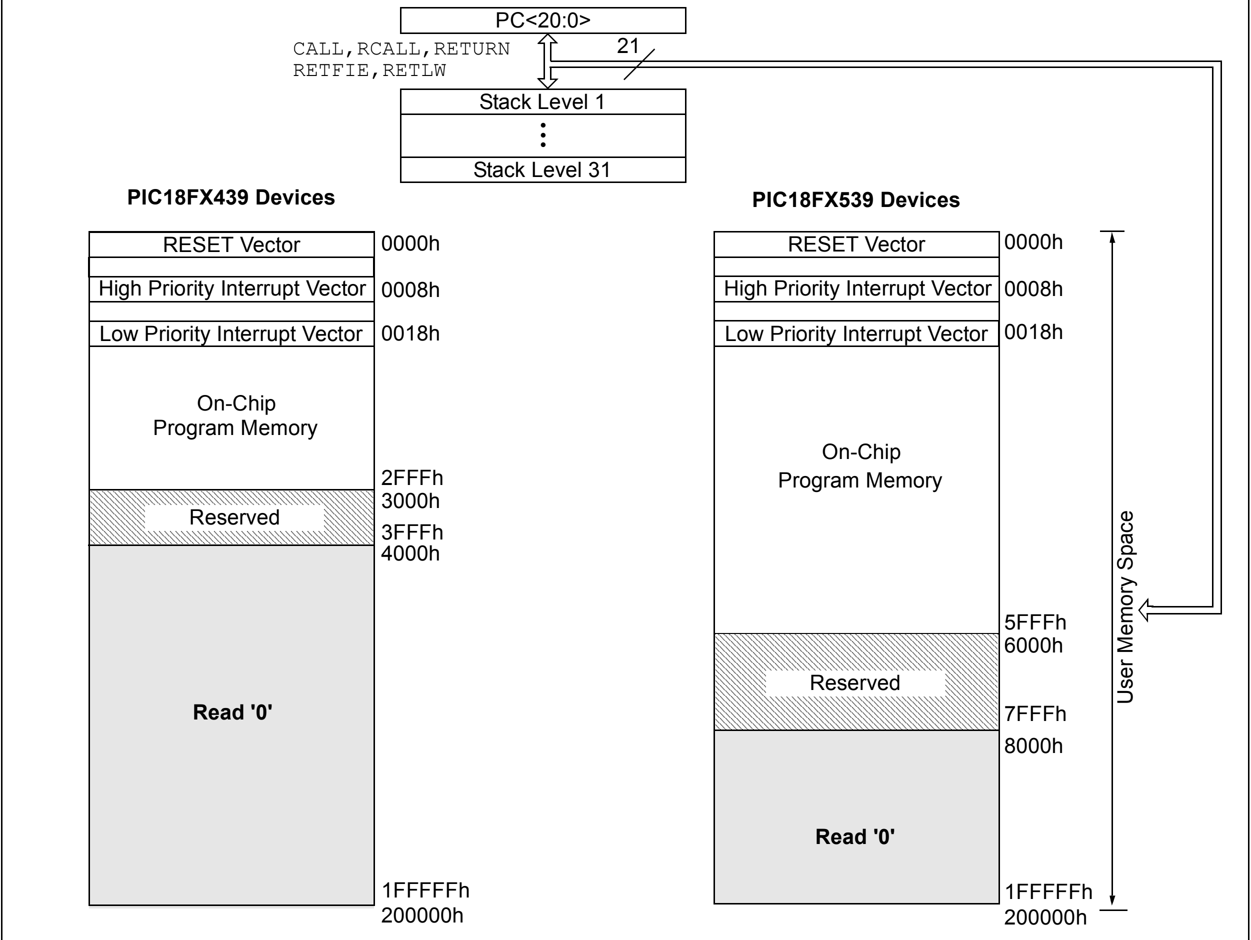
The PIC18F2439 and PIC18F4439 each have 12 Kbytes, or 6K of single word instructions of FLASH memory, from addresses 0000h to 2FFFh. The next 4 Kbytes of this space (from 3000h to 3FFFh) are reserved for the Motor Control kernel; accessing locations in this range will return random information.

The RESET vector address for all devices is at 0000h, and the interrupt vector addresses are at 0008h and 0018h.

The memory maps for the PIC18FX439 and PIC18FX539 devices are shown in Figure 4-1.

Note: The ProMPT Motor Control kernel is identical for all PIC18FXX39 devices, regardless of the difference in reserved block size between PIC18FX439 and PIC18FX539 devices

FIGURE 4-1: PROGRAM MEMORY MAP AND STACK FOR PIC18FXX39 DEVICES



Note: Size of memory areas not to scale.

4.2 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` or `CALL` instructions.

The stack operates as a 31-word by 21-bit RAM and a 5-bit stack pointer, with the stack pointer initialized to 00000b after all RESETS. There is no RAM associated with stack pointer 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 are transferred to the PC and then the stack pointer is decremented.

The stack space is not part of either program or data space. The stack pointer is readable and writable, and the address on the top of the stack is readable and writable through SFR registers. Data can also be pushed to, or popped from the stack using the top-of-stack SFRs. Status bits indicate if the stack pointer is at, or beyond the 31 levels provided.

4.2.1 TOP-OF-STACK ACCESS

The top of the stack 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.

4.2.2 RETURN STACK POINTER (STKPTR)

The STKPTR register contains the stack pointer value, the STKFUL (stack full) status bit, and the STKUNF (stack underflow) status bits. Register 4-1 shows the STKPTR register. The value of the stack pointer can be 0 through 31. The stack pointer increments when values are pushed onto the stack and decrements when values are popped off the stack. At RESET, the stack pointer value will be ‘0’. The user may read and write the stack pointer value. This feature can be used by a Real-Time Operating System for return stack maintenance.

After the PC is pushed onto the stack 31 times (without popping any values off the stack), the STKFUL bit is set. The STKFUL bit can only be cleared in software or by a POR.

The action that takes place when the stack becomes full depends on the state of the STVREN (Stack Overflow Reset Enable) configuration bit. Refer to Section 21.0 for a description of the device configuration bits. If STVREN is set (default), the 31st push will push the (PC + 2) value onto the stack, set the STKFUL bit, and reset the device. The STKFUL bit will remain set and the stack pointer will be set to ‘0’.

If STVREN is cleared, the STKFUL bit will be set on the 31st push and the stack pointer will increment to 31. Any additional pushes will not overwrite the 31st push, and STKPTR will remain at 31.

When the stack has been popped enough times to unload the stack, the next pop will return a value of zero to the PC and sets the STKUNF bit, while the stack pointer remains at ‘0’. The STKUNF bit will remain set until cleared in software or a POR occurs.

Note:	Returning a value of 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.

REGISTER 4-1: STKPTR REGISTER

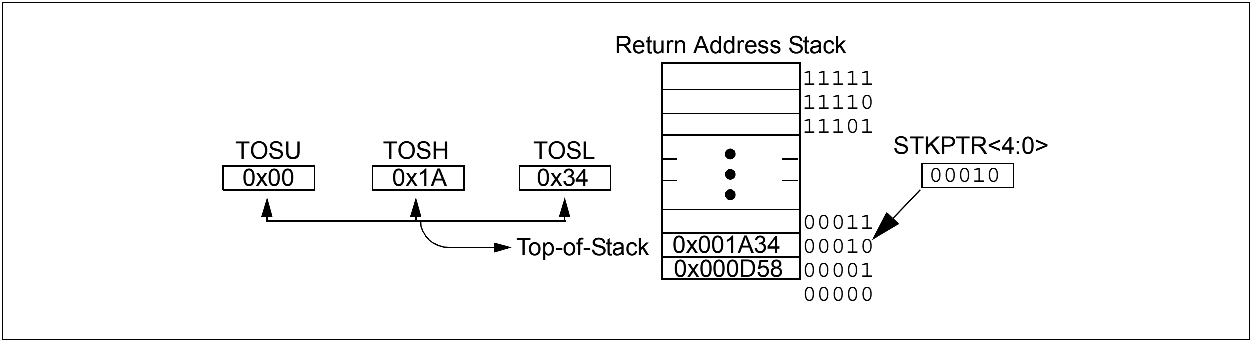
R/C-0	R/C-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STKFUL	STKUNF	—	SP4	SP3	SP2	SP1	SP0
bit 7							bit 0

bit 7[(1)]	**STKFUL:** Stack Full Flag bit `1` = Stack became full or overflowed `0` = Stack has not become full or overflowed
bit 6[(1)]	**STKUNF:** Stack Underflow Flag bit `1` = Stack underflow occurred `0` = Stack underflow did not occur
bit 5	**Unimplemented:** Read as '0'
bit 4-0	**SP4:SP0:** Stack Pointer Location bits

Note 1: Bit 7 and bit 6 can only be cleared in user software or by a POR.

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

FIGURE 4-2: RETURN ADDRESS STACK AND ASSOCIATED REGISTERS



4.2.3 `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 a return address on the stack.

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.

4.2.4 STACK FULL/UNDERFLOW RESETS

These RESETS are enabled by programming the STVREN configuration bit. When the STVREN bit is disabled, a full or underflow condition will set the appropriate STKFUL or STKUNF bit, but not cause a device RESET. When the STVREN bit is enabled, a full or underflow condition will set the appropriate STKFUL or STKUNF bit and then cause a device RESET. The STKFUL or STKUNF bits are only cleared by the user software or a POR Reset.

4.3 Fast Register Stack

For PIC18FXX39 devices, a "fast interrupt return" option is available for high priority interrupts. A single level Fast Register Stack is provided for the STATUS, WREG and BSR registers; it is not readable or writable. When the processor vectors for an interrupt, the stack is loaded with the current value of the corresponding register. If the `FAST RETURN` instruction is used to return from the interrupt, the values in the registers are then loaded back into the working registers.

Note:	The fast interrupt return for PIC18FXX39 devices is reserved for use by the ProMPT kernel and the Timer2 match interrupt. It is not available to the user for any other interrupts or returns from subroutines.

4.4 PCL, PCLATH and PCLATU

The program counter (PC) specifies the address of the instruction to fetch for execution. The PC is 21-bits wide. The low byte is called the PCL register. This register is readable and writable. The high byte is called the PCH register. This register contains the PC<15:8> bits and is not directly readable or writable. Updates to the PCH register may be performed through the PCLATH register. The upper byte is called PCU. This register contains the PC<20:16> bits and is not directly readable or writable. Updates to the PCU register may be performed through the PCLATU register.

The PC addresses bytes in the program memory. To prevent the PC from becoming misaligned with word instructions, the LSB of PCL is fixed to a value of '0'. The PC increments by 2 to address sequential instructions in the program memory.

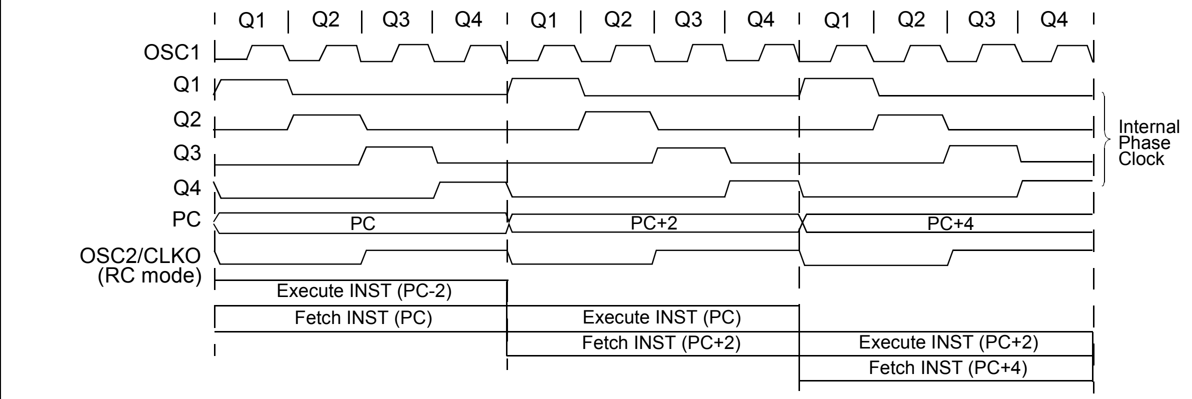
The `CALL`, `RCALL`, `GOTO` and program branch instructions write to the program counter directly. For these instructions, the contents of PCLATH and PCLATU are not transferred to the program counter.

The contents of PCLATH and PCLATU will be transferred to the program counter by an operation that writes PCL. Similarly, the upper two bytes of the program counter will be transferred to PCLATH and PCLATU by an operation that reads PCL. This is useful for computed offsets to the PC (see Section 4.8.1).

4.5 Clocking Scheme/Instruction Cycle

The clock input (from OSC1) is internally divided by four to generate four non-overlapping quadrature clocks, namely Q1, Q2, Q3 and Q4. Internally, the program counter (PC) is incremented every Q1, 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 shown in Figure 4-3.

FIGURE 4-3: CLOCK/INSTRUCTION CYCLE



4.6 Instruction Flow/Pipelining

An “Instruction Cycle” consists of four Q cycles (Q1, Q2, Q3 and Q4). The instruction fetch and execute are pipelined such that fetch takes one instruction cycle, while decode and execute takes another instruction cycle. However, due to the pipelining, each instruction effectively executes in one cycle. If an instruction causes the program counter to change (e.g., `GOTO`), then two cycles are required to complete the instruction (Example 4-1).

A fetch cycle begins with the program counter (PC) 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: INSTRUCTION PIPELINE FLOW

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
`1. MOVLW 55h`	Fetch 1	Execute 1				
`2. MOVWF PORTB`		Fetch 2	Execute 2			
`3. BRA  SUB_1`			Fetch 3	Execute 3		
`4. BSF   PORTA, BIT3 (Forced NOP)`				Fetch 4	Flush (`NOP`)	
`5. Instruction @ address SUB_1`					Fetch SUB_1	Execute SUB_1

All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is “flushed” from the pipeline while the new instruction is being fetched and then executed.

4.7 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 4-4 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’ (see Section 4.4).

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 4-4 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. Section 21.0 provides further details of the instruction set.

FIGURE 4-4: INSTRUCTIONS IN PROGRAM MEMORY

Program Memory Byte Locations →

			LSB = 1	LSB = 0	Word Address ↓
					`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`

4.7.1 TWO-WORD INSTRUCTIONS

The PIC18FXX39 devices have four two-word instructions: `MOVFF`, `CALL`, `GOTO` and `LFSR`. The second word of these instructions has the 4 MSBs set to '1's and is a special kind of `NOP` instruction. The lower 12 bits of the second word contain data to be used by the instruction. If the first word of the instruction is executed, the data in the second word is accessed. If the second word of the instruction is executed by itself (first word was skipped), it will execute as a `NOP`. This action is necessary when the two-word instruction is preceded by a conditional instruction that changes the PC. A program example that demonstrates this concept is shown in Example 4-2. Refer to Section 21.0 for further details of the instruction set.

EXAMPLE 4-2: TWO-WORD INSTRUCTIONS

CASE 1:			
Object Code	**Source Code**		
0110 0110 0000 0000	TSTFSZ	REG1	; is RAM location 0?
1100 0001 0010 0011	MOVFF	REG1, REG2	; No, execute 2-word instruction
1111 0100 0101 0110			; 2nd operand holds address of REG2
0010 0100 0000 0000	ADDWF	REG3	; continue code
CASE 2:			
Object Code	**Source Code**		
0110 0110 0000 0000	TSTFSZ	REG1	; is RAM location 0?
1100 0001 0010 0011	MOVFF	REG1, REG2	; Yes
1111 0100 0101 0110			; 2nd operand becomes NOP
0010 0100 0000 0000	ADDWF	REG3	; continue code

4.8 Lookup Tables

Lookup tables are implemented two ways. These are:

- Computed `GOTO`
- Table Reads

4.8.1 COMPUTED GOTO

A computed `GOTO` is accomplished by adding an offset to the program counter (`ADDWF PCL`).

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.

The offset value (value in WREG) specifies the number of bytes that the program counter should advance.

In this method, only one data byte may be stored in each instruction location and room on the return address stack is required.

Note: The `ADDWF PCL` instruction does not update PCLATH and PCLATU. A read operation on PCL must be performed to update PCLATH and PCLATU.

4.8.2 TABLE READS/TABLE WRITES

A better method of storing data in program memory allows 2 bytes of data to be stored in each instruction location.

Lookup table data may be stored 2 bytes per program word by using table reads and writes. The table pointer (TBLPTR) specifies the byte address and the table latch (TABLAT) contains the data that is read from, or written to program memory. Data is transferred to/from program memory, one byte at a time.

A description of the Table Read/Table Write operation is shown in Section 5.1.

4.9 Data Memory Organization

The data memory is implemented as static RAM. Each register in the data memory has a 12-bit address, allowing up to 4096 bytes of data memory. The data memory map is divided into 16 banks that contain 256 bytes each. The lower 4 bits of the Bank Select Register (BSR<3:0>) select which bank will be accessed. The upper 4 bits for the BSR are not implemented.

The data memory contains Special Function Registers (SFRs) and General Purpose Registers (GPRs). The SFRs are used for control and status of the controller and peripheral functions, while GPRs are used for data storage and scratch pad operations in the user's application. The SFRs start at the last location of Bank 15 (FFFh) and extend downwards. Any remaining space beyond the SFRs in the Bank may be implemented as GPRs. GPRs start at the first location of Bank 0 and grow upwards. Any read of an unimplemented location will read as '0's.

The organization of the data memory space for these devices is shown in Figure 4-5 and Figure 4-6. PIC18FX439 devices have 640 bytes of data RAM, extending from Bank 0 to Bank 2 (000h through 27Fh). The block of 128 bytes above this to the top of the bank (280h to 2FFh) is used as data memory for the Motor Control kernel, and is not available to the user. Reading these locations will return random information that reflects the kernel's "scratch" data. Modifying the data in these locations may disrupt the operation of the ProMPT kernel.

PIC18FX539 devices have 1408 bytes of data RAM, extending from Bank 0 to Bank 5 (000h through 57Fh). As with the PIC18FX439 devices, the block of 128 bytes above this to the end of the bank (580h to 5FFh) is used by the Motor Control kernel.

The entire data memory may be accessed directly or indirectly. Direct addressing may require the use of the BSR register. Indirect addressing requires the use of a File Select Register (FSRn) and a corresponding Indirect File Operand (INDFn). Each FSR holds a 12-bit address value that can be used to access any location in the Data Memory map without banking.

The instruction set and architecture allow operations across all banks. This may be accomplished by indirect addressing, or by the use of the `MOVFF` instruction. The `MOVFF` instruction is a two-word/two-cycle instruction that moves a value from one register to another.

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. A segment of Bank 0 and a segment of Bank 15 comprise the Access RAM. Section 4.10 provides a detailed description of the Access RAM.

4.9.1 GENERAL PURPOSE REGISTER FILE

The register file can be accessed either directly or indirectly. Indirect addressing operates using a File Select Register and corresponding Indirect File Operand. The operation of indirect addressing is shown in Section 4.12.

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.

Data RAM is available for use as GPR registers by all instructions. The top half of Bank 15 (F80h to FFFh) contains SFRs. All other banks of data memory contain GPR registers, starting with Bank 0.

4.9.2 SPECIAL FUNCTION REGISTERS

The Special Function Registers (SFRs) are registers used by the CPU and Peripheral Modules for controlling the desired operation of the device. These registers are implemented as static RAM. A list of these registers is given in Table 4-1 and Table 4-2.

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. The unused SFR locations will be unimplemented and read as '0's. See Table 4-1 for addresses for the SFRs.

Note:	In this chapter and throughout this document, certain SFR names and individual bits are marked with an asterisk (*). This denotes registers that are not implemented in PIC18FXX39 devices, but whose names are retained to maintain compatibility with PIC18FXX2 devices. The designated bits within these registers are reserved and may be used by certain modules or the Motor Control kernel. Users should not write to these registers or alter these bit values. Failure to do this may result in erratic microcontroller operation.

FIGURE 4-5: DATA MEMORY MAP FOR PIC18FX439

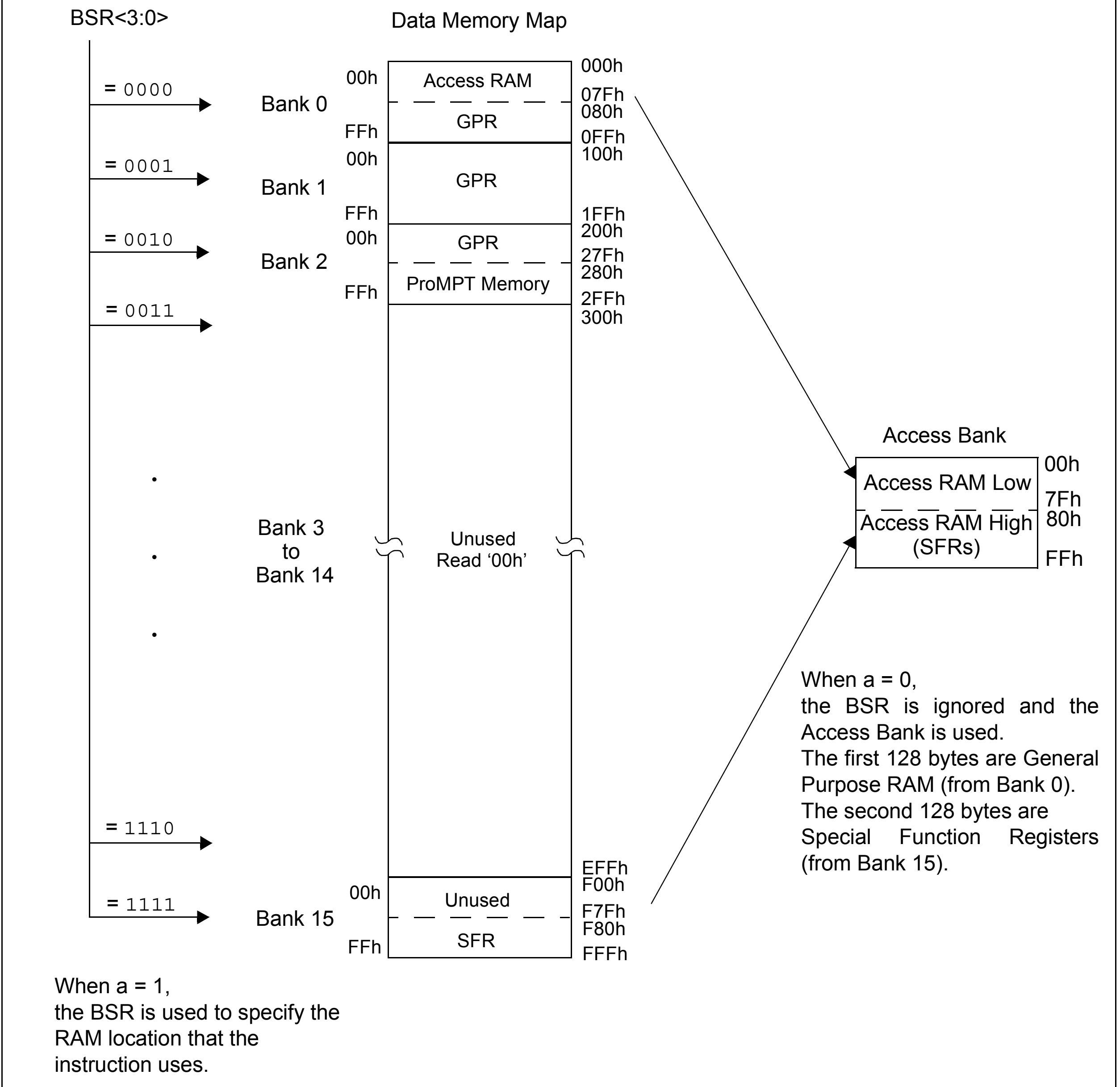
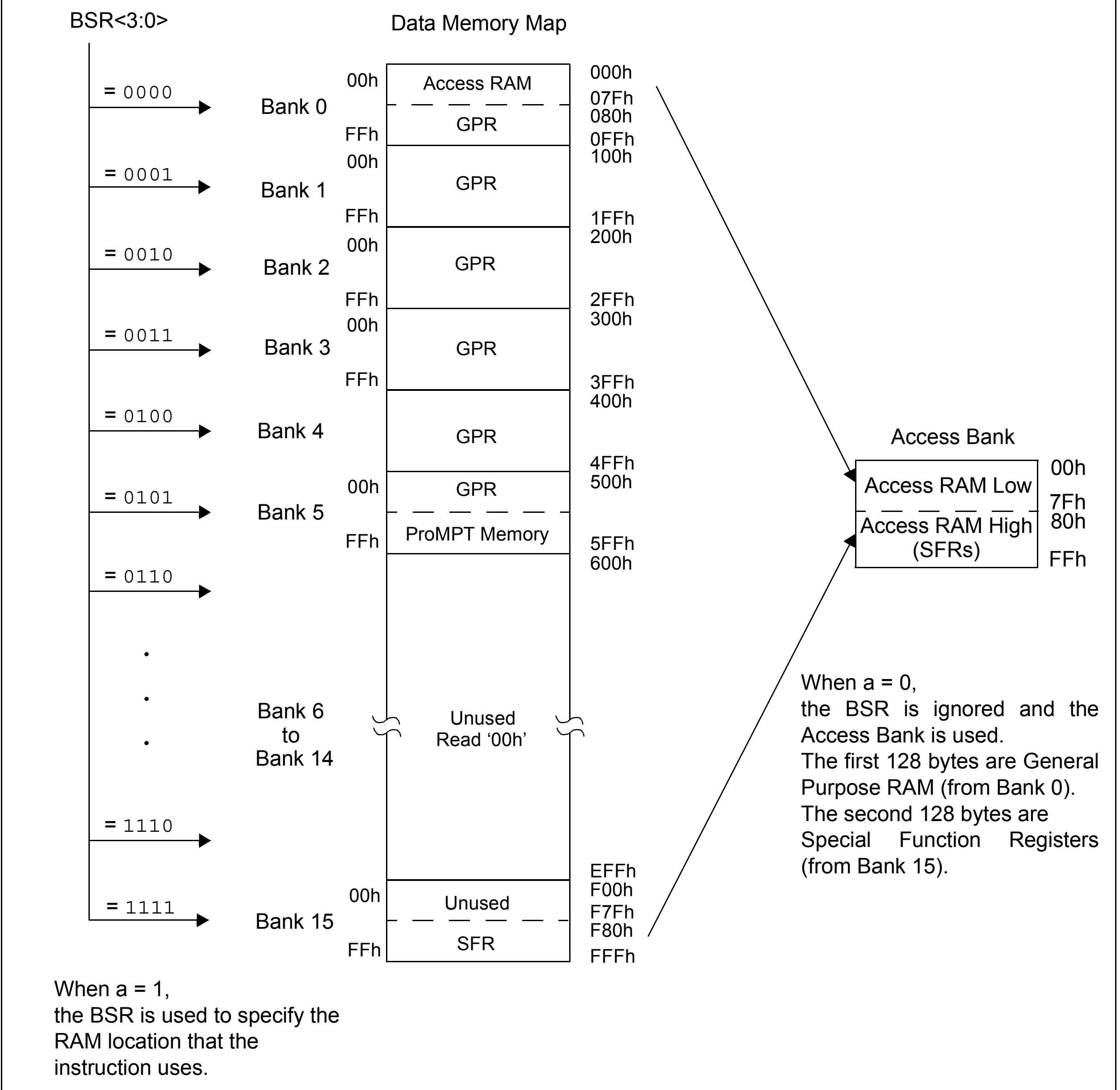


FIGURE 4-6: DATA MEMORY MAP FOR PIC18FX539



When a = 0,
the BSR is ignored and the Access Bank is used.
The first 128 bytes are General Purpose RAM (from Bank 0).
The second 128 bytes are Special Function Registers (from Bank 15).

When a = 1,
the BSR is used to specify the RAM location that the instruction uses.

TABLE 4-1: SPECIAL FUNCTION REGISTER MAP

Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2(3)	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2(3)	FBEh	CCPR1L*	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2(3)	FBDh	CCP1CON*	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2(3)	FBCh	CCPR2H	F9Ch	—
FFBh	PCLATU	FDBh	PLUSW2(3)	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(2)
FF5h	TABLAT	FD5h	T0CON	FB5h	—	F95h	TRISD(2)
FF4h	PRODH	FD4h	—	FB4h	—	F94h	TRISC(4)
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(3)	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—
FEEh	POSTINC0(3)	FCEh	TMR1L	FAEh	RCREG	F8Eh	—
FEDh	POSTDEC0(3)	FCDh	T1CON	FADh	TXREG	F8Dh	LATE(2)
FECh	PREINC0(3)	FCCh	TMR2*	FACh	TXSTA	F8Ch	LATD(2)
FEBh	PLUSW0(3)	FCBh	PR2*	FABh	RCSTA	F8Bh	LATC(4)
FEAh	FSR0H	FCAh	T2CON*	FAAh	—	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	—
FE7h	INDF1(3)	FC7h	SSPSTAT	FA7h	EECON2	F87h	—
FE6h	POSTINC1(3)	FC6h	SSPCON1	FA6h	EECON1	F86h	—
FE5h	POSTDEC1(3)	FC5h	SSPCON2	FA5h	—	F85h	—
FE4h	PREINC1(3)	FC4h	ADRESH	FA4h	—	F84h	PORTE(2)
FE3h	PLUSW1(3)	FC3h	ADRESL	FA3h	—	F83h	PORTD(2)
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC(4)
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	—	FA0h	PIE2	F80h	PORTA

* These registers are retained to maintain compatibility with PIC18FXX2 devices; however, one or more bits are reserved in PIC18FXX39 devices. Users should not alter the values of these bits.

Note 1: Unimplemented registers are read as ‘0’.
2: This register is not available on PIC18F2X39 devices.
3: This is not a physical register.
4: Bits 1 and 2 are reserved; users should not alter their values.

TABLE 4-2: REGISTER FILE SUMMARY

File Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Details on page:
TOSU	—	—	—	Top-of-Stack Upper Byte (TOS<20:16>)					`---0 0000`	26, 34
TOSH	Top-of-Stack High Byte (TOS<15:8>)								`0000 0000`	26, 34
TOSL	Top-of-Stack Low Byte (TOS<7:0>)								`0000 0000`	26, 34
STKPTR	STKFUL	STKUNF	—	Return Stack Pointer					`00-0 0000`	26, 35
PCLATU	—	—	—	Holding Register for PC<20:16>					`---0 0000`	26, 36
PCLATH	Holding Register for PC<15:8>								`0000 0000`	26, 36
PCL	PC Low Byte (PC<7:0>)								`0000 0000`	26, 36
TBLPTRU	—	—	bit21[(2)]	Program Memory Table Pointer Upper Byte (TBLPTR<20:16>)					`--00 0000`	26, 54
TBLPTRH	Program Memory Table Pointer High Byte (TBLPTR<15:8>)								`0000 0000`	26, 54
TBLPTRL	Program Memory Table Pointer Low Byte (TBLPTR<7:0>)								`0000 0000`	26, 54
TABLAT	Program Memory Table Latch								`0000 0000`	26, 54
PRODH	Product Register High Byte								`xxxx xxxx`	26, 67
PRODL	Product Register Low Byte								`xxxx xxxx`	26, 67
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	`0000 000x`	26, 71
INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	`1111 -1-1`	26, 72
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF	`11-0 0-00`	26, 73
INDF0	Uses contents of FSR0 to address data memory - value of FSR0 not changed (not a physical register)								n/a	26, 47
POSTINC0	Uses contents of FSR0 to address data memory - value of FSR0 post-incremented (not a physical register)								n/a	26, 47
POSTDEC0	Uses contents of FSR0 to address data memory - value of FSR0 post-decremented (not a physical register)								n/a	26, 47
PREINC0	Uses contents of FSR0 to address data memory - value of FSR0 pre-incremented (not a physical register)								n/a	26, 47
PLUSW0	Uses contents of FSR0 to address data memory - value of FSR0 (not a physical register). Offset by value in WREG.								n/a	26, 47
FSR0H	—	—	—	—	Indirect Data Memory Address Pointer 0 High Byte				`---- 0000`	26, 47
FSR0L	Indirect Data Memory Address Pointer 0 Low Byte								`xxxx xxxx`	26, 47
WREG	Working Register								`xxxx xxxx`	26
INDF1	Uses contents of FSR1 to address data memory - value of FSR1 not changed (not a physical register)								n/a	26, 47
POSTINC1	Uses contents of FSR1 to address data memory - value of FSR1 post-incremented (not a physical register)								n/a	26, 47
POSTDEC1	Uses contents of FSR1 to address data memory - value of FSR1 post-decremented (not a physical register)								n/a	26, 47
PREINC1	Uses contents of FSR1 to address data memory - value of FSR1 pre-incremented (not a physical register)								n/a	26, 47
PLUSW1	Uses contents of FSR1 to address data memory - value of FSR1 (not a physical register). Offset by value in WREG.								n/a	26, 47
FSR1H	—	—	—	—	Indirect Data Memory Address Pointer 1 High Byte				`---- 0000`	27, 47
FSR1L	Indirect Data Memory Address Pointer 1 Low Byte								`xxxx xxxx`	27, 47
BSR	—	—	—	—	Bank Select Register				`---- 0000`	27, 46
INDF2	Uses contents of FSR2 to address data memory - value of FSR2 not changed (not a physical register)								n/a	27, 47
POSTINC2	Uses contents of FSR2 to address data memory - value of FSR2 post-incremented (not a physical register)								n/a	27, 47
POSTDEC2	Uses contents of FSR2 to address data memory - value of FSR2 post-decremented (not a physical register)								n/a	27, 47
PREINC2	Uses contents of FSR2 to address data memory - value of FSR2 pre-incremented (not a physical register)								n/a	27, 47
PLUSW2	Uses contents of FSR2 to address data memory - value of FSR2 (not a physical register). Offset by value in WREG.								n/a	27, 47
FSR2H	—	—	—	—	Indirect Data Memory Address Pointer 2 High Byte				`---- 0000`	27, 47
FSR2L	Indirect Data Memory Address Pointer 2 Low Byte								`xxxx xxxx`	27, 47
STATUS	—	—	—	N	OV	Z	DC	C	`---x xxxx`	27, 49

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, `q` = value depends on condition
* These registers (or individual bits) are retained to maintain compatibility with PIC18FXX2 devices; however, the indicated bits are reserved in PIC18FXX39 devices. Users should not alter the values of these bits. See Section 4.9.2 for details.

Note 1: RA6 and associated bits are configured as port pins in RCIO and ECIO Oscillator mode only and read '0' in all other Oscillator modes.
2: Bit 21 of the TBLPTRU allows access to the device configuration bits.
3: These registers and bits are reserved on the PIC18F2X39 devices; always maintain these clear.

TABLE 4-2: REGISTER FILE SUMMARY (CONTINUED)

File Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Details on page:
TMR0H	Timer0 Register High Byte								`0000 0000`	27, 101
TMR0L	Timer0 Register Low Byte								`xxxx xxxx`	27, 101
T0CON	TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	`1111 1111`	27, 99
OSCCON*	—	—	—	—	—	—	—	*	`---- ---0`	27
LVDCON	—	—	IRVST	LVDEN	LVDL3	LVDL2	LVDL1	LVDL0	`--00 0101`	27, 191
WDTCON	—	—	—	—	—	—	—	SWDTE	`---- ---0`	27, 203
RCON	IPEN	—	—	$\overline{RI}$	$\overline{TO}$	$\overline{PD}$	$\overline{POR}$	$\overline{BOR}$	`0--1 11qq`	25, 50, 80
TMR1H	Timer1 Register High Byte								`xxxx xxxx`	27, 103
TMR1L	Timer1 Register Low Byte								`xxxx xxxx`	27, 103
T1CON	RD16	—	T1CKPS1	T1CKPS0	—	$\overline{T1SYNC}$	TMR1CS	TMR1ON	`0-00 0000`	27, 103
TMR2*	*	*	*	*	*	*	*	*	`0000 0000`	27
PR2*	*	*	*	*	*	*	*	*	`1111 1111`	27
T2CON*	*	*	*	*	*	*	*	*	`-000 0000`	27
SSPBUF	SSP Receive Buffer/Transmit Register								`xxxx xxxx`	27, 125
SSPADD	SSP Address Register in I^{2}C Slave mode. SSP Baud Rate Reload Register in I^{2}C Master mode.								`0000 0000`	27, 134
SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	`0000 0000`	27, 126
SSPCON1	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	`0000 0000`	27, 127
SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	`0000 0000`	27, 137
ADRESH	A/D Result Register High Byte								`xxxx xxxx`	187,188
ADRESL	A/D Result Register Low Byte								`xxxx xxxx`	187,188
ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	`0000 00-0`	28, 181
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	`00-- 0000`	28, 182
CCPR1H	PWM Register1 High Byte (Read only)								`xxxx xxxx`	28, 124
CCPR1L*	*	*	*	*	*	*	*	*	`xxxx xxxx`	28, 124
CCP1CON*	—	—	*	*	*	*	*	*	`--00 0000`	28, 124
CCPR2H	PWM Register2 High Byte (Read only)								`xxxx xxxx`	28, 124
CCPR2L*	*	*	*	*	*	*	*	*	`xxxx xxxx`	28, 124
CCP2CON*	—	—	*	*	*	*	*	*	`--00 0000`	28, 124
TMR3H	Timer3 Register High Byte								`xxxx xxxx`	28, 109
TMR3L	Timer3 Register Low Byte								`xxxx xxxx`	28, 109
T3CON	RD16	—	T3CKPS1	T3CKPS0	—	$\overline{T3SYNC}$	TMR3CS	TMR3ON	`0000 0000`	28, 109
SPBRG	USART1 Baud Rate Generator								`0000 0000`	28, 168
RCREG	USART1 Receive Register								`0000 0000`	28, 175, 178
TXREG	USART1 Transmit Register								`0000 0000`	28, 173, 176
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	`0000 -010`	28, 166
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	`0000 000x`	28, 167
EEADR	Data EEPROM Address Register								`0000 0000`	28, 61, 65
EEDATA	Data EEPROM Data Register								`0000 0000`	28, 65
EECON2	Data EEPROM Control Register 2 (not a physical register)								`---- ----`	28, 61, 65
EECON1	EEPGD	CFGS	—	FREE	WRERR	WREN	WR	RD	`xx-0 x000`	28, 62

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, `q` = value depends on condition
* These registers (or individual bits) are retained to maintain compatibility with PIC18FXX2 devices; however, the indicated bits are reserved in PIC18FXX39 devices. Users should not alter the values of these bits. See Section 4.9.2 for details.
Note 1: RA6 and associated bits are configured as port pins in RCIO and ECIO Oscillator mode only and read '0' in all other Oscillator modes.
2: Bit 21 of the TBLPTRU allows access to the device configuration bits.
3: These registers and bits are reserved on the PIC18F2X39 devices; always maintain these clear.

TABLE 4-2: REGISTER FILE SUMMARY (CONTINUED)

File Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Details on page:
IPR2	—	—	—	EEIP	BCLIP	LVDIP	TMR3IP	—	`---1 1111`	29, 79
PIR2	—	—	—	EEIF	BCLIF	LVDIF	TMR3IF	—	`---0 0000`	29, 75
PIE2	—	—	—	EEIE	BCLIE	LVDIE	TMR3IE	—	`---0 0000`	29, 77
IPR1	PSPIP(3)	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	`1111 1111`	29, 78
PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF	TMR1IF	`0000 0000`	29, 74
PIE1	PSPIE(3)	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	`0000 0000`	29, 76
TRISE(3)	IBF	OBF	IBOV	PSPMODE	—	Data Direction bits for PORTE			`0000 -111`	29, 94
TRISD(3)	Data Direction Control Register for PORTD								`1111 1111`	29, 92
TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	*	*	TRISC0	`1111 1111`	29, 89
TRISB	Data Direction Control Register for PORTB								`1111 1111`	29, 86
TRISA	—	TRISA6(1)	Data Direction Control Register for PORTA						`-111 1111`	29, 83
LATE(3)	—	—	—	—	—	Read PORTE Data Latch, Write PORTE Data Latch			`---- -xxx`	29, 95
LATD(3)	Read PORTD Data Latch, Write PORTD Data Latch								`xxxx xxxx`	29, 91
LATC	LATC7	LATC6	LATC5	LATC4	LATC3	*	*	LATC0	`xxxx xxxx`	29, 89
LATB	Read PORTB Data Latch, Write PORTB Data Latch								`xxxx xxxx`	29, 86
LATA	—	LATA6(1)	Read PORTA Data Latch, Write PORTA Data Latch(1)						`-xxx xxxx`	29, 83
PORTE(3)	Read PORTE pins, Write PORTE Data Latch								`---- -000`	29, 95
PORTD(3)	Read PORTD pins, Write PORTD Data Latch								`xxxx xxxx`	29, 91
PORTC	RC7	RC6	RC5	RC4	RC3	*	*	RC0	`xxxx xxxx`	29, 89
PORTB	Read PORTB pins, Write PORTB Data Latch								`xxxx xxxx`	29, 86
PORTA	—	RA6(1)	Read PORTA pins, Write PORTA Data Latch(1)						`-x0x 0000`	29, 83

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, `q` = value depends on condition
* These registers (or individual bits) are retained to maintain compatibility with PIC18FXX2 devices; however, the indicated bits are reserved in PIC18FXX39 devices. Users should not alter the values of these bits. See Section 4.9.2 for details.

Note 1: RA6 and associated bits are configured as port pins in RCIO and ECIO Oscillator mode only and read '0' in all other Oscillator modes.
2: Bit 21 of the TBLPTRU allows access to the device configuration bits.
3: These registers and bits are reserved on the PIC18F2X39 devices; always maintain these clear.

4.10 Access Bank

The Access Bank is an architectural enhancement which 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 128 bytes in Bank 15 (SFRs) and the lower 128 bytes in Bank 0. These two sections will be referred to as Access RAM High and Access RAM Low, respectively. Figure 4-5 and Figure 4-6 indicate the Access RAM areas.

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 and modifying control bits.

4.11 Bank Select Register (BSR)

The need for a large general purpose memory space dictates a RAM banking scheme. The data memory is partitioned into sixteen banks. When using direct addressing, the BSR should be configured for the desired bank.

BSR<3:0> holds the upper 4 bits of the 12-bit RAM address. The BSR<7:4> bits will always read '0's, and writes will have no effect.

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

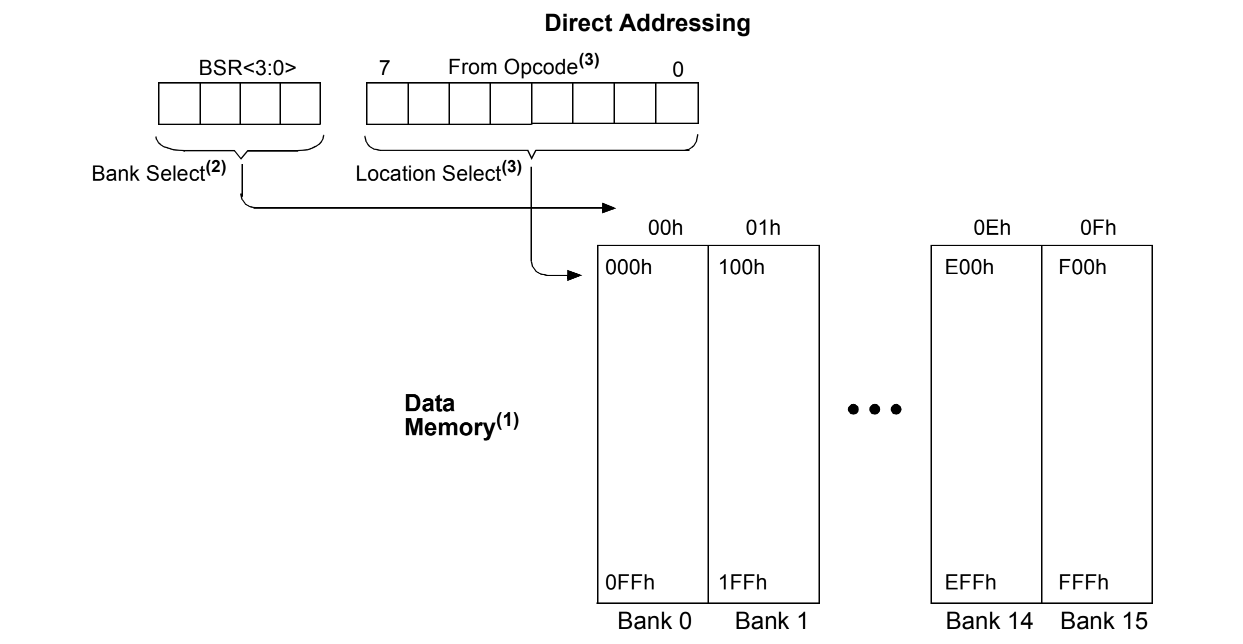
If the currently selected bank is not implemented, any read will return all '0's and all writes are ignored. The STATUS register bits will be set/cleared as appropriate for the instruction performed.

Each Bank extends up to FFh (256 bytes). All data memory is implemented as static RAM.

A `MOVFF` instruction ignores the BSR, since the 12-bit addresses are embedded into the instruction word.

Section 4.12 provides a description of indirect addressing, which allows linear addressing of the entire RAM space.

FIGURE 4-7: DIRECT ADDRESSING



Note 1: For register file map detail, see Table 4-1.

2: The access bit of the instruction can be used to force an override of the selected bank (BSR<3:0>) to the registers of the Access Bank.

3: The `MOVFF` instruction embeds the entire 12-bit address in the instruction.

4.12 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 FSR 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 4-8 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 one of the INDF registers. 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 `NOP` operation. The FSR register contains a 12-bit address, which is shown in Figure 4-9.

The INDFn register is not a physical register. Addressing INDFn actually addresses the register whose address is contained in the FSRn register (FSRn is a pointer). This is indirect addressing.

Example 4-3 shows a simple use of indirect addressing to clear the RAM in Bank 1 (locations 100h-1FFh) in a minimum number of instructions.

EXAMPLE 4-3: HOW TO CLEAR RAM (BANK 1) USING INDIRECT ADDRESSING

```
        LFSR  FSR0 ,0x100  ;
NEXT    CLRF  POSTINC0     ; Clear INDF
                           ; register and
                           ; inc pointer
        BTFSS FSR0H, 1     ; All done with
                           ; Bank1?
        GOTO  NEXT         ; NO, clear next
CONTINUE                   ; YES, continue
```

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 reads the data from the address pointed to by FSR1H:FSR1L. INDFn can be used in code anywhere an operand can be used.

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

4.12.1 INDIRECT ADDRESSING OPERATION

Each FSR register has an INDF register associated with it, plus four additional register addresses. Performing an operation on one of these five registers determines how the FSR will be modified during indirect addressing.

When data access is done to 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.

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 4-8: INDIRECT ADDRESSING OPERATION

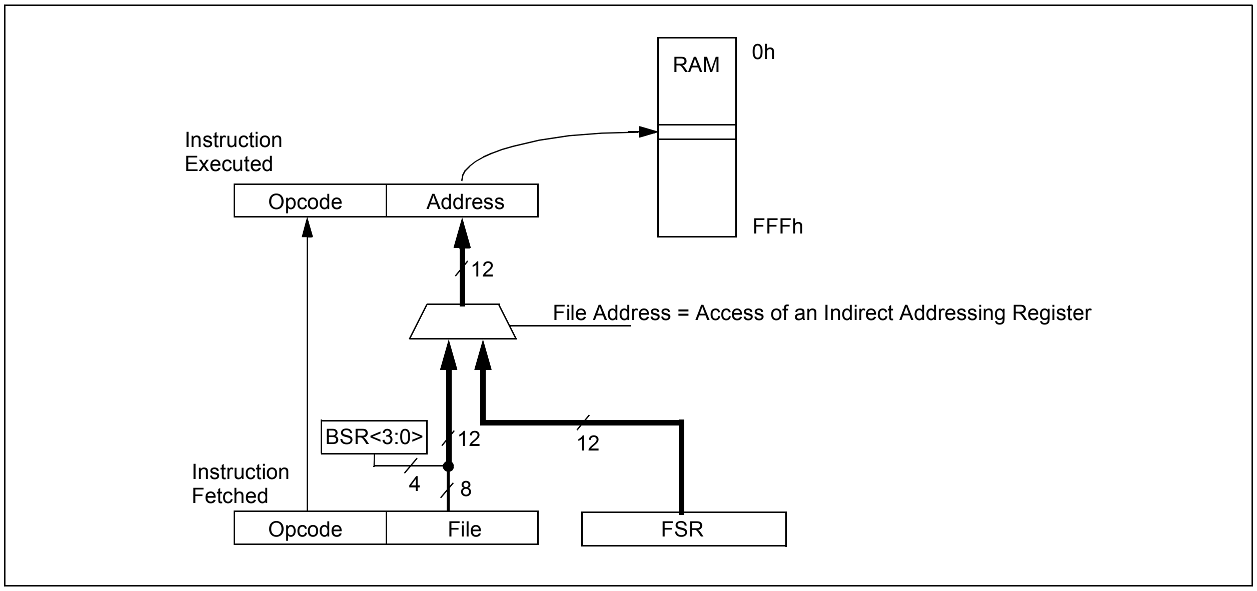
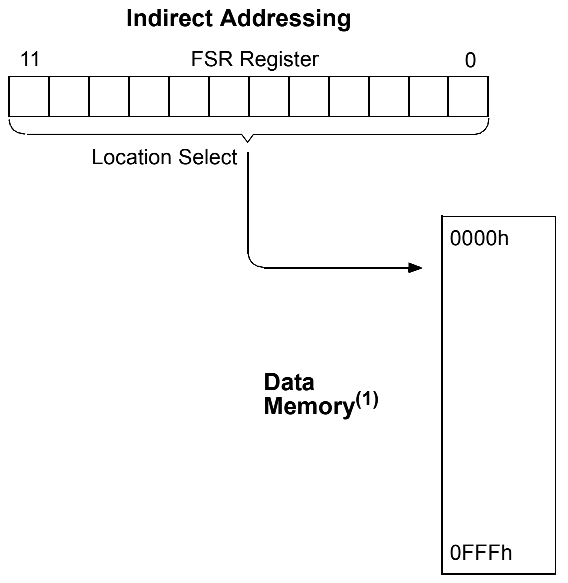


FIGURE 4-9: INDIRECT ADDRESSING



Note 1: For register file map detail, see Table 4-1.

4.13 STATUS Register

The STATUS register, shown in Register 4-2, 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 from the STATUS register. For other instructions not affecting any status bits, see Table 21-2.

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

REGISTER 4-2: 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 (bit 7) 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 two’s complement of the second operand. For rotate (`RRF, RLF`) instructions, this bit is loaded with either the bit 4 or bit 3 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 two’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	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

4.14 RCON Register

The Reset Control (RCON) register contains flag bits that allow differentiation between the sources of a device RESET. These flags include the $\overline{\text{TO}}$, $\overline{\text{PD}}$, $\overline{\text{POR}}$, $\overline{\text{BOR}}$ and $\overline{\text{RI}}$ bits. This register is readable and writable.

For PIC18FXX39 devices, the IPEN bit must always be set (= `1`) for the ProMPT kernel to function correctly. Refer to Section 8.0 (page 69) for a more detailed discussion.

Note 1: If the BOREN configuration bit is set (Brown-out Reset enabled), the $\overline{\text{BOR}}$ bit is '1' on a Power-on Reset. After a Brown-out Reset has occurred, the BOR bit will be cleared, and must be set by firmware to indicate the occurrence of the next Brown-out Reset.

2: It is recommended that the $\overline{\text{POR}}$ bit be set after a Power-on Reset has been detected, so that subsequent Power-on Resets may be detected.

REGISTER 4-3: RCON REGISTER

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

bit 7 **IPEN:** Interrupt Priority Enable bit
Always maintain this bit set for proper operation of ProMPT kernel.

bit 6-5 **Unimplemented:** Read as '0'

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

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

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

bit 1 **$\overline{\text{POR}}$:** Power-on Reset Status 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 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	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

5.0 FLASH PROGRAM MEMORY

The FLASH Program Memory is readable, writable, and erasable during normal operation over the entire VDD range.

A read from program memory is executed on one byte at a time. A write to program memory is executed on blocks of 8 bytes at a time. Program memory is erased in blocks of 64 bytes at a time. A bulk erase operation may not be issued from user code.

Writing or erasing program memory will cease instruction fetches until the operation is complete. The program memory cannot be accessed during the write or erase, therefore, code cannot execute. An internal programming timer terminates program memory writes and erases.

A value written to program memory does not need to be a valid instruction. Executing a program memory location that forms an invalid instruction results in a `NOP`.

5.1 Table Reads and Table Writes

In order to read and write program memory, there are two operations that allow the processor to move bytes between the program memory space and the data RAM:

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

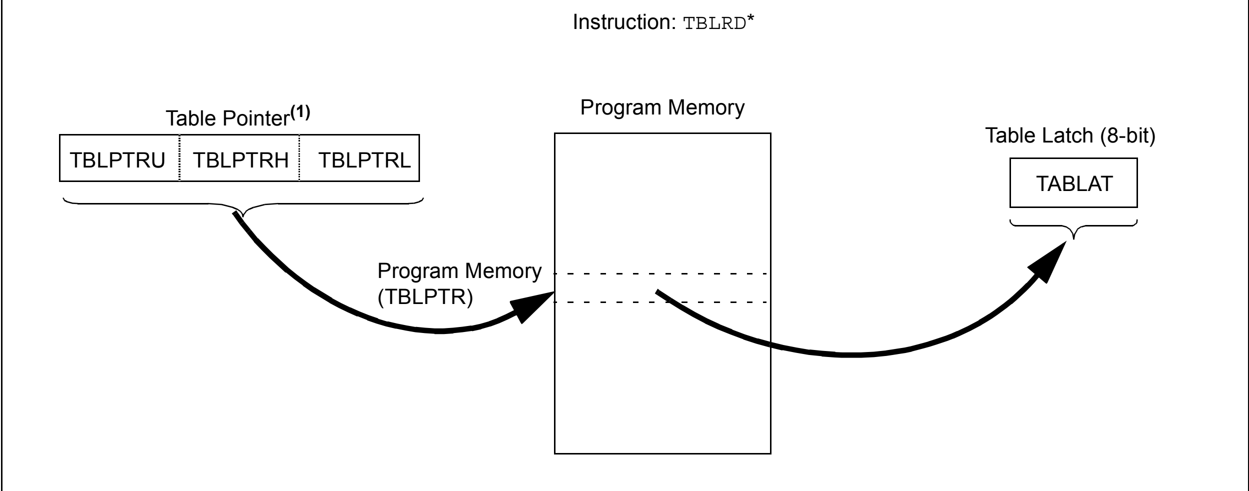
The program memory space is 16-bits wide, while the data RAM space is 8-bits wide. Table Reads and Table Writes move data between these two memory spaces through an 8-bit register (TABLAT).

Table Read operations retrieve data from program memory and places it into the data RAM space. Figure 5-1 shows the operation of a Table Read with program memory and data RAM.

Table Write operations store data from the data memory space into holding registers in program memory. The procedure to write the contents of the holding registers into program memory is detailed in Section 5.5, “Writing to FLASH Program Memory”. Figure 5-2 shows the operation of a Table Write with program memory and data RAM.

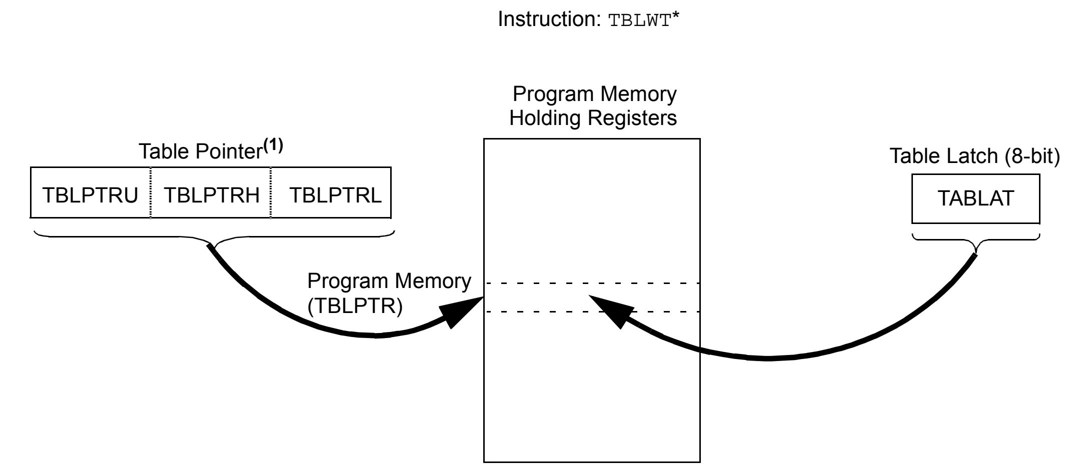
Table operations work with byte entities. A table block containing data, rather than program instructions, is not required to be word aligned. Therefore, a table block can start and end at any byte address. If a Table Write is being used to write executable code into program memory, program instructions will need to be word aligned.

FIGURE 5-1: TABLE READ OPERATION



Note 1: Table Pointer points to a byte in program memory.

FIGURE 5-2: TABLE WRITE OPERATION



Note 1: Table Pointer actually points to one of eight holding registers, the address of which is determined by TBLPTRL<2:0>. The process for physically writing data to the Program Memory Array is discussed in Section 5.5.

5.2 Control Registers

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

- EECON1 register
- EECON2 register
- TABLAT register
- TBLPTR registers

5.2.1 EECON1 AND EECON2 REGISTERS

EECON1 is the control register for memory accesses.

EECON2 is not a physical register. Reading EECON2 will read all '0's. The EECON2 register is used exclusively in the memory write and erase sequences.

Control bit EEPGD determines if the access will be a program or data EEPROM memory access. When clear, any subsequent operations will operate on the data EEPROM memory. When set, any subsequent operations will operate on the program memory.

Control bit CFGS determines if the access will be to the configuration registers or to program memory/data EEPROM memory. When set, subsequent operations will operate on configuration registers, regardless of EEPGD (see Section 20.0, “Special Features of the CPU”). When clear, memory selection access is determined by EEPGD.

The FREE bit, when set, will allow a program memory erase operation. When the FREE bit is set, the erase operation is initiated on the next WR command. When FREE is clear, only writes are enabled.

The WREN bit, when set, will allow a write operation. On power-up, the WREN bit is clear. The WRERR bit is set when a write operation is interrupted by a $\overline{\text{MCLR}}$ Reset, or a WDT Time-out Reset, during normal operation. In these situations, the user can check the WRERR bit and rewrite the location. It is necessary to reload the data and address registers (EEDATA and EEADR), due to `RESET` values of zero.

Control bit WR initiates write operations. This bit cannot be cleared, only set, in software. It is cleared in hardware at the completion of the write operation. The inability to clear the WR bit in software prevents the accidental or premature termination of a write operation.

Note: Interrupt flag bit, EEIF in the PIR2 register, is set when the write is complete. It must be cleared in software.

REGISTER 5-1: EECON1 REGISTER (ADDRESS FA6h)

R/W-x	R/W-x	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	CFGS	—	FREE	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7 **EEPGD:** FLASH Program or Data EEPROM Memory Select bit
1 = Access FLASH program memory
0 = Access data EEPROM memory

bit 6 **CFGS:** FLASH Program/Data EE or Configuration Select bit
1 = Access configuration registers
0 = Access FLASH program or data EEPROM memory

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

bit 4 **FREE:** FLASH Row Erase Enable bit
1 = Erase the program memory row addressed by TBLPTR on the next WR command (cleared by completion of erase operation)
0 = Perform write only

bit 3 **WRERR:** FLASH Program/Data EE Error Flag bit
1 = A write operation is prematurely terminated (any RESET during self-timed programming in normal operation)
0 = The write operation completed

Note: When a WRERR occurs, the EEPGD and CFGS bits are not cleared. This allows tracing of the error condition.

bit 2 **WREN:** FLASH Program/Data EE Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR:** Write Control bit
1 = Initiates a data EEPROM erase/write cycle or a program memory erase cycle or write cycle. (The operation is self-timed and the bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.)
0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit
1 = Initiates an EEPROM read (Read takes one cycle. RD is cleared in hardware. The RD bit can only be set (not cleared) in software. RD bit cannot be set when EEPGD = 1.)
0 = Does not initiate an EEPROM read

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

5.2.2 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 RAM.

5.2.3 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, Table Pointer High Byte and Table Pointer Low Byte (TBLPTRU:TBLPTRH:TBLPTRL). These three registers join to form a 22-bit wide pointer. The low order 21 bits allow the device to address up to 2 Mbytes 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 5-1. These operations on the TBLPTR only affect the low order 21 bits.

5.2.4 TABLE POINTER BOUNDARIES

TBLPTR is used in reads, writes, and erases of the FLASH program memory.

When a `TBLRD` is executed, all 22 bits of the Table Pointer determine which byte is read from program memory into TABLAT.

When a `TBLWT` is executed, the three LSbs of the Table Pointer (TBLPTR<2:0>) determine which of the eight program memory holding registers is written to. When the timed write to program memory (long write) begins, the 19 MSbs of the Table Pointer, TBLPTR (TBLPTR<21:3>), will determine which program memory block of 8 bytes is written to. For more detail, see Section 5.5 (“Writing to FLASH Program Memory”).

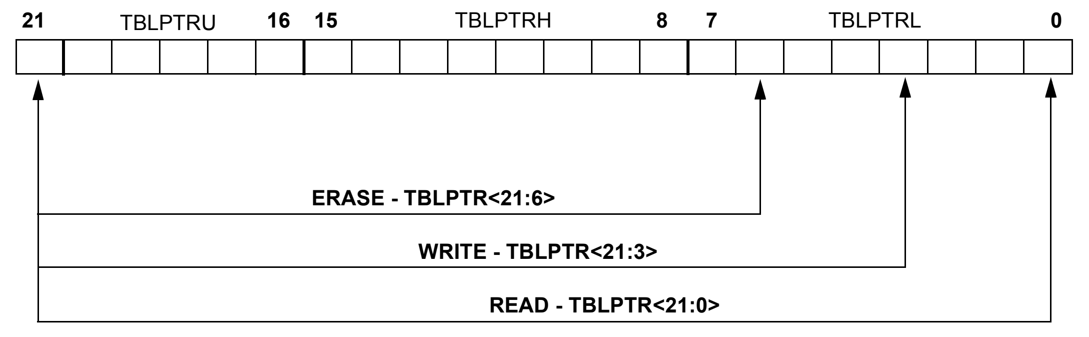
When an erase of program memory is executed, the 16 MSbs of the Table Pointer (TBLPTR<21:6>) point to the 64-byte block that will be erased. The Least Significant bits (TBLPTR<5:0>) are ignored.

Figure 5-3 describes the relevant boundaries of TBLPTR based on FLASH program memory operations.

TABLE 5-1: TABLE POINTER OPERATIONS WITH `TBLRD` AND `TBLWT` INSTRUCTIONS

Example	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

FIGURE 5-3: TABLE POINTER BOUNDARIES BASED ON OPERATION



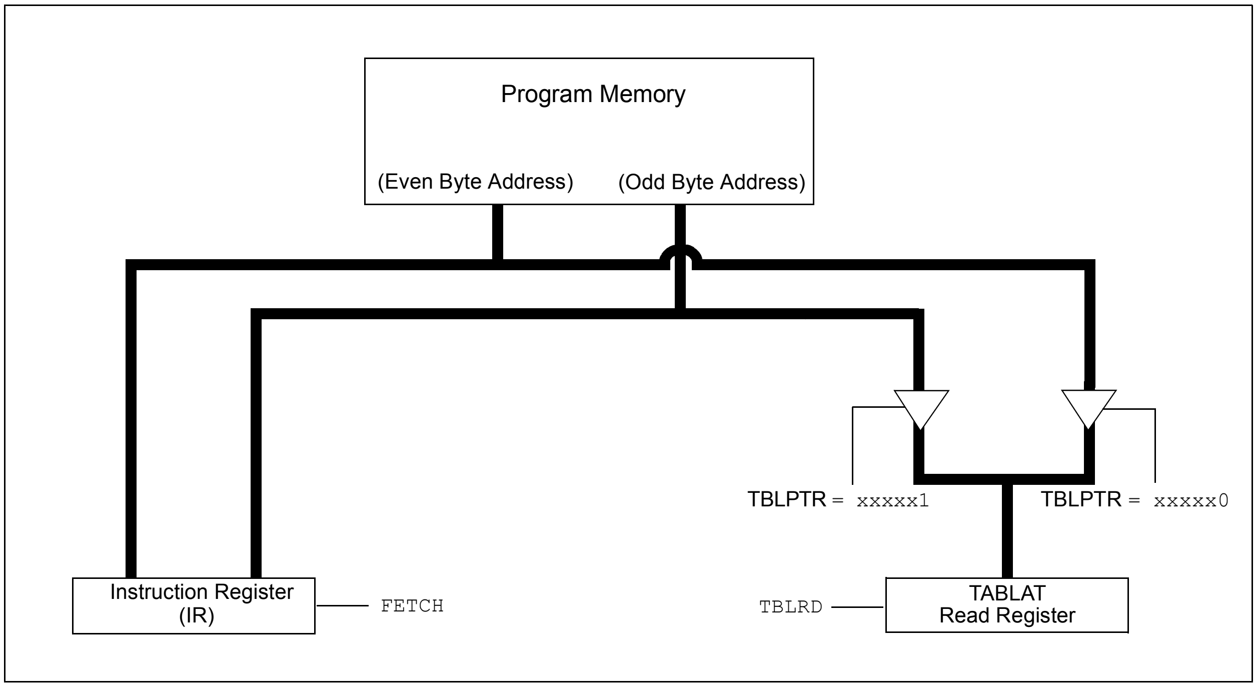
5.3 Reading the FLASH Program Memory

The `TBLRD` instruction is used to retrieve data from program memory and place into data RAM. Table Reads from program memory are performed one byte at a time.

TBLPTR points to a byte address in program space. Executing `TBLRD` places the byte pointed to into TABLAT. In addition, TBLPTR can be modified automatically for the next Table Read operation.

The internal program memory is typically organized by words. The Least Significant bit of the address selects between the high and low bytes of the word. Figure 5-4 shows the interface between the internal program memory and the TABLAT.

FIGURE 5-4: READS FROM FLASH PROGRAM MEMORY



EXAMPLE 5-1: READING A FLASH PROGRAM MEMORY WORD

```
           MOVLW CODE_ADDR_UPPER         ; Load TBLPTR with the base
           MOVWF TBLPTRU                 ; address of the word
           MOVLW CODE_ADDR_HIGH
           MOVWF TBLPTRH
           MOVLW CODE_ADDR_LOW
           MOVWF TBLPTRL
READ_WORD
           TBLRD*+                       ; read into TABLAT and increment
           MOVF TABLAT, W                ; get data
           MOVWF WORD_EVEN
           TBLRD*+                       ; read into TABLAT and increment
           MOVF TABLAT, W                ; get data
           MOVWF WORD_ODD
```

5.4 Erasing FLASH Program memory

The minimum erase block is 32 words or 64 bytes. Only through the use of an external programmer, or through ICSP control can larger blocks of program memory be bulk erased. Word erase in the FLASH array is not supported.

When initiating an erase sequence from the micro-controller itself, a block of 64 bytes of program memory is erased. The Most Significant 16 bits of the TBLPTR<21:6> point to the block being erased; TBLPTR<5:0> are ignored.

The EECON1 register commands the erase operation. The EEPGD bit must be set to point to the FLASH program memory. The WREN bit must be set to enable write operations. The FREE bit is set to select an erase operation.

For protection, the write initiate sequence for EECON2 must be used.

A long write is necessary for erasing the internal FLASH. Instruction execution is halted while in a long write cycle. The long write will be terminated by the internal programming timer.

5.4.1 FLASH PROGRAM MEMORY ERASE SEQUENCE

The sequence of events for erasing a block of internal program memory location is:

1. Load table pointer with address of row being erased.
2. Set EEPGD bit to point to program memory, clear CFGS bit to access program memory, set WREN bit to enable writes, and set FREE bit to enable the erase.
3. Disable interrupts.
4. Write 55h to EECON2.
5. Write AAh to EECON2.
6. Set the WR bit. This will begin the row erase cycle.
7. The CPU will stall for duration of the erase (about 2 ms using internal timer).
8. Re-enable interrupts.

EXAMPLE 5-2: ERASING A FLASH PROGRAM MEMORY ROW

```
                 MOVLW   CODE_ADDR_UPPER        ; load TBLPTR with the base
                 MOVWF   TBLPTRU                ; address of the memory block
                 MOVLW   CODE_ADDR_HIGH
                 MOVWF   TBLPTRH
                 MOVLW   CODE_ADDR_LOW
                 MOVWF   TBLPTRL
      ERASE_ROW
                 BSF     EECON1,EEPGD           ; point to FLASH program memory
                 BCF     EECON1,CFGS            ; access FLASH program memory
                 BSF     EECON1,WREN            ; enable write to memory
                 BSF     EECON1,FREE            ; enable Row Erase operation
                 BCF     INTCON,GIE             ; disable interrupts
                 MOVLW   55h
    Required     MOVWF   EECON2                 ; write 55h
    Sequence     MOVLW   AAh
                 MOVWF   EECON2                 ; write AAh
                 BSF     EECON1,WR              ; start erase (CPU stall)
                 BSF     INTCON,GIE             ; re-enable interrupts
```

5.5 Writing to FLASH Program Memory

The minimum programming block is 4 words or 8 bytes. Word or byte programming is not supported.

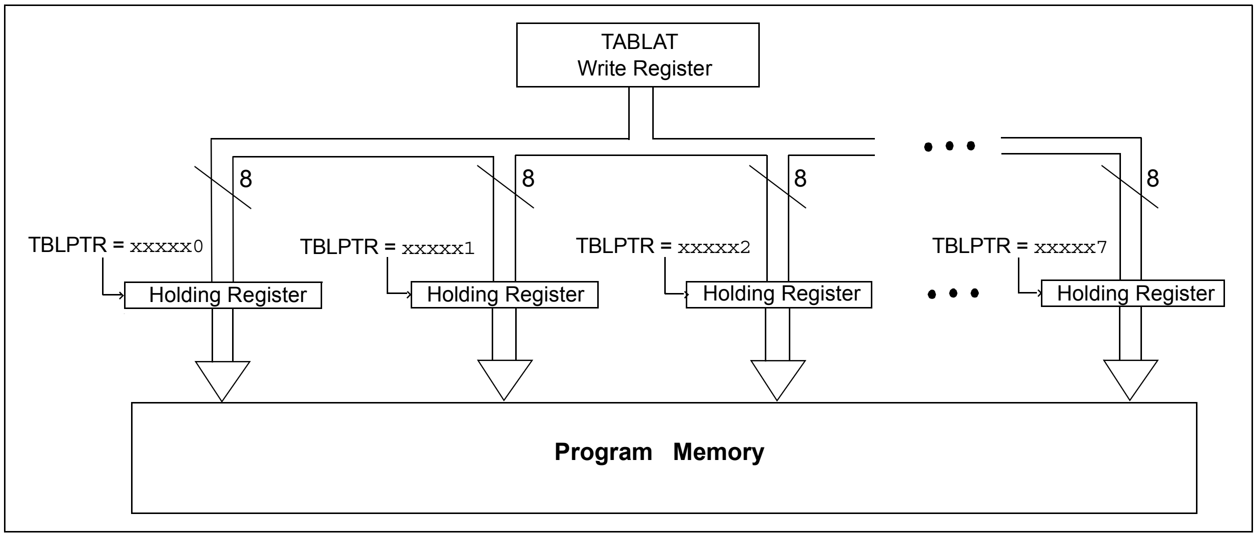
Table Writes are used internally to load the holding registers needed to program the FLASH memory. There are 8 holding registers used by the Table Writes for programming.

Since the Table Latch (TABLAT) is only a single byte, the `TBLWT` instruction has to be executed 8 times for each programming operation. All of the Table Write operations will essentially be short writes, because only the holding registers are written. At the end of updating 8 registers, the EECON1 register must be written to, to start the programming operation with a long write.

The long write is necessary for programming the internal FLASH. Instruction execution is halted while in a long write cycle. The long write will be terminated by the internal programming timer.

The EEPROM on-chip timer controls the write time. The write/erase voltages are generated by an on-chip charge pump rated to operate over the voltage range of the device for byte or word operations.

FIGURE 5-5: TABLE WRITES TO FLASH PROGRAM MEMORY



5.5.1 FLASH PROGRAM MEMORY WRITE SEQUENCE

The sequence of events for programming an internal program memory location should be:

1. Read 64 bytes into RAM.
2. Update data values in RAM as necessary.
3. Load Table Pointer with address being erased.
4. Do the row erase procedure.
5. Load Table Pointer with address of first byte being written.
6. Write the first 8 bytes into the holding registers with auto-increment (`TBLWT*+` or `TBLWT+*`).
7. Set EEPGD bit to point to program memory, clear the CFGS bit to access program memory, and set WREN to enable byte writes.
8. Disable interrupts.
9. Write 55h to EECON2.
10. Write AAh to EECON2.
11. Set the WR bit. This will begin the write cycle.
12. The CPU will stall for duration of the write (about 2 ms using internal timer).
13. Re-enable interrupts.
14. Repeat steps 6-14 seven times, to write 64 bytes.
15. Verify the memory (Table Read).

This procedure will require about 18 ms to update one row of 64 bytes of memory. An example of the required code is given in Example 5-3.

Note:	Before setting the WR bit, the table pointer address needs to be within the intended address range of the 8 bytes in the holding registers.

EXAMPLE 5-3: WRITING TO FLASH PROGRAM MEMORY

```
           MOVLW   D'64                     ; number of bytes in erase block
           MOVWF   COUNTER
           MOVLW   BUFFER_ADDR_HIGH         ; point to buffer
           MOVWF   FSR0H
           MOVLW   BUFFER_ADDR_LOW
           MOVWF   FSR0L
           MOVLW   CODE_ADDR_UPPER          ; Load TBLPTR with the base
           MOVWF   TBLPTRU                  ; address of the memory block
           MOVLW   CODE_ADDR_HIGH
           MOVWF   TBLPTRH
           MOVLW   CODE_ADDR_LOW
           MOVWF   TBLPTRL
READ_BLOCK
           TBLRD*+                          ; read into TABLAT, and inc
           MOVF    TABLAT, W                ; get data
           MOVWF   POSTINC0                 ; store data
           DECFSZ  COUNTER                  ; done?
           BRA     READ_BLOCK               ; repeat
MODIFY_WORD
           MOVLW   DATA_ADDR_HIGH           ; point to buffer
           MOVWF   FSR0H
           MOVLW   DATA_ADDR_LOW
           MOVWF   FSR0L
           MOVLW   NEW_DATA_LOW             ; update buffer word
           MOVWF   POSTINC0
           MOVLW   NEW_DATA_HIGH
           MOVWF   INDF0
ERASE_BLOCK
           MOVLW   CODE_ADDR_UPPER          ; load TBLPTR with the base
           MOVWF   TBLPTRU                  ; address of the memory block
           MOVLW   CODE_ADDR_HIGH
           MOVWF   TBLPTRH
           MOVLW   CODE_ADDR_LOW
           MOVWF   TBLPTRL
           BSF     EECON1,EEPGD             ; point to FLASH program memory
           BCF     EECON1,CFGS              ; access FLASH program memory
           BSF     EECON1,WREN              ; enable write to memory
           BSF     EECON1,FREE              ; enable Row Erase operation
           BCF     INTCON,GIE               ; disable interrupts
           MOVLW   55h
           MOVWF   EECON2                   ; write 55h
           MOVLW   AAh
           MOVWF   EECON2                   ; write AAh
           BSF     EECON1,WR                ; start erase (CPU stall)
           BSF     INTCON,GIE               ; re-enable interrupts
           TBLRD*-                          ; dummy read decrement
WRITE_BUFFER_BACK
           MOVLW   8                        ; number of write buffer groups of 8 bytes
           MOVWF   COUNTER_HI
           MOVLW   BUFFER_ADDR_HIGH         ; point to buffer
           MOVWF   FSR0H
           MOVLW   BUFFER_ADDR_LOW
           MOVWF   FSR0L
PROGRAM_LOOP
           MOVLW   8                        ; number of bytes in holding register
           MOVWF   COUNTER
WRITE_WORD_TO_HREGS
           MOVF    POSTINC0, W              ; get low byte of buffer data
           MOVWF   TABLAT                   ; present data to table latch
           TBLWT+*                          ; write data, perform a short write
                                            ; to internal TBLWT holding register.
           DECFSZ  COUNTER                  ; loop until buffers are full
           BRA     WRITE_WORD_TO_HREGS
```

EXAMPLE 5-3: WRITING TO FLASH PROGRAM MEMORY (CONTINUED)

```
PROGRAM_MEMORY
          BSF     EECON1,EEPGD           ; point to FLASH program memory
          BCF     EECON1,CFGS            ; access FLASH program memory
          BSF     EECON1,WREN            ; enable write to memory
          BCF     INTCON,GIE             ; disable interrupts
          MOVLW   55h
Required  MOVWF   EECON2                 ; write 55h
Sequence  MOVLW   AAh
          MOVWF   EECON2                 ; write AAh
          BSF     EECON1,WR              ; start program (CPU stall)
          BSF     INTCON,GIE             ; re-enable interrupts
          DECFSZ  COUNTER_HI             ; loop until done
          BRA     PROGRAM_LOOP
          BCF     EECON1,WREN            ; disable write to memory
```

5.5.2 WRITE VERIFY

Depending on the application, good programming practice may dictate that the value written to the memory should be verified against the original value. This should be used in applications where excessive writes can stress bits near the specification limit.

5.5.3 UNEXPECTED TERMINATION OF WRITE OPERATION

If a write is terminated by an unplanned event, such as loss of power or an unexpected RESET, the memory location just programmed should be verified and reprogrammed if needed.The WRERR bit is set when a write operation is interrupted by a $\overline{\text{MCLR}}$ Reset, or a WDT Time-out Reset during normal operation. In these situations, users can check the WRERR bit and rewrite the location.

5.5.4 PROTECTION AGAINST SPURIOUS WRITES

To protect against spurious writes to FLASH program memory, the write initiate sequence must also be followed. See “Special Features of the CPU” (Section 20.0) for more detail.

5.6 FLASH Program Operation During Code Protection

See “Special Features of the CPU” (Section 20.0) for details on code protection of FLASH program memory.

TABLE 5-2: REGISTERS ASSOCIATED WITH PROGRAM FLASH MEMORY

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
TBLPTRU	—	—	bit 21	Program Memory Table Pointer Upper Byte (TBLPTR<20:16>)					`--00 0000`	`--00 0000`
TBPLTRH	Program Memory Table Pointer High Byte (TBLPTR<15:8>)								`0000 0000`	`0000 0000`
TBLPTRL	Program Memory Table Pointer High Byte (TBLPTR<7:0>)								`0000 0000`	`0000 0000`
TABLAT	Program Memory Table Latch								`0000 0000`	`0000 0000`
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	`0000 000x`	`0000 000u`
EECON2	EEPROM Control Register2 (not a physical register)								—	—
EECON1	EEPGD	CFGS	—	FREE	WRERR	WREN	WR	RD	`xx-0 x000`	`uu-0 u000`
IPR2	—	—	—	EEIP	BCLIP	LVDIP	TMR3IP	—	`---1 1111`	`---1 1111`
PIR2	—	—	—	EEIF	BCLIF	LVDIF	TMR3IF	—	`---0 0000`	`---0 0000`
PIE2	—	—	—	EEIE	BCLIE	LVDIE	TMR3IE	—	`---0 0000`	`---0 0000`

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'. Shaded cells are not used during FLASH/EEPROM access.

NOTES:

6.0 DATA EEPROM MEMORY

The Data EEPROM is readable and writable during normal operation over the entire VDD range. The data memory is not directly mapped in the register file space. Instead, it is indirectly addressed through the Special Function Registers (SFR).

There are four SFRs used to read and write the program and data EEPROM memory. These registers are:

- EECON1
- EECON2
- EEDATA
- EEADR

The EEPROM data memory allows byte read and write. When interfacing to the data memory block, EEDATA holds the 8-bit data for read/write and EEADR holds the address of the EEPROM location being accessed. These devices have 256 bytes of data EEPROM with an address range from 0h to FFh.

The EEPROM data memory is rated for high erase/write cycles. A byte write automatically erases the location and writes the new data (erase-before-write). The write time is controlled by an on-chip timer. The write time will vary with voltage and temperature, as well as from chip to chip. Please refer to parameter D122 (Electrical Characteristics, Section 23.0) for exact limits.

6.1 EEADR

The address register can address up to a maximum of 256 bytes of data EEPROM.

6.2 EECON1 and EECON2 Registers

EECON1 is the control register for EEPROM memory accesses.

EECON2 is not a physical register. Reading EECON2 will read all '0's. The EECON2 register is used exclusively in the EEPROM write sequence.

Control bits RD and WR initiate read and write operations, respectively. These bits cannot be cleared, only set, in software. They are cleared in hardware at the completion of the read or write operation. The inability to clear the WR bit in software prevents the accidental or premature termination of a write operation.

The WREN bit, when set, will allow a write operation. On power-up, the WREN bit is clear. The WRERR bit is set when a write operation is interrupted by a $\overline{\text{MCLR}}$ Reset, or a WDT Time-out Reset during normal operation. In these situations, the user can check the WRERR bit and rewrite the location. It is necessary to reload the data and address registers (EEDATA and EEADR), due to the RESET condition forcing the contents of the registers to zero.

Note:	Interrupt flag bit, EEIF in the PIR2 register, is set when write is complete. It must be cleared in software.

REGISTER 6-1: EECON1 REGISTER (ADDRESS FA6h)

R/W-x	R/W-x	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	CFGS	—	FREE	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7 **EEPGD:** FLASH Program or Data EEPROM Memory Select bit
1 = Access FLASH program memory
0 = Access data EEPROM memory

bit 6 **CFGS:** FLASH Program/Data EE or Configuration Select bit
1 = Access configuration or calibration registers
0 = Access FLASH program or data EEPROM memory

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

bit 4 **FREE:** FLASH Row Erase Enable bit
1 = Erase the program memory row addressed by TBLPTR on the next WR command (cleared by completion of erase operation)
0 = Perform write only

bit 3 **WRERR:** FLASH Program/Data EE Error Flag bit
1 = A write operation is prematurely terminated (any $\overline{\text{MCLR}}$ or any WDT Reset during self-timed programming in normal operation)
0 = The write operation completed

Note: When a WRERR occurs, the EEPGD or FREE bits are not cleared. This allows tracing of the error condition.

bit 2 **WREN:** FLASH Program/Data EE Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR:** Write Control bit
1 = Initiates a data EEPROM erase/write cycle or a program memory erase cycle or write cycle. (The operation is self-timed and the bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.)
0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit
1 = Initiates an EEPROM read (Read takes one cycle. RD is cleared in hardware. The RD bit can only be set (not cleared) in software. RD bit cannot be set when EEPGD = 1.)
0 = Does not initiate an EEPROM read

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

6.3 Reading the Data EEPROM Memory

To read a data memory location, the user must write the address to the EEADR register, clear the EEPGD control bit (EECON1<7>), clear the CFGS control bit (EECON1<6>), and then set control bit RD (EECON1<0>). The data is available for the very next instruction cycle; therefore, the EEDATA register can be read by the next instruction. EEDATA will hold this value until another read operation, or until it is written to by the user (during a write operation).

EXAMPLE 6-1: DATA EEPROM READ

```
        MOVLW   DATA_EE_ADDR   ;
        MOVWF   EEADR          ; Data Memory Address to read
        BCF     EECON1, EEPGD  ; Point to DATA memory
        BCF     EECON1, CFGS   ; Access program FLASH or Data EEPROM memory
        BSF     EECON1, RD     ; EEPROM Read
        MOVF    EEDATA, W      ; W = EEDATA
```

6.4 Writing to the Data EEPROM Memory

To write an EEPROM data location, the address must first be written to the EEADR register and the data written to the EEDATA register. Then, the sequence in Example 6-2 must be followed to initiate the write cycle.

The write will not initiate if the above sequence is not exactly followed (write 55h to EECON2, write AAh to EECON2, then set WR bit) for each byte. It is strongly recommended that interrupts be disabled during this code segment.

Additionally, the WREN bit in EECON1 must be set to enable writes. This mechanism prevents accidental writes to data EEPROM due to unexpected code execution (i.e., runaway programs). The WREN bit should be kept clear at all times, except when updating the EEPROM. The WREN bit is not cleared by hardware.

After a write sequence has been initiated, EECON1, EEADR and EDATA cannot be modified. The WR bit will be inhibited from being set unless the WREN bit is set. The WREN bit must be set on a previous instruction. Both WR and WREN cannot be set with the same instruction.

At the completion of the write cycle, the WR bit is cleared in hardware and the EEPROM Write Complete Interrupt Flag bit (EEIF) is set. The user may either enable this interrupt, or poll this bit. EEIF must be cleared by software.

EXAMPLE 6-2: DATA EEPROM WRITE

```
            MOVLW   DATA_EE_ADDR   ;
            MOVWF   EEADR          ; Data Memory Address to read
            MOVLW   DATA_EE_DATA   ;
            MOVWF   EEDATA         ; Data Memory Value to write
            BCF     EECON1, EEPGD  ; Point to DATA memory
            BCF     EECON1, CFGS   ; Access program FLASH or Data EEPROM memory
            BSF     EECON1, WREN   ; Enable writes


            BCF     INTCON, GIE    ; Disable interrupts
Required    MOVLW   55h            ;
Sequence    MOVWF   EECON2         ; Write 55h
            MOVLW   AAh            ;
            MOVWF   EECON2         ; Write AAh
            BSF     EECON1, WR     ; Set WR bit to begin write
            BSF     INTCON, GIE    ; Enable interrupts


            .                      ; user code execution
            .
            .
            BCF     EECON1, WREN   ; Disable writes on write complete (EEIF set)
```

6.5 Write Verify

Depending on the application, good programming practice may dictate that the value written to the memory should be verified against the original value. This should be used in applications where excessive writes can stress bits near the specification limit.

6.6 Protection Against Spurious Write

There are conditions when the device may not want to write to the data EEPROM memory. To protect against spurious EEPROM writes, various mechanisms have been built-in. On power-up, the WREN bit is cleared. Also, the Power-up Timer (72 ms duration) prevents EEPROM write.

The write initiate sequence and the WREN bit together help prevent an accidental write during brown-out, power glitch, or software malfunction.

6.7 Operation During Code Protect

Data EEPROM memory has its own code protect mechanism. External read and write operations are disabled if either of these mechanisms are enabled.

The microcontroller itself can both read and write to the internal Data EEPROM, regardless of the state of the code protect configuration bit. Refer to “Special Features of the CPU” (Section 20.0) for additional information.

6.8 Using the Data EEPROM

The data EEPROM is a high endurance, byte addressable array that has been optimized for the storage of frequently changing information (e.g., program variables or other data that are updated often). Frequently changing values will typically be updated more often than specification D124. If this is not the case, an array refresh must be performed. For this reason, variables that change infrequently (such as constants, IDs, calibration, etc.) should be stored in FLASH program memory.

A simple data EEPROM refresh routine is shown in Example 6-3.

Note:	If data EEPROM is only used to store constants and/or data that changes rarely, an array refresh is likely not required. See specification D124.

EXAMPLE 6-3: DATA EEPROM REFRESH ROUTINE

```
       clrf    EEADR                 ; Start at address 0
       bcf     EECON1,CFGS           ; Set for memory
       bcf     EECON1,EEPGD          ; Set for Data EEPROM
       bcf     INTCON,GIE            ; Disable interrupts
       bsf     EECON1,WREN           ; Enable writes
Loop                                 ; Loop to refresh array
       bsf     EECON1,RD             ; Read current address
       movlw   55h                   ;
       movwf   EECON2                ; Write 55h
       movlw   AAh                   ;
       movwf   EECON2                ; Write AAh
       bsf     EECON1,WR             ; Set WR bit to begin write
       btfsc   EECON1,WR             ; Wait for write to complete
       bra     $-2
       incfsz  EEADR,F               ; Increment address
       bra     Loop                  ; Not zero, do it again

       bcf     EECON1,WREN           ; Disable writes
       bsf     INTCON,GIE            ; Enable interrupts
```

TABLE 6-1: REGISTERS ASSOCIATED WITH DATA EEPROM MEMORY

Address	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
FF2h	INTCON	GIE/ GIEH	PEIE/ GIEL	T0IE	INTE	RBIE	T0IF	INTF	RBIF	`0000 000x`	`0000 000u`
FA9h	EEADR	EEPROM Address Register								`0000 0000`	`0000 0000`
FA8h	EEDATA	EEPROM Data Register								`0000 0000`	`0000 0000`
FA7h	EECON2	EEPROM Control Register2 (not a physical register)								—	—
FA6h	EECON1	EEPGD	CFGS	—	FREE	WRERR	WREN	WR	RD	`xx-0 x000`	`uu-0 u000`
FA2h	IPR2	—	—	—	EEIP	BCLIP	LVDIP	TMR3IP	—	`---1 1111`	`---1 1111`
FA1h	PIR2	—	—	—	EEIF	BCLIF	LVDIF	TMR3IF	—	`---0 0000`	`---0 0000`
FA0h	PIE2	—	—	—	EEIE	BCLIE	LVDIE	TMR3IE	—	`---0 0000`	`---0 0000`

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, read as '0'.
Shaded cells are not used during FLASH/EEPROM access.

NOTES:

7.0 8 X 8 HARDWARE MULTIPLIER

7.1 Introduction

An 8 x 8 hardware multiplier is included in the ALU of the PIC18FXX39 devices. By making the multiply a hardware operation, it completes in a single instruction cycle. This is an unsigned multiply that gives a 16-bit result. The result is stored into the 16-bit product register pair (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 multiply algorithms

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

Table 7-1 shows a performance comparison between enhanced devices using the single cycle hardware multiply, and performing the same function without the hardware multiply.

7.2 Operation

Example 7-1 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 7-2 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 7-1: 8 x 8 UNSIGNED MULTIPLY ROUTINE

```
MOVF    ARG1, W        ;
MULWF   ARG2           ; ARG1 * ARG2 ->
                       ; PRODH:PRODL
```

EXAMPLE 7-2: 8 x 8 SIGNED MULTIPLY ROUTINE

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

TABLE 7-1: PERFORMANCE COMPARISON

Routine	Multiply Method	Program Memory (Words)	Cycles (Max)	Time		
				@ 40 MHz	@ 10 MHz	@ 4 MHz
8 x 8 unsigned	Without hardware multiply	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 multiply	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 multiply	21	242	24.2 μs	96.8 μs	242 μs
	Hardware multiply	28	28	2.8 μs	11.2 μs	28 μs
16 x 16 signed	Without hardware multiply	52	254	25.4 μs	102.6 μs	254 μs
	Hardware multiply	35	40	4.0 μs	16.0 μs	40 μs

Example 7-3 shows the sequence to do a 16 x 16 unsigned multiply. Equation 7-1 shows the algorithm that is used. The 32-bit result is stored in four registers, RES3:RES0.

EQUATION 7-1: 16 x 16 UNSIGNED MULTIPLICATION ALGORITHM

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

EXAMPLE 7-3: 16 x 16 UNSIGNED MULTIPLY ROUTINE

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

Example 7-4 shows the sequence to do a 16 x 16 signed multiply. Equation 7-2 shows the algorithm used. The 32-bit result is stored in four registers, RES3: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 7-2: 16 x 16 SIGNED MULTIPLICATION ALGORITHM

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

EXAMPLE 7-4: 16 x 16 SIGNED MULTIPLY ROUTINE

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

8.0 INTERRUPTS

The PIC18FXX39 devices have multiple interrupt sources and an interrupt priority feature that allows each interrupt source to be assigned a high priority level or a low priority level. The high priority interrupt vector is at 000008h and the low priority interrupt vector is at 000018h. High priority interrupt events will override any low priority interrupts that may be in progress.

There are ten registers which are used to control interrupt operation. These registers are:

- RCON
- INTCON
- INTCON2
- INTCON3
- PIR1, PIR2
- PIE1, PIE2
- IPR1, IPR2

It is recommended that the Microchip header files supplied with MPLAB® IDE be used for the symbolic bit names in these registers. This allows the assembler/compiler to automatically take care of the placement of these bits within the specified register.

Each interrupt source, except INT0, has three bits to control its operation. The functions of these bits are:

- Flag bit to indicate that an interrupt event occurred
- Enable bit that allows program execution to branch to the interrupt vector address when the flag bit is set
- Priority bit to select high priority or low priority

The interrupt priority feature is enabled by setting the IPEN bit (RCON<7>). When interrupt priority is enabled, there are two bits which enable interrupts globally. Setting the GIEH bit (INTCON<7>) enables all interrupts that have the priority bit set. Setting the GIEL bit (INTCON<6>) enables all interrupts that have the priority bit cleared. When the interrupt flag, enable bit and appropriate global interrupt enable bit are set, the interrupt will vector immediately to address 000008h or 000018h, depending on the priority level. Individual interrupts can be disabled through their corresponding enable bits.

While PIC18FXX39 devices have two interrupt priority levels like other PIC18 microcontrollers, their allocation is different. In these devices, the high priority interrupt is used exclusively by the ProMPT kernel via the Timer2 match interrupt. In order for the kernel to function properly, it is imperative that all other interrupts either set as low priority (IPR bit = 0), or disabled.

Note:	Disabling interrupts, or setting interrupts as low priority, is **not** the same as disabling interrupt priorities. The interrupt priority levels must remain enabled (IPEN = 1). Clearing the IPEN bit will result in erratic operation of the ProMPT kernel.

When an interrupt is responded to, the Global Interrupt Enable bit is cleared to disable further interrupts. If the IPEN bit is cleared, this is the GIE bit. If interrupt priority levels are used, this will be either the GIEH or GIEL bit. High priority interrupt sources can interrupt a low priority interrupt.

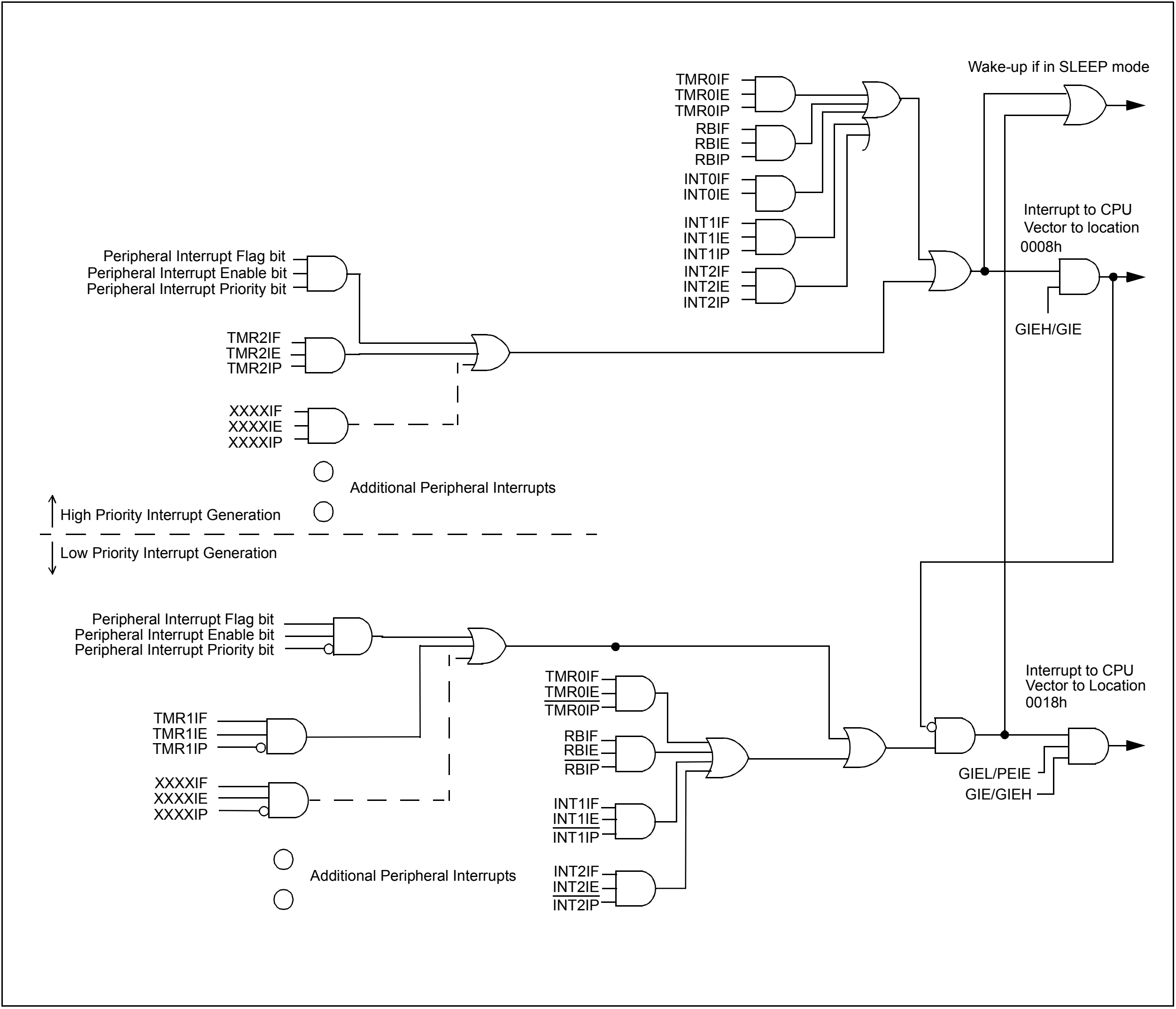
The return address is pushed onto the stack and the PC is loaded with the interrupt vector address (000008h or 000018h). Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bits must be cleared in software before re-enabling interrupts to avoid recursive interrupts.

The "return from interrupt" instruction, `RETFIE`, exits the interrupt routine and sets the GIEH or GIEL bits (as applicable), which re-enables interrupts.

For external interrupt events, such as the INT pins or the PORTB input change interrupt, the interrupt latency will be three to four instruction cycles. The exact latency is the same for one or two-cycle instructions. Individual interrupt flag bits are set, regardless of the status of their corresponding enable bit or the GIE bit.

Note:	Do not use the `MOVFF` instruction to modify any of the Interrupt control registers while **any** interrupt is enabled. Doing so may cause erratic microcontroller behavior.

FIGURE 8-1: INTERRUPT LOGIC



8.1 INTCON Registers

The INTCON Registers are readable and writable registers, which contain various enable, priority and flag bits.

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 8-1: 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(1)	RBIE	TMR0IF	INT0IF	RBIF(2)
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit
`1` = Enables all high priority interrupts
`0` = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit
`1` = Enables all low priority peripheral interrupts
`0` = Disables all low 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(1):** 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(2):** 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

Note 1: Maintain this bit cleared (= 0).

2: A mismatch condition will continue to set this bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared.

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

REGISTER 8-2: 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(1)
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 Interrupt 0 Edge Select bit `1` = Interrupt on rising edge `0` = Interrupt on falling edge
bit 5	**INTEDG1**: External Interrupt 1 Edge Select bit `1` = Interrupt on rising edge `0` = Interrupt on falling edge
bit 4	**INTEDG2**: External Interrupt 2 Edge Select bit `1` = Interrupt on rising edge `0` = Interrupt on falling edge
bit 3	**Unimplemented:** Read as '0'
bit 2	**TMR0IP(1)**: TMR0 Overflow Interrupt Priority bit `1` = High priority `0` = Low priority
bit 1	**Unimplemented:** Read as '0'
bit 0	**RBIP(1)**: RB Port Change Interrupt Priority bit `1` = High priority `0` = Low priority

Note 1: Maintain this bit cleared (= 0).

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	‘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 8-3: 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(1)	INT1IP(1)	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

bit 7 **INT2IP(1):** INT2 External Interrupt Priority bit
`1` = High priority
`0` = Low priority

bit 6 **INT1IP(1):** INT1 External Interrupt Priority bit
`1` = High priority
`0` = Low priority

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

Note 1: Maintain this bit cleared (= 0).

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	‘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.

8.2 PIR Registers

The PIR registers contain the individual flag bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Flag registers (PIR1, PIR2).

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>).

2: User software should ensure the appropriate interrupt flag bits are cleared prior to enabling an interrupt, and after servicing that interrupt.

REGISTER 8-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

R/W-0	R/W-0	R-0	R-0	R/W-0	U-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF(2)	TMR1IF
bit 7							bit 0

bit 7 **PSPIF(1):** 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 6 **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 5 **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 4 **TXIF**: USART Transmit Interrupt Flag bit (see Section 17.0 for details on TXIF functionality)
`1` = The USART transmit buffer, TXREG, is empty (cleared when TXREG is written)
`0` = The USART transmit buffer is full

bit 3 **SSPIF**: Master Synchronous Serial Port Interrupt Flag bit
`1` = The transmission/reception is complete (must be cleared in software)
`0` = Waiting to transmit/receive

bit 2 **Unimplemented:** Read as ‘0’

bit 1 **TMR2IF(2):** 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 0 **TMR1IF:** TMR1 Overflow Interrupt Flag bit
`1` = TMR1 register overflowed (must be cleared in software)
`0` = MR1 register did not overflow

Note 1: This bit is reserved on PIC18F2X39 devices; always maintain this bit clear.
2: This bit is reserved for use by the ProMPT kernel; do not alter its value.

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

REGISTER 8-5: PIR2: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 2

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
—	—	—	EEIF	BCLIF	LVDIF	TMR3IF	—
bit 7							bit 0

bit 7-5	**Unimplemented:** Read as '0'
bit 4	**EEIF**: Data EEPROM/FLASH Write Operation Interrupt Flag bit 1 = The write operation is complete (must be cleared in software) 0 = The write operation is not complete, or has not been started
bit 3	**BCLIF**: Bus Collision Interrupt Flag bit 1 = A bus collision occurred (must be cleared in software) 0 = No bus collision occurred
bit 2	**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
bit 1	**TMR3IF**: TMR3 Overflow Interrupt Flag bit 1 = TMR3 register overflowed (must be cleared in software) 0 = TMR3 register did not overflow
bit 0	**Unimplemented:** Read as ‘0’

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

8.3 PIE Registers

The PIE registers contain the individual enable bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Enable registers (PIE1, PIE2). When IPEN = 0, the PEIE bit must be set to enable any of these peripheral interrupts.

REGISTER 8-6: PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE(2)	TMR1IE
bit 7							bit 0

bit 7	**PSPIE(1):** Parallel Slave Port Read/Write Interrupt Enable bit `1` = Enables the PSP read/write interrupt `0` = Disables the PSP read/write interrupt
bit 6	**ADIE**: A/D Converter Interrupt Enable bit `1` = Enables the A/D interrupt `0` = Disables the A/D interrupt
bit 5	**RCIE**: USART Receive Interrupt Enable bit `1` = Enables the USART receive interrupt `0` = Disables the USART receive interrupt
bit 4	**TXIE**: USART Transmit Interrupt Enable bit `1` = Enables the USART transmit interrupt `0` = Disables the USART transmit interrupt
bit 3	**SSPIE**: Master Synchronous Serial Port Interrupt Enable bit `1` = Enables the MSSP interrupt `0` = Disables the MSSP interrupt
bit 2	**Unimplemented:** Read as ‘0’
bit 1	**TMR2IE(2):** TMR2 to PR2 Match Interrupt Enable bit `1` = Enables the TMR2 to PR2 match interrupt `0` = Disables the TMR2 to PR2 match interrupt
bit 0	**TMR1IE**: TMR1 Overflow Interrupt Enable bit `1` = Enables the TMR1 overflow interrupt `0` = Disables the TMR1 overflow interrupt

Note 1: This bit is reserved on PIC18F2X39 devices; always maintain this bit clear.
2: This bit is reserved for use by the ProMPT kernel; do not alter its value.

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

REGISTER 8-7: PIE2: PERIPHERAL INTERRUPT ENABLE REGISTER 2

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
—	—	—	EEIE	BCLIE	LVDIE	TMR3IE	—
bit 7							bit 0

bit 7-5	**Unimplemented:** Read as '0'
bit 4	**EEIE**: Data EEPROM/FLASH Write Operation Interrupt Enable bit `1` = Enabled `0` = Disabled
bit 3	**BCLIE**: Bus Collision Interrupt Enable bit `1` = Enabled `0` = Disabled
bit 2	**LVDIE**: Low Voltage Detect Interrupt Enable bit `1` = Enabled `0` = Disabled
bit 1	**TMR3IE**: TMR3 Overflow Interrupt Enable bit `1` = Enables the TMR3 overflow interrupt `0` = Disables the TMR3 overflow interrupt
bit 0	**Unimplemented:** Read as ‘0’

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

8.4 IPR Registers

The IPR registers contain the individual priority bits for the peripheral interrupts. Due to the number of peripheral interrupt sources, there are two Peripheral Interrupt Priority registers (IPR1, IPR2). The operation of the priority bits requires that the Interrupt Priority Enable (IPEN) bit be set.

For PIC18FXX39 devices, the Motor Control kernel requires that the Timer2 to PR2 match interrupt be the only high priority interrupt. Failure to do this may result in unpredictable operation of the kernel or the entire microcontroller.

In practical terms, this means:

- Interrupt priority levels are enabled (IPEN = 1);
- High priority interrupts are enabled (INTCON<7> = 1);
- Timer2 interrupt is enabled and set as high priority (PIE1<1> and IPR<1> = 1); and
- all other interrupts are disabled (INTCON or PIR bits = 0), or set as low priority (IPR bits = 0).

Note: Configuring the interrupts is automatically done by the API method `void ProMPT_Init(PWMfrequency)`. It is the user's responsibility to make certain that this method is called at the very beginning of the application.

REGISTER 8-8: IPR1: PERIPHERAL INTERRUPT PRIORITY REGISTER 1

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	U-1	R/W-1	R/W-1
PSPIP(1,2)	ADIP(2)	RCIP(2)	TXIP(2)	SSPIP(2)	—	TMR2IP(3)	TMR1IP(2)
bit 7							bit 0

bit 7 **PSPIP(1,2):** Parallel Slave Port Read/Write Interrupt Priority bit
1 = High priority
0 = Low priority

bit 6 **ADIP(2)**: A/D Converter Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **RCIP(2)**: USART Receive Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **TXIP(2)**: USART Transmit Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **SSPIP(2)**: Master Synchronous Serial Port Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **Unimplemented:** Read as '1'

bit 1 **TMR2IP(3)**: TMR2 to PR2 Match Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **TMR1IP(2)**: TMR1 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

Note 1: This bit is reserved on PIC18F2X39 devices.
2: Maintain this bit cleared (= 0).
3: This bit is reserved for use by the ProMPT kernel; always maintain this bit set (= 1).

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

REGISTER 8-9: IPR2: PERIPHERAL INTERRUPT PRIORITY REGISTER 2

U-0	U-0	U-0	R/W-1	R/W-1	R/W-1	R/W-1	U-1
—	—	—	EEIP(1)	BCLIP(1)	LVDIP(1)	TMR3IP(1)	—
bit 7							bit 0

bit 7-5	**Unimplemented:** Read as '0'
bit 4	**EEIP(1)**: Data EEPROM/FLASH Write Operation Interrupt Priority bit `1` = High priority `0` = Low priority
bit 3	**BCLIP(1)**: Bus Collision Interrupt Priority bit `1` = High priority `0` = Low priority
bit 2	**LVDIP(1)**: Low Voltage Detect Interrupt Priority bit `1` = High priority `0` = Low priority
bit 1	**TMR3IP(1)**: TMR3 Overflow Interrupt Priority bit `1` = High priority `0` = Low priority
bit 0	**Unimplemented:** Read as ‘1’

Note 1: Maintain this bit cleared (= 0).

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

8.5 RCON Register

The RCON register contains the bit which is used to enable prioritized interrupts (IPEN). For PIC18FXX39 devices, the IPEN bit must always be set (= 1) for the ProMPT kernel to function correctly. Refer to page 69 for a more detailed discussion on interrupt priorities.

REGISTER 8-10: RCON REGISTER

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

bit 7 **IPEN[(1)]:** Interrupt Priority Enable bit
1 = Enable priority levels on interrupts
0 = Disable priority levels on interrupts (not used)

bit 6-5 **Unimplemented:** Read as '0'

bit 4 **$\overline{\text{RI}}$:** RESET Instruction Flag bit
For details of bit operation, see Register 4-3

bit 3 **$\overline{\text{TO}}$:** Watchdog Time-out Flag bit
For details of bit operation, see Register 4-3

bit 2 **$\overline{\text{PD}}$:** Power-down Detection Flag bit
For details of bit operation, see Register 4-3

bit 1 **$\overline{\text{POR}}$:** Power-on Reset Status bit
For details of bit operation, see Register 4-3

bit 0 **$\overline{\text{BOR}}$:** Brown-out Reset Status bit
For details of bit operation, see Register 4-3

Note 1: Maintain this bit set (= 1).

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

8.6 INT0 Interrupt

External interrupts on the RB0/INT0, RB1/INT1 and RB2/INT2 pins are edge triggered: either rising, if the corresponding INTEDGx bit is set in the INTCON2 register, or falling, if the INTEDGx bit is clear. When a valid edge appears on the RBx/INTx pin, the corresponding flag bit INTxF is set. This interrupt can be disabled by clearing the corresponding enable bit INTxE. Flag bit INTxF must be cleared in software in the Interrupt Service Routine before re-enabling the interrupt. All external interrupts (INT0, INT1 and INT2) can wake-up the processor from SLEEP, if bit INTxE was set prior to going into SLEEP. If the global interrupt enable bit GIE is set, the processor will branch to the interrupt vector following wake-up.

The INT0 interrupt is always configured as a high priority interrupt, and cannot be reconfigured. Interrupt priority for INT1 and INT2 is determined by the value contained in the interrupt priority bits, INT1IP (INTCON3<6>) and INT2IP (INTCON3<7>).

Because it is always configured as a high priority interrupt, INT0 cannot be used in conjunction with the ProMPT kernel; it must always be disabled (INTCON<4> = 0). Failure to do this may result in erratic operation of the motor control.

8.7 TMR0 Interrupt

In 8-bit mode (which is the default), an overflow in the TMR0 register (FFh → 00h) will set flag bit TMR0IF. In 16-bit mode, an overflow in the TMR0H:TMR0L register pair (FFFFh → 0000h) will set flag bit TMR0IF. The interrupt can be enabled or disabled by setting or clearing enable bit TMR0IE (INTCON<5>). Interrupt priority for Timer0 is determined by the value contained in the interrupt priority bit TMR0IP (INTCON2<2>). See Section 10.0 for further details on the Timer0 module.

8.8 PORTB Interrupt-on-Change

An input change on PORTB<7:4> sets flag bit RBIF (INTCON<0>). The interrupt can be enabled or disabled by setting or clearing the enable bit RBIE (INTCON<3>). Interrupt priority for PORTB interrupt-on-change is determined by the value contained in the interrupt priority bit RBIP (INTCON2<0>).

8.9 Context Saving During Interrupts

During an interrupt, the return PC value is saved on the stack. Additionally, the WREG, STATUS and BSR registers are saved on the fast return stack. If a fast return from interrupt is not used (see Section 4.3), the user may need to save the WREG, STATUS and BSR registers in software. Depending on the user's application, other registers may also need to be saved. Example 8-1 saves and restores the WREG, STATUS and BSR registers during an Interrupt Service Routine.

EXAMPLE 8-1: SAVING STATUS, WREG AND BSR REGISTERS IN RAM

```
MOVWF   W_TEMP                          ; W_TEMP is in virtual bank
MOVFF   STATUS, STATUS_TEMP             ; STATUS_TEMP located anywhere
MOVFF   BSR,    BSR_TEMP                ; BSR located anywhere
;
; USER ISR CODE
;
MOVFF   BSR_TEMP,  BSR                  ; Restore BSR
MOVF    W_TEMP,    W                    ; Restore WREG
MOVFF   STATUS_TEMP,STATUS              ; Restore STATUS
```

NOTES:

9.0 I/O PORTS

Depending on the device selected, there are either three or five ports available. Some pins of the I/O ports are multiplexed with an alternate function from the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

Each port has three registers for its operation. These registers are:

- TRIS register (data direction register)
- PORT register (reads the levels on the pins of the device)
- LAT register (output latch)

The data latch (LAT register) is useful for read-modify-write operations on the value that the I/O pins are driving.

9.1 PORTA, TRISA and LATA Registers

PORTA is a 7-bit wide, bi-directional port. The corresponding Data Direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a High Impedance mode). Clearing a TRISA bit (= 0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin).

Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch.

The Data Latch register (LATA) is also memory mapped. Read-modify-write operations on the LATA register reads and writes the latched output value for PORTA.

The RA4 pin is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin. The RA4/T0CKI 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.

The other PORTA pins are multiplexed with analog inputs and the analog VREF+ and VREF- inputs. The operation of each pin is selected by clearing/setting the control bits in the ADCON1 register (A/D Control Register1).

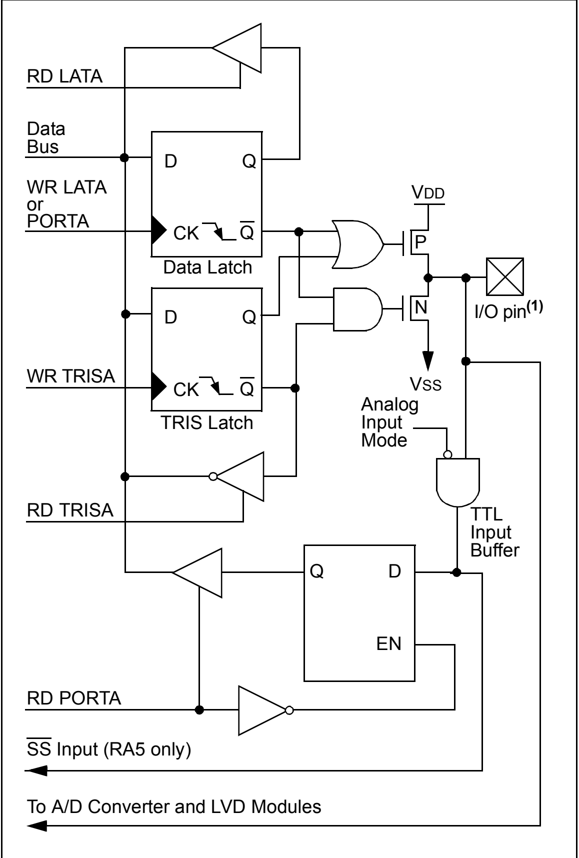
Note: On a Power-on Reset, RA5 and RA3:RA0 are configured as analog inputs and read as '0'. RA6 and RA4 are configured as digital inputs.

The TRISA register controls the direction of the RA pins, even when they are being used as analog inputs. The user must ensure the bits in the TRISA register are maintained set when using them as analog inputs.

EXAMPLE 9-1: INITIALIZING PORTA

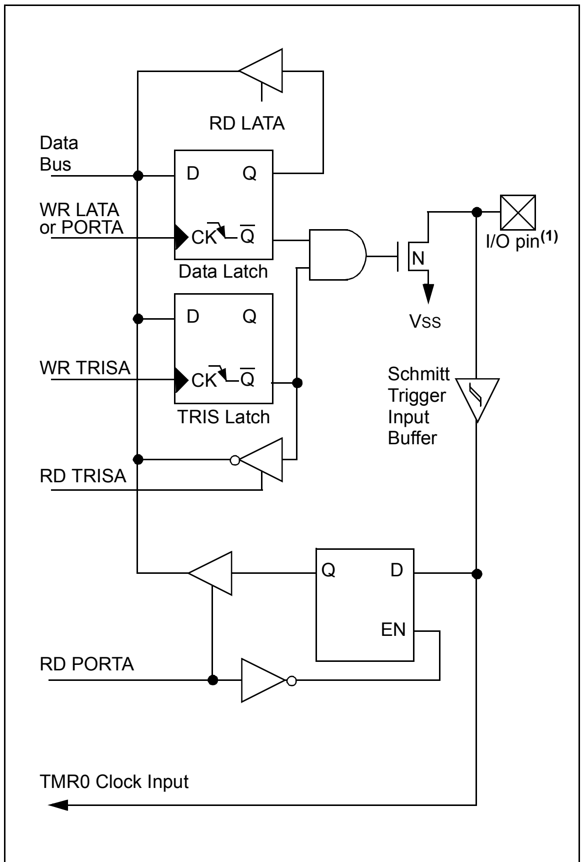
```
CLRF PORTA      ; Initialize PORTA by
                ; clearing output
                ; data latches
CLRF LATA       ; Alternate method
                ; to clear output
                ; data latches
MOVLW 0x07      ; Configure A/D
MOVWF ADCON1    ; for digital inputs
MOVLW 0xCF      ; Value used to
                ; initialize data
                ; direction
MOVWF TRISA     ; Set RA<3:0> as inputs
                ; RA<5:4> as outputs
```

FIGURE 9-1: BLOCK DIAGRAM OF RA3:RA0 AND RA5 PINS



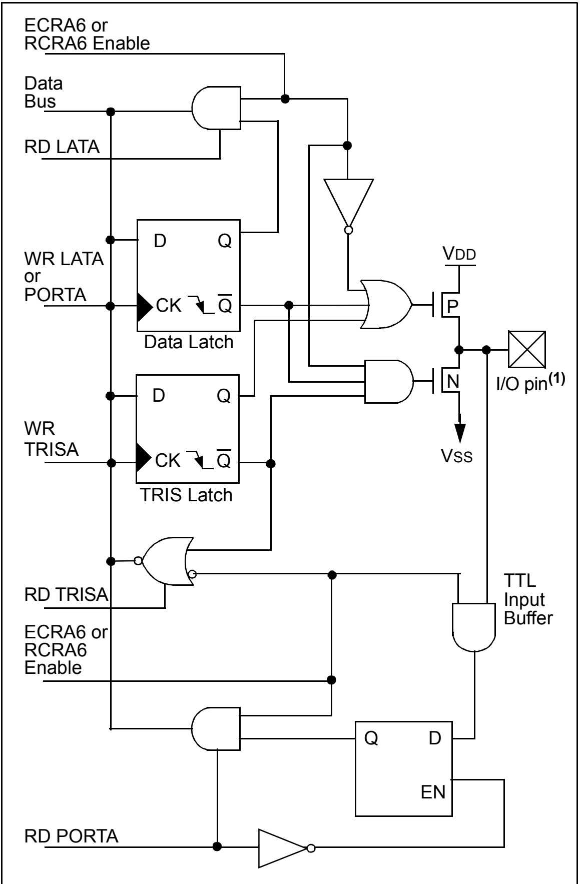
Note 1: I/O pins have protection diodes to VDD and VSS.

FIGURE 9-2: BLOCK DIAGRAM OF RA4/T0CKI PIN



Note 1: I/O pin has protection diode to VSS only.

FIGURE 9-3: BLOCK DIAGRAM OF RA6 PIN



Note 1: I/O pins have protection diodes to VDD and VSS.

TABLE 9-1: PORTA FUNCTIONS

Name	Bit#	Buffer	Function
RA0/AN0	bit0	TTL	Input/output or analog input.
RA1/AN1	bit1	TTL	Input/output or analog input.
RA2/AN2/VREF-	bit2	TTL	Input/output or analog input or VREF-.
RA3/AN3/VREF+	bit3	TTL	Input/output or analog input or VREF+.
RA4/T0CKI	bit4	ST	Input/output or external clock input for Timer0. Output is open drain type.
RA5/$\overline{\text{SS}}$/AN4/LVDIN	bit5	TTL	Input/output or slave select input for synchronous serial port or analog input, or low voltage detect input.
OSC2/CLKO/RA6	bit6	TTL	OSC2 or clock output or I/O pin.

Legend: TTL = TTL input, ST = Schmitt Trigger input

TABLE 9-2: 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
PORTA	—	RA6	RA5	RA4	RA3	RA2	RA1	RA0	`-x0x 0000`	`-u0u 0000`
LATA	—	LATA Data Output Register							`-xxx xxxx`	`-uuu uuuu`
TRISA	—	PORTA Data Direction Register							`-111 1111`	`-111 1111`
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.

9.2 PORTB, TRISB and LATB Registers

PORTB is an 8-bit wide, bi-directional port. The corresponding Data Direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input (i.e., put the corresponding output driver in a High Impedance mode). Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output (i.e., put the contents of the output latch on the selected pin).

The Data Latch register (LATB) is also memory mapped. Read-modify-write operations on the LATB register reads and writes the latched output value for PORTB.

EXAMPLE 9-2: INITIALIZING PORTB

```
CLRF   PORTB   ; Initialize PORTB by
               ; clearing output
               ; data latches
CLRF   LATB    ; Alternate method
               ; to clear output
               ; data latches
MOVLW 0xCF     ; Value used to
               ; initialize data
               ; direction
MOVWF TRISB    ; Set RB<3:0> as inputs
               ; RB<5:4> as outputs
               ; RB<7:6> as inputs
```

Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit $\overline{\text{RBPU}}$ (INTCON2<7>). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

Note: On a Power-on Reset, these pins are configured as digital inputs.

Four of the PORTB 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 “mismatch” outputs of RB7:RB4 are OR’ed together to generate the RB Port Change Interrupt with flag bit, RBIF (INTCON<0>).

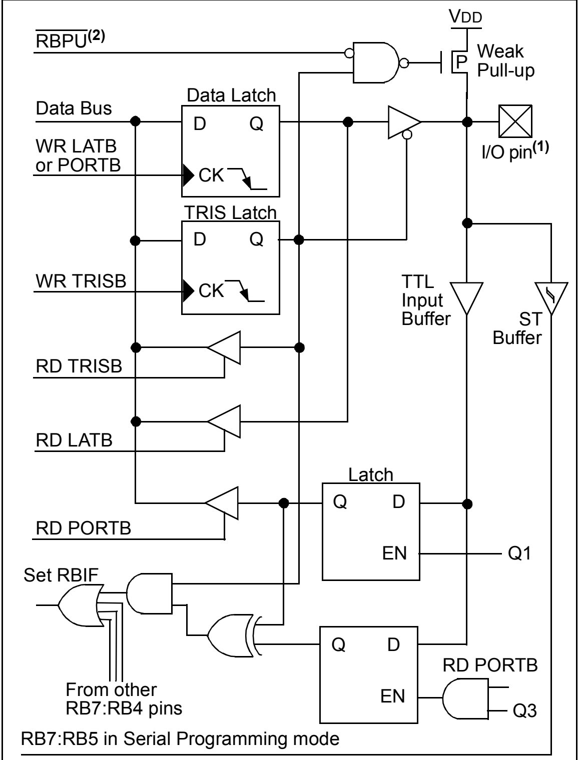
This interrupt can wake the device from SLEEP. The user, in the Interrupt Service Routine, can clear the interrupt in the following manner:

a) Any read or write of PORTB (except with the `MOVFF` instruction). This will end the mismatch condition.
b) Clear flag bit RBIF.

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.

The interrupt-on-change feature is recommended for wake-up on key depression operation 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 9-4: BLOCK DIAGRAM OF RB7:RB4 PINS



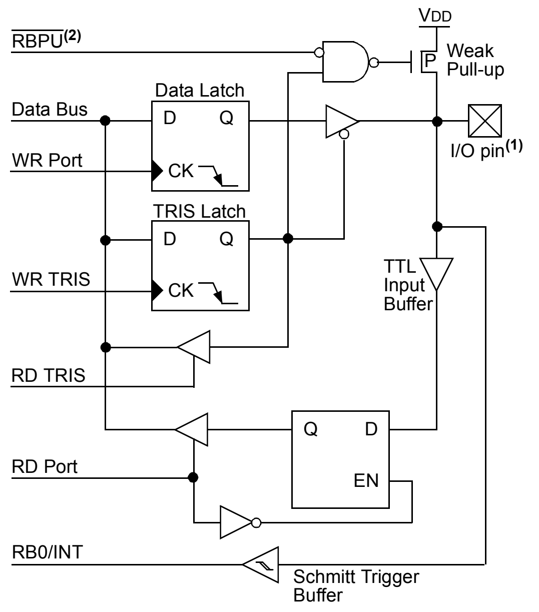
Note 1: I/O pins have diode protection to VDD and VSS.

2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the $\overline{\text{RBPU}}$ bit (INTCON2<7>).

Note 1: While in Low Voltage ICSP mode, the RB5 pin can no longer be used as a general purpose I/O pin, and should be held low during normal operation to protect against inadvertent ICSP mode entry.

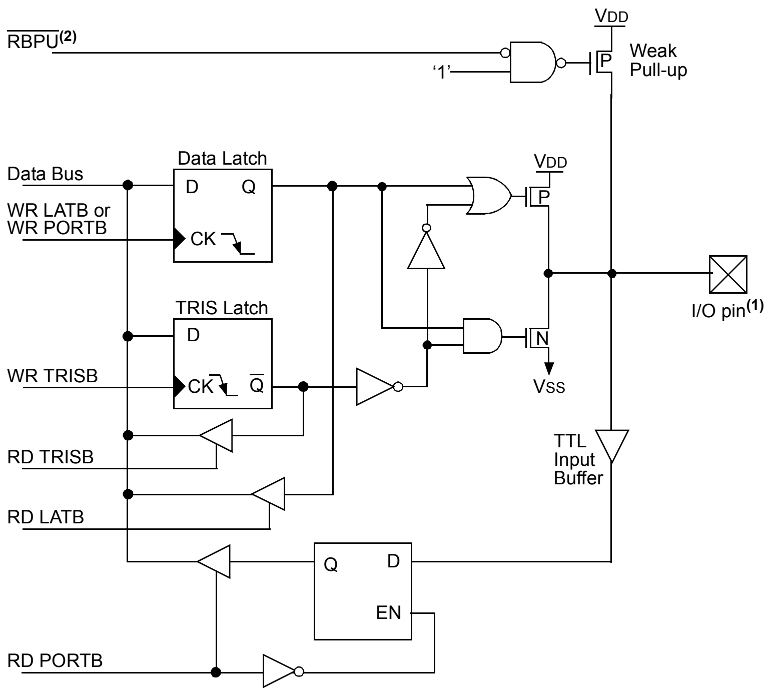
2: When using Low Voltage ICSP programming (LVP), the pull-up on RB5 becomes disabled. If TRISB bit 5 is cleared, thereby setting RB5 as an output, LATB bit 5 must also be cleared for proper operation.

FIGURE 9-5: BLOCK DIAGRAM OF RB2:RB0 PINS



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

FIGURE 9-6: BLOCK DIAGRAM OF RB3 PIN



Note 1: I/O pin has diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate DDR bit(s) and clear the RBPU bit (INTCON2<7>).

TABLE 9-3: PORTB FUNCTIONS

Name	Bit#	Buffer	Function
RB0/INT0	bit0	TTL/ST(1)	Input/output pin or external interrupt input0. Internal software programmable weak pull-up.
RB1/INT1	bit1	TTL/ST(1)	Input/output pin or external interrupt input1. Internal software programmable weak pull-up.
RB2/INT2	bit2	TTL/ST(1)	Input/output pin or external interrupt input2. Internal software programmable weak pull-up.
RB3	bit3	TTL	Input/output pin. Internal software programmable weak pull-up.
RB4	bit4	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB5/PGM(4)	bit5	TTL/ST(2)	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Low voltage ICSP enable pin.
RB6/PGC	bit6	TTL/ST(2)	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up. Serial programming clock.
RB7/PGD	bit7	TTL/ST(2)	Input/output 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.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: A device configuration bit selects which I/O pin the CCP2 pin is multiplexed on.
4: Low Voltage ICSP Programming (LVP) is enabled by default, which disables the RB5 I/O function. LVP must be disabled to enable RB5 as an I/O pin and allow maximum compatibility to the other 28-pin and 40-pin mid-range devices.

TABLE 9-4: 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
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	`xxxx xxxx`	`uuuu uuuu`
LATB	LATB Data Output Register								`xxxx xxxx`	`uuuu uuuu`
TRISB	PORTB Data Direction Register								`1111 1111`	`1111 1111`
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	`0000 000x`	`0000 000u`
INTCON2	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. Shaded cells are not used by PORTB.

9.3 PORTC, TRISC and LATC Registers

PORTC is a 6-bit wide, bi-directional port. The corresponding Data Direction register is TRISC. Setting a TRISC bit (= 1) will make the corresponding PORTC pin an input (i.e., put the corresponding output driver in a High Impedance mode). Clearing a TRISC bit (= 0) will make the corresponding PORTC pin an output (i.e., put the contents of the output latch on the selected pin).

The Data Latch register (LATC) is also memory mapped. Read-modify-write operations on the LATC register reads and writes the latched output value for PORTC.

PORTC is multiplexed with the serial communication functions (Table 9-5). 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. The user should refer to the corresponding peripheral section for the correct TRIS bit settings.

Note: On a Power-on Reset, these pins are configured as digital inputs.

The pin override value is not loaded into the TRIS register. This allows read-modify-write of the TRIS register, without concern due to peripheral overrides.

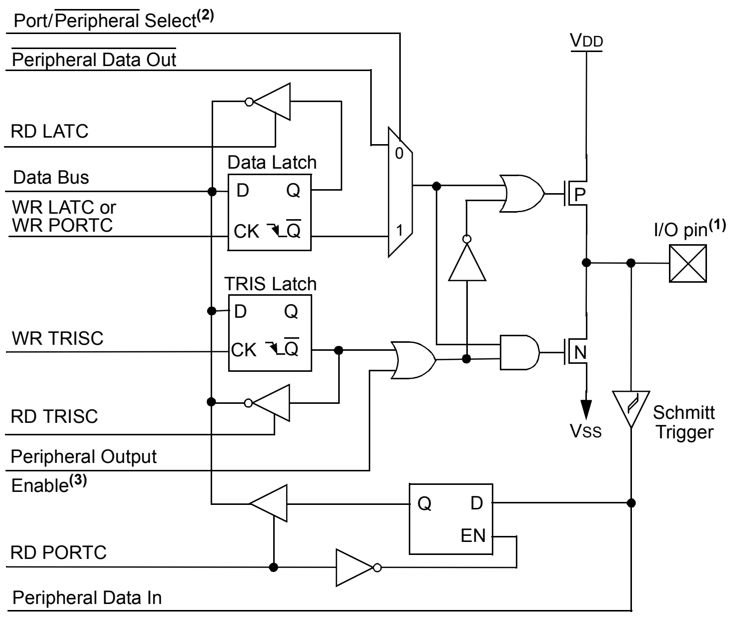
EXAMPLE 9-3: INITIALIZING PORTC

```
CLRF   PORTC    ; Initialize PORTC by
                ; clearing output
                ; data latches
CLRF   LATC     ; Alternate method
                ; to clear output
                ; data latches
MOVLW 0xC9      ; Value used to
                ; initialize data
                ; direction
MOVWF TRISC     ; Set RC<3>,RC<0> as inputs,
                ; RC<5:4> as outputs, and
                ; RC<7:6> as inputs
```

PIC18FXX39 devices differ from other PIC18 microcontrollers in allocation of PORTC pins. For most PIC18 devices, PORTC is an 8-bit-wide port. For the PIC18FXX39 family, two of the PORTC pins (RC1 and RC2) are re-allocated as PWM output only pins for use with the Motor Control kernel. To maintain pinout compatibility with other PICmicro® devices, the remaining PORTC pins are assigned in a manner consistent with other PIC18 devices. For this reason, PORTC has pins RC0 and RC3 through RC7, but not RC1 and RC2.

To maintain compatibility with PIC18FXX2 devices, the individual port and corresponding latch and direction bits for RC1 and RC2 are present in the appropriate registers, but are not available to the user. To avoid erratic device operation, the values of these bits should not be modified.

FIGURE 9-7: PORTC BLOCK DIAGRAM (PERIPHERAL OUTPUT OVERRIDE)



Note 1: I/O pins have diode protection to VDD and VSS.

2: Port/$\overline{\text{Peripheral}}$ Select signal selects between port data (input) and peripheral output.

3: Peripheral Output Enable is only active if Peripheral Select is active.

TABLE 9-5: PORTC FUNCTIONS

Name	Bit#	Buffer Type	Function
RC0/T13CKI	bit0	ST	Input/output port pin or Timer1 oscillator output/Timer1 clock input.
RC3/SCK/SCL	bit3	ST	RC3 can also be the synchronous serial clock for both SPI and I^2C modes.
RC4/SDI/SDA	bit4	ST	RC4 can also be the SPI Data In (SPI mode) or Data I/O (I^2C mode).
RC5/SDO	bit5	ST	Input/output port pin or Synchronous Serial Port data output.
RC6/TX/CK	bit6	ST	Input/output port pin, Addressable USART Asynchronous Transmit, or Addressable USART Synchronous Clock.
RC7/RX/DT	bit7	ST	Input/output port pin, Addressable USART Asynchronous Receive, or Addressable USART Synchronous Data.

Legend: ST = Schmitt Trigger input

TABLE 9-6: 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
PORTC	RC7	RC6	RC5	RC4	RC3	*	*	RC0	xxxx xxxx	uuuu uuuu
LATC	LATC7	LATC6	LATC5	LATC4	LATC3	*	*	LATC0	xxxx xxxx	uuuu uuuu
TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	*	*	TRISC0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTC.
* Reserved bits; do not modify.

9.4 PORTD, TRISD and LATD Registers

This section is applicable only to the PIC18F4X39 devices.

PORTD is an 8-bit wide, bi-directional port. The corresponding Data Direction register is TRISD. Setting a TRISD bit (= 1) will make the corresponding PORTD pin an input (i.e., put the corresponding output driver in a High Impedance mode). Clearing a TRISD bit (= 0) will make the corresponding PORTD pin an output (i.e., put the contents of the output latch on the selected pin).

The Data Latch register (LATD) is also memory mapped. Read-modify-write operations on the LATD register reads and writes the latched output value for PORTD.

PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.

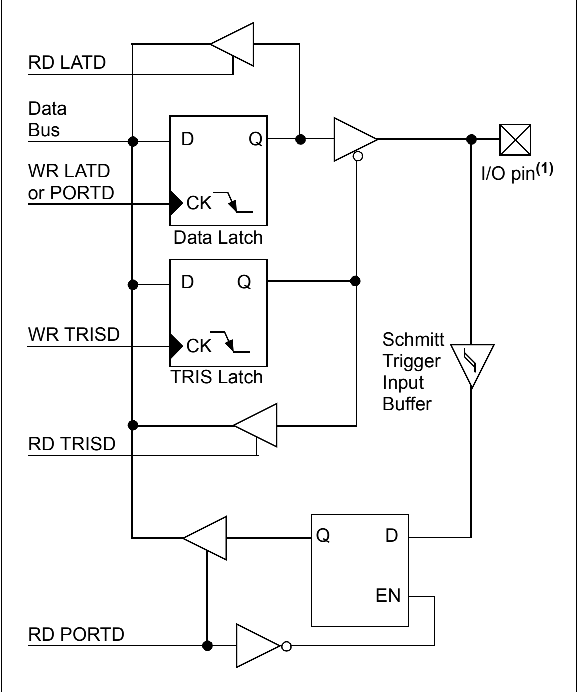
Note: On a Power-on Reset, these pins are configured as digital inputs.

PORTD can be configured as an 8-bit wide microprocessor port (parallel slave port) by setting control bit PSPMODE (TRISE<4>). In this mode, the input buffers are TTL. See Section 9.6 for additional information on the Parallel Slave Port (PSP).

EXAMPLE 9-4: INITIALIZING PORTD

```
CLRF   PORTD  ; Initialize PORTD by
              ; clearing output
              ; data latches
CLRF   LATD   ; Alternate method
              ; to clear output
              ; data latches
MOVLW 0xCF    ; Value used to
              ; initialize data
              ; direction
MOVWF TRISD   ; Set RD<3:0> as inputs
              ; RD<5:4> as outputs
              ; RD<7:6> as inputs
```

FIGURE 9-8: PORTD BLOCK DIAGRAM IN I/O PORT MODE



Note 1: I/O pins have diode protection to VDD and VSS.

TABLE 9-7: PORTD FUNCTIONS

Name	Bit#	Buffer Type	Function
RD0/PSP0	bit0	ST/TTL(1)	Input/output port pin or parallel slave port bit0.
RD1/PSP1	bit1	ST/TTL(1)	Input/output port pin or parallel slave port bit1.
RD2/PSP2	bit2	ST/TTL(1)	Input/output port pin or parallel slave port bit2.
RD3/PSP3	bit3	ST/TTL(1)	Input/output port pin or parallel slave port bit3.
RD4/PSP4	bit4	ST/TTL(1)	Input/output port pin or parallel slave port bit4.
RD5/PSP5	bit5	ST/TTL(1)	Input/output port pin or parallel slave port bit5.
RD6/PSP6	bit6	ST/TTL(1)	Input/output port pin or parallel slave port bit6.
RD7/PSP7	bit7	ST/TTL(1)	Input/output port pin or parallel slave port bit7.

Legend: ST = Schmitt Trigger input, TTL = TTL input

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

TABLE 9-8: 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
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	`xxxx xxxx`	`uuuu uuuu`
LATD	LATD Data Output Register								`xxxx xxxx`	`uuuu uuuu`
TRISD	PORTD Data Direction Register								`1111 1111`	`1111 1111`
TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			`0000 -111`	`0000 -111`

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTD.

9.5 PORTE, TRISE and LATE Registers

This section is only applicable to the PIC18F4X39 devices.

PORTE is a 3-bit wide, bi-directional port. The corresponding Data Direction register is TRISE. Setting a TRISE bit (= 1) will make the corresponding PORTE pin an input (i.e., put the corresponding output driver in a High Impedance mode). Clearing a TRISE bit (= 0) will make the corresponding PORTE pin an output (i.e., put the contents of the output latch on the selected pin).

The Data Latch register (LATE) is also memory mapped. Read-modify-write operations on the LATE register reads and writes the latched output value for PORTE.

PORTE has three pins (RE0/AN5/$\overline{\text{RD}}$, RE1/AN6/$\overline{\text{WR}}$ and RE2/AN7/$\overline{\text{CS}}$) which are individually configurable as inputs or outputs. These pins have Schmitt Trigger input buffers.

Register 9-1 shows the TRISE register, which also controls the parallel slave port operation.

PORTE pins are multiplexed with analog inputs. When selected as an analog input, these pins will read as '0's.

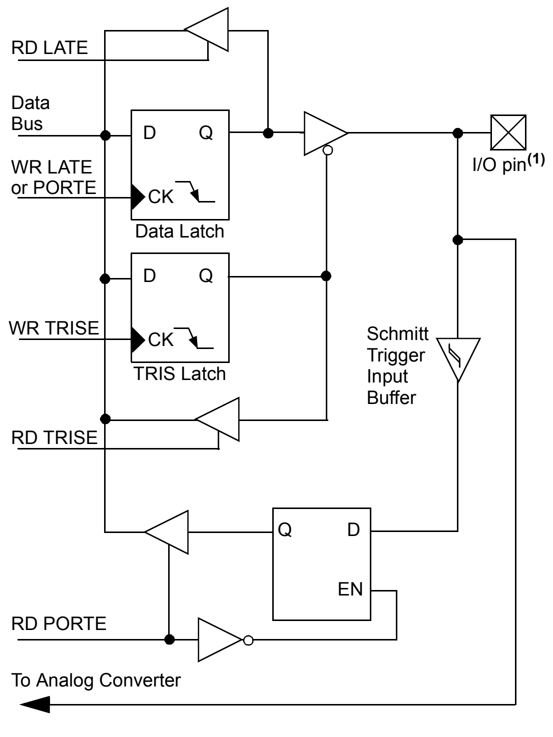
TRISE controls the direction of the RE pins, even when they are being used as analog inputs. The user must make sure to keep the pins configured as inputs when using them as analog inputs.

Note:	On a Power-on Reset, these pins are configured as analog inputs.

EXAMPLE 9-5: INITIALIZING PORTE

```
CLRF    PORTE    ; Initialize PORTE by
                 ; clearing output
                 ; data latches
CLRF    LATE     ; Alternate method
                 ; to clear output
                 ; data latches
MOVLW   0x07     ; Configure A/D
MOVWF   ADCON1   ; for digital inputs
MOVLW   0x05     ; Value used to
                 ; initialize data
                 ; direction
MOVWF   TRISE    ; Set RE<0> as inputs
                 ; RE<1> as outputs
                 ; RE<2> as inputs
```

FIGURE 9-9: PORTE BLOCK DIAGRAM IN I/O PORT MODE



Note 1: I/O pins have diode protection to VDD and VSS.

REGISTER 9-1: 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	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

TABLE 9-9: PORTE FUNCTIONS

Name	Bit#	Buffer Type	Function
RE0/AN5/$\overline{RD}$	bit0	ST/TTL(1)	Input/output port pin or analog input or read control input in Parallel Slave Port mode For $\overline{RD}$ (PSP mode): `1` = Not a read operation `0` = Read operation. Reads PORTD register (if chip selected).
RE1/AN6/$\overline{WR}$	bit1	ST/TTL(1)	Input/output port pin or analog input or write control input in Parallel Slave Port mode For $\overline{WR}$ (PSP mode): `1` = Not a write operation `0` = Write operation. Writes PORTD register (if chip selected).
RE2/AN7/$\overline{CS}$	bit2	ST/TTL(1)	Input/output port pin or analog input or chip select control input in Parallel Slave Port mode For $\overline{CS}$ (PSP mode): `1` = Device is not selected `0` = Device is selected

Legend: ST = Schmitt Trigger input, TTL = TTL input

Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

TABLE 9-10: 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
PORTE	—	—	—	—	—	RE2	RE1	RE0	`---- -000`	`---- -000`
LATE	—	—	—	—	—	LATE Data Output Register			`---- -xxx`	`---- -uuu`
TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			`0000 -111`	`0000 -111`
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	`00-- 0000`	`00-- 0000`

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTE.

9.6 Parallel Slave Port

The Parallel Slave Port is implemented on the 40-pin devices only (PIC18F4X39).

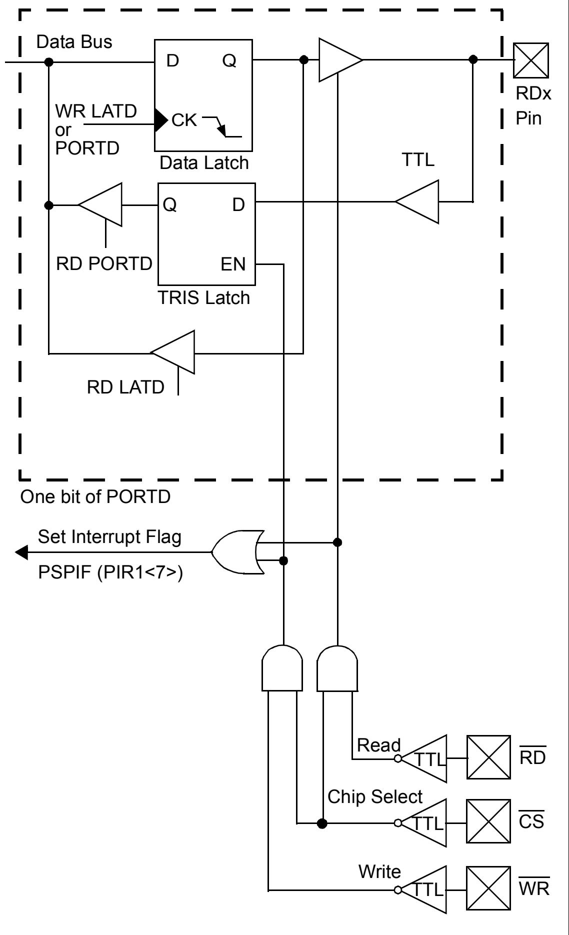
PORTD also operates as an 8-bit wide Parallel Slave Port, or microprocessor port, when control bit PSPMODE (TRISE<4>) is set. It is asynchronously readable and writable by the external world through $\overline{RD}$ control input pin, RE0/AN5/$\overline{RD}$ and $\overline{WR}$ control input pin, RE1/AN6/$\overline{WR}$.

The PSP can directly interface to an 8-bit microprocessor data bus. The external microprocessor can read or write the PORTD latch as an 8-bit latch. Setting bit PSPMODE enables port pin RE0/AN5/$\overline{RD}$ to be the $\overline{RD}$ input, RE1/AN6/$\overline{WR}$ to be the $\overline{WR}$ input and RE2/AN7/$\overline{CS}$ to be the $\overline{CS}$ (chip select) input. For this functionality, the corresponding data direction bits of the TRISE register (TRISE<2:0>) must be configured as inputs (set). The A/D port configuration bits, PCFG2:PCFG0 (ADCON1<2:0>), must be set, which will configure pins RE2:RE0 as digital I/O.

A write to the PSP occurs when both the $\overline{CS}$ and $\overline{WR}$ lines are first detected low. A read from the PSP occurs when both the $\overline{CS}$ and $\overline{RD}$ lines are first detected low.

The PORTE I/O pins become control inputs for the microprocessor port when bit PSPMODE (TRISE<4>) is set. In this mode, the user must make sure that the TRISE<2:0> bits are set (pins are configured as digital inputs), and the ADCON1 is configured for digital I/O. In this mode, the input buffers are TTL.

FIGURE 9-10: PORTD AND PORTE BLOCK DIAGRAM (PARALLEL SLAVE PORT)



Note: I/O pin has protection diodes to VDD and VSS.

FIGURE 9-11: PARALLEL SLAVE PORT WRITE WAVEFORMS

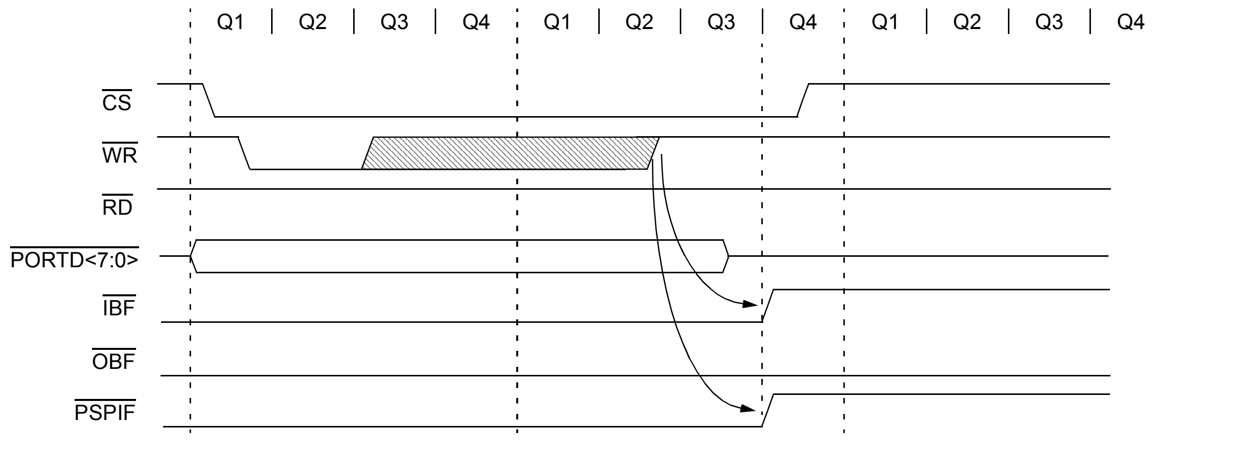


FIGURE 9-12: PARALLEL SLAVE PORT READ WAVEFORMS

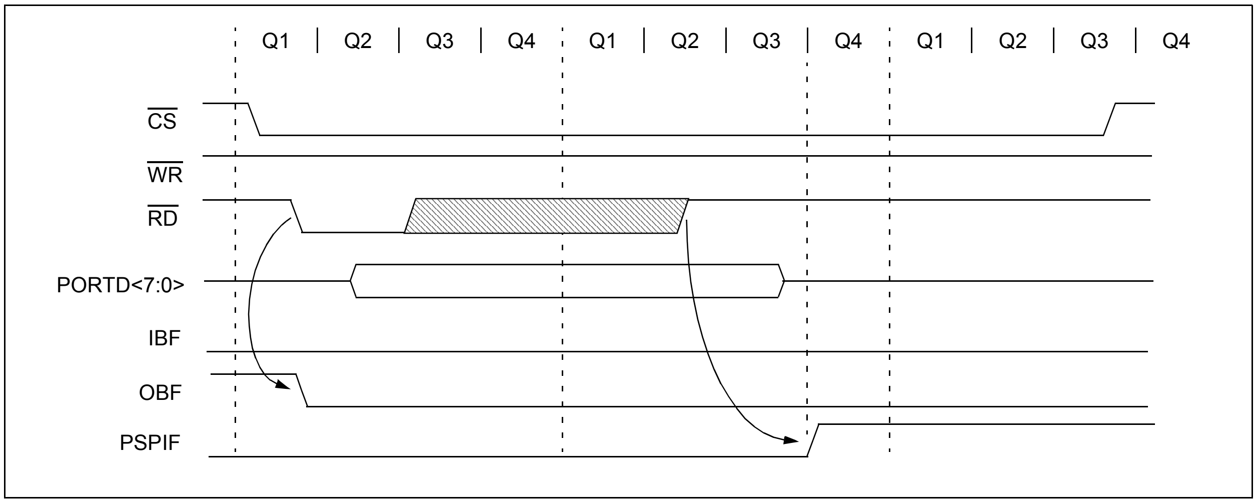


TABLE 9-11: REGISTERS ASSOCIATED WITH PARALLEL SLAVE PORT

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	—	—	—	—	—	RE2	RE1	RE0	`---- -000`	`---- -000`
LATE	—	—	—	—	—	LATE Data Output bits			`---- -xxx`	`---- -uuu`
TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			`0000 -111`	`0000 -111`
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IF	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	`0000 000x`	`0000 000u`
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF	TMR1IF	`0000 0000`	`0000 0000`
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	`0000 0000`	`0000 0000`
IPR1	PSPIP	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	`0000 0000`	`0000 0000`
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	`00-- 0000`	`00-- 0000`

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Parallel Slave Port.

NOTES:

10.0 TIMER0 MODULE

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 in 8-bit mode and FFFFh to 0000h in 16-bit mode
- Edge select for external clock

Figure 10-1 shows a simplified block diagram of the Timer0 module in 8-bit mode and Figure 10-2 shows a simplified block diagram of the Timer0 module in 16-bit mode.

The T0CON register (Register 10-1) is a readable and writable register that controls all the aspects of Timer0, including the prescale selection.

REGISTER 10-1: 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
`0` = Internal instruction cycle clock (CLKO)

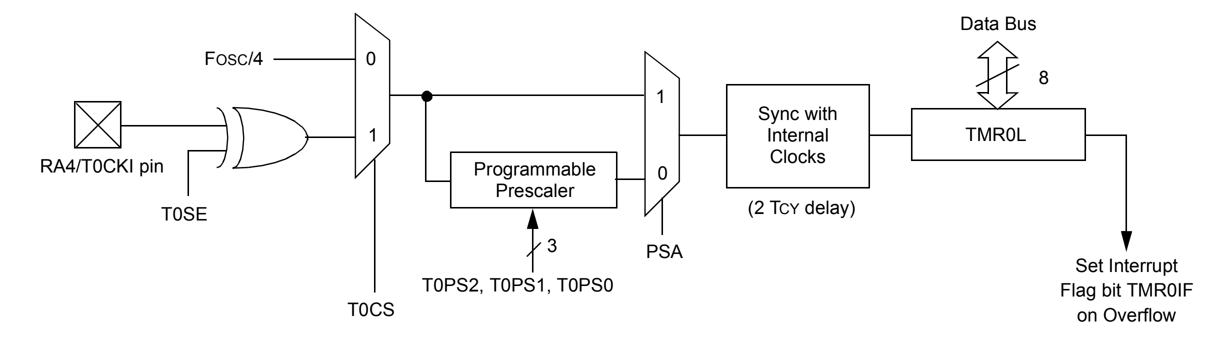
bit 4 **T0SE**: Timer0 Source Edge Select bit
`1` = Increment on high-to-low transition on T0CKI pin
`0` = Increment on low-to-high transition on T0CKI pin

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
`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

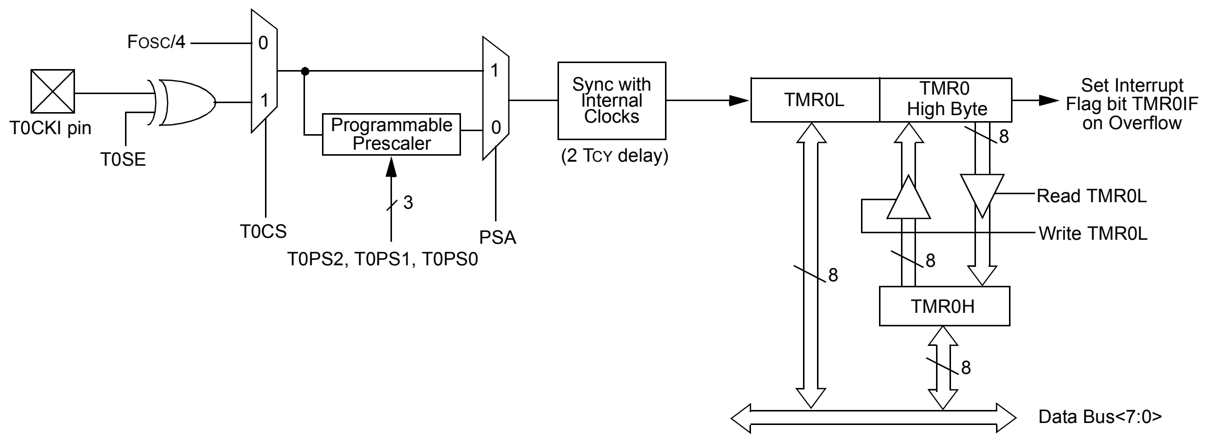
Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
- n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

FIGURE 10-1: TIMER0 BLOCK DIAGRAM IN 8-BIT MODE



Note: Upon RESET, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

FIGURE 10-2: TIMER0 BLOCK DIAGRAM IN 16-BIT MODE



Note: Upon RESET, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

10.1 Timer0 Operation

Timer0 can operate as a timer or as a counter.

Timer mode is selected by clearing the T0CS bit. In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0L 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 TMR0L register.

Counter mode is selected by setting the T0CS bit. In Counter mode, Timer0 will increment, either on every rising or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit (T0SE). Clearing the T0SE bit selects the rising edge. Restrictions on the external clock input are discussed below.

When an external clock input is used for Timer0, it must meet certain requirements. The requirements ensure the external clock can be synchronized with the internal phase clock (TOSC). Also, there is a delay in the actual incrementing of Timer0 after synchronization.

10.2 Prescaler

An 8-bit counter is available as a prescaler for the Timer0 module. The prescaler is not readable or writable.

The PSA and T0PS2:T0PS0 bits determine the prescaler assignment and prescale ratio.

Clearing bit PSA will assign the prescaler to the Timer0 module. When the prescaler is assigned to the Timer0 module, prescale values in power-of-2 increments, from 1:2 through 1:256, are selectable.

When assigned to the Timer0 module, all instructions writing to the TMR0L register (e.g., `CLRF TMR0`, `MOVWF TMR0`, `BSF TMR0`, etc.) will clear the prescaler count.

Note:	Writing to TMR0L when the prescaler is assigned to Timer0 will clear the prescaler count, but will not change the prescaler assignment.

10.2.1 SWITCHING PRESCALER ASSIGNMENT

The prescaler assignment is fully under software control; it can be changed "on-the-fly" during program execution.

10.3 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h in 8-bit mode, or FFFFh to 0000h in 16-bit mode. This overflow sets the TMR0IF bit. The interrupt can be masked by clearing the TMR0IE bit. The TMR0IE bit must be cleared in software by the Timer0 module Interrupt Service Routine before re-enabling this interrupt. The TMR0 interrupt cannot awaken the processor from SLEEP, since the timer is shut-off during SLEEP.

10.4 16-bit Mode Timer Reads and Writes

TMR0H is not the high byte of the timer/counter in 16-bit mode, but is actually a buffered version of the high byte of Timer0 (see Figure 10-2). The high byte of the Timer0 counter/timer is not directly readable nor writable. TMR0H is updated with the contents of the high byte of Timer0 during a read of TMR0L. This provides 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.

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 all 16 bits of Timer0 to be updated at once.

TABLE 10-1: 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 Low Byte Register								xxxx xxxx	uuuu uuuu
TMR0H	Timer0 Module 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							-111 1111	-111 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

NOTES:

11.0 TIMER1 MODULE

The Timer1 module timer/counter has the following features:

- 16-bit timer/counter (two 8-bit registers, TMR1H and TMR1L)
- Readable and writable (both registers)
- Internal or external clock select
- Interrupt-on-overflow from FFFFh to 0000h

Figure 11-1 is a simplified block diagram of the Timer1 module.

Register 11-1 details the Timer1 control register, which sets the Operating mode of the Timer1 module. Timer1 can be enabled or disabled by setting or clearing control bit TMR1ON (T1CON<0>).

REGISTER 11-1: T1CON: TIMER1 CONTROL REGISTER

R/W-0	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0
RD16	—	T1CKPS1	T1CKPS0	—	$\overline{\text{T1SYNC}}$	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 **Unimplemented:** Maintain as '0'

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 RC0/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	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

11.1 Timer1 Operation

Timer1 can operate in one of these modes:

- As a timer
- As a synchronous counter
- As an asynchronous counter

The Operating mode is determined by the clock select bit, TMR1CS (T1CON<1>). When TMR1CS = 0, Timer1 increments every instruction cycle. When TMR1CS = 1, Timer1 increments on every rising edge of the external clock input.

FIGURE 11-1: TIMER1 BLOCK DIAGRAM

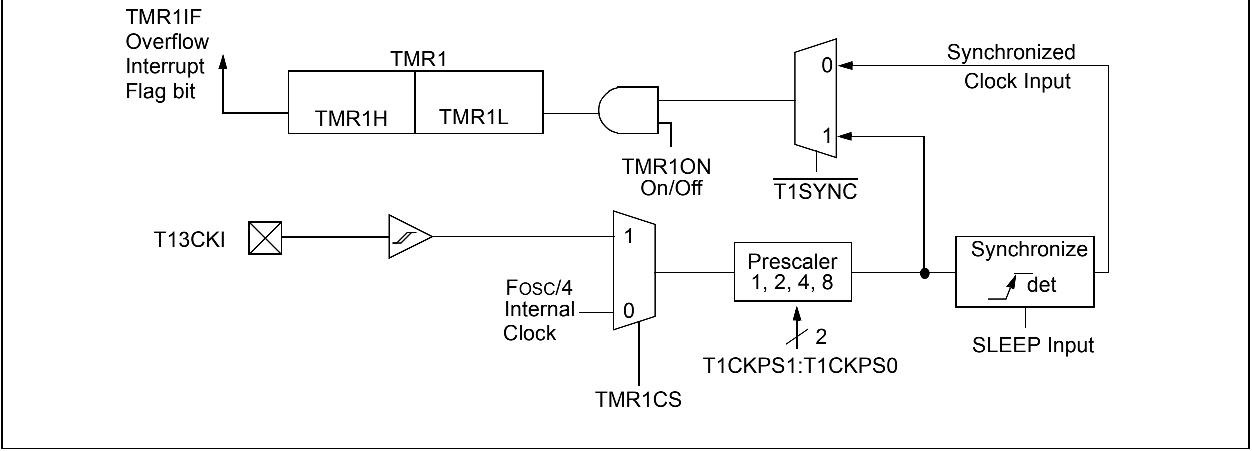
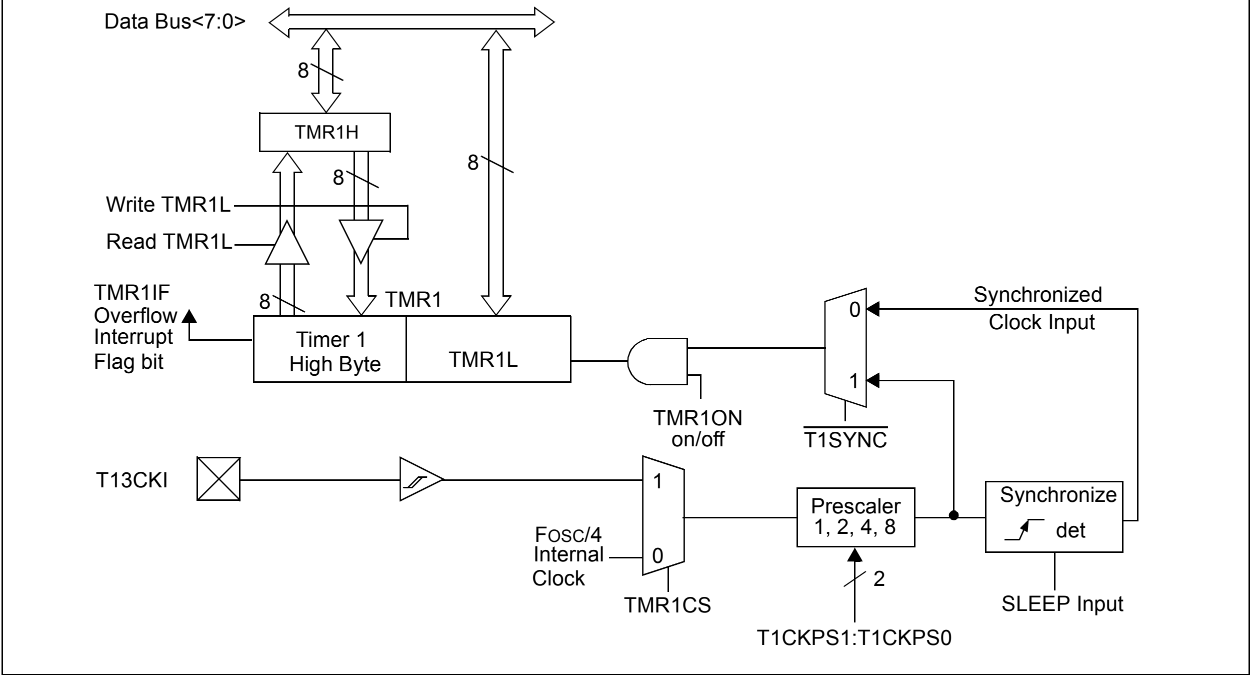


FIGURE 11-2: TIMER1 BLOCK DIAGRAM: 16-BIT READ/WRITE MODE



11.2 Timer1 Interrupt

The TMR1 register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 interrupt, if enabled, is generated on overflow, which is latched in interrupt flag bit TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit, TMR1IE (PIE1<0>).

11.3 Timer1 16-bit Read/Write Mode

Timer1 can be configured for 16-bit reads and writes (see Figure 11-2). When the RD16 control bit (T1CON<7>) 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.

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.

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.

TABLE 11-1: 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/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1	PSPIF[(1)]	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE[(1)]	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP[(1)]	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	0000 0000	0000 0000
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	—	T1SYNC	TMR1CS	TMR1ON	0-00 0000	u-uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.

Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18F2X39 devices; always maintain these bits clear.

NOTES:

12.0 TIMER2 MODULE

The Timer2 module is an 8-bit timer with a selectable 8-bit period. It has the following features:

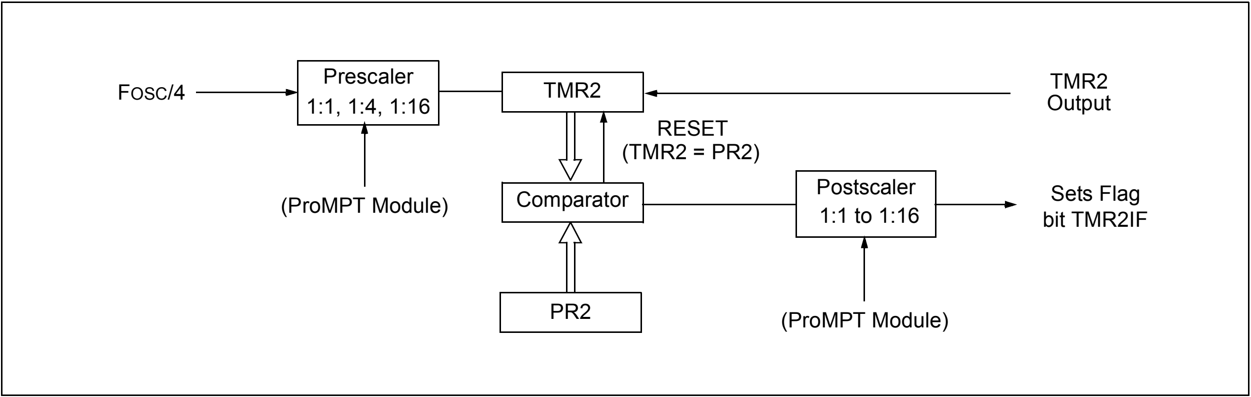
- Input from system clock at FOSC/4 with programmable input prescaler
- Interrupt on timer-to-period match with programmable postscaler

The module has three registers: the TMR2 counter, the PR2 period register, and the T2CON control register. The general operation of Timer2 is shown in Figure 12-1.

Additional information on the use of Timer2 as a time-base is available in Section 15.0 (PWM Modules).

Note:	In PIC18FXX39 devices, Timer2 is used exclusively as a time-base for the PWM modules in motor control applications. As such, it is not available to users as a resource. Although their locations are shown on the device data memory maps, none of the Timer2 registers are directly accessible. Users should not alter the values of these registers.

FIGURE 12-1: TIMER2 BLOCK DIAGRAM



NOTES:

13.0 TIMER3 MODULE

The Timer3 module timer/counter has the following features:

- 16-bit timer/counter (two 8-bit registers: TMR3H and TMR3L)
- Readable and writable (both registers)
- Internal or external clock select
- Interrupt-on-overflow from FFFFh to 0000h

Figure 13-1 is a simplified block diagram of the Timer3 module.

Register 13-1 shows the Timer1 control register, which sets the Operating mode of the Timer1 module.

REGISTER 13-1: 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	—	T3CKPS1	T3CKPS0	—	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON
bit 7							bit 0

bit 7	**RD16:** 16-bit Read/Write Mode Enable bit `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	**Unimplemented:** Maintain as ‘0’
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 T13CKI (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	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

13.1 Timer3 Operation

Timer3 can operate in one of these modes:

- As a timer
- As a synchronous counter
- As an asynchronous counter

The Operating mode is determined by the clock select bit, TMR3CS (T3CON<1>). When TMR3CS = 0, Timer3 increments every instruction cycle. When TMR3CS = 1, Timer3 increments on every rising edge of the Timer1 external clock input.

FIGURE 13-1: TIMER3 BLOCK DIAGRAM

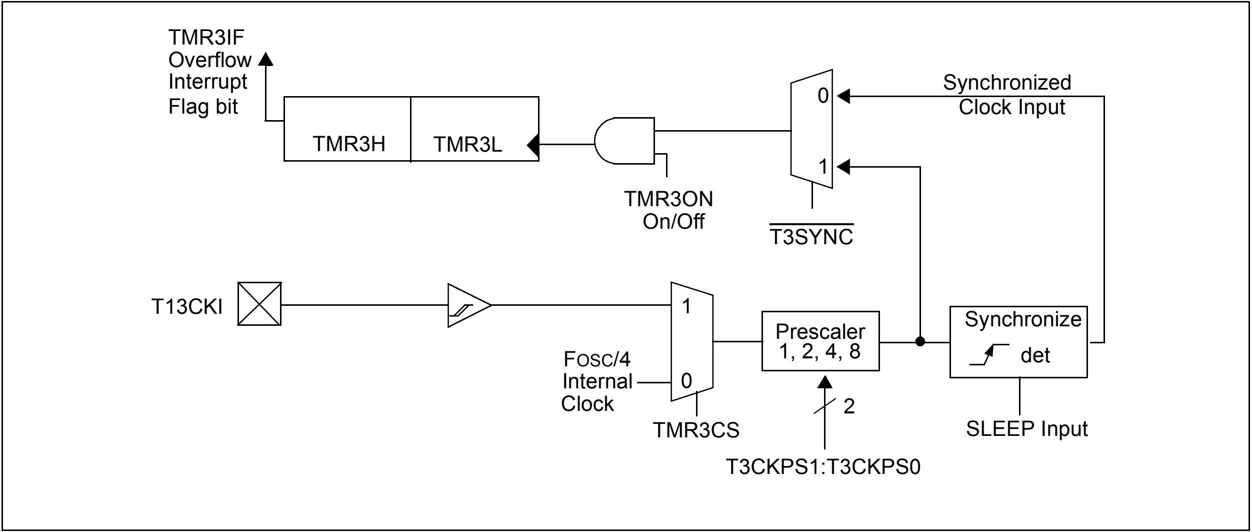
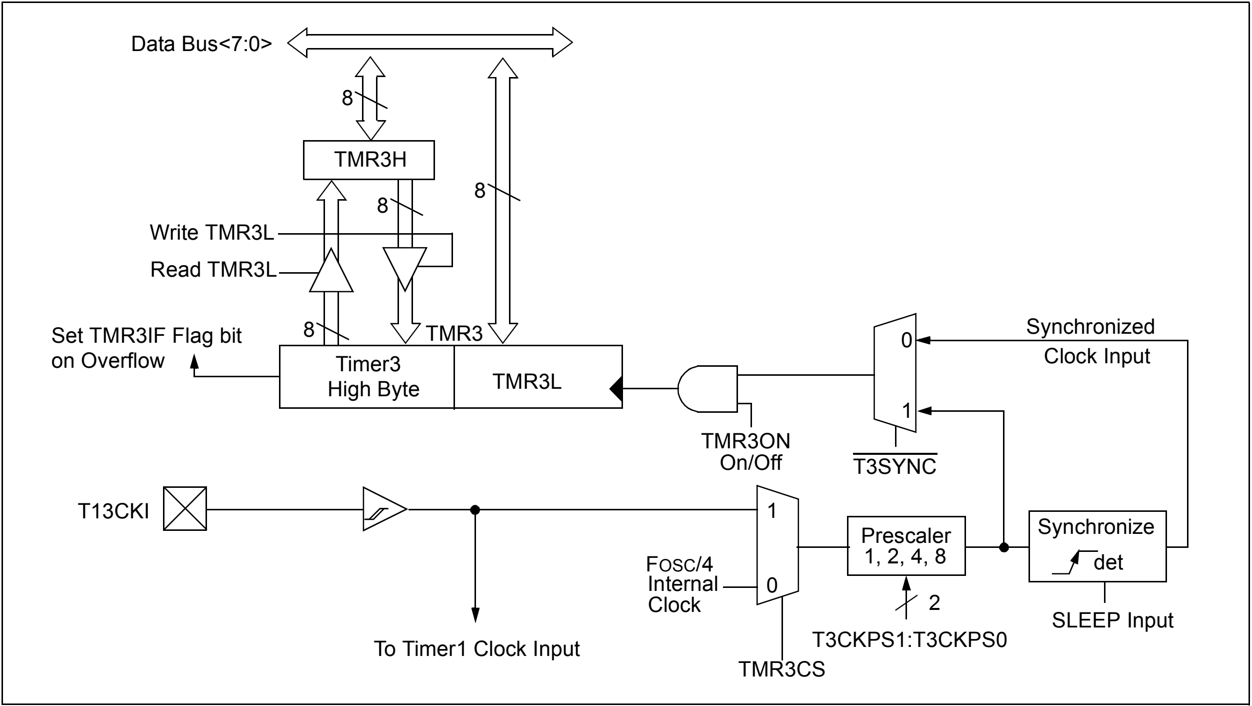


FIGURE 13-2: TIMER3 BLOCK DIAGRAM CONFIGURED IN 16-BIT READ/WRITE MODE



13.2 Timer3 Interrupt

The TMR3 Register pair (TMR3H:TMR3L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR3 Interrupt, if enabled, is generated on overflow, which is latched in interrupt flag bit, TMR3IF (PIR2<1>). This interrupt can be enabled/disabled by setting/clearing TMR3 interrupt enable bit, TMR3IE (PIE2<1>).

TABLE 13-1: 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/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	`0000 000x`	`0000 000u`
PIR2	—	—	—	EEIF	BCLIF	LVDIF	TMR3IF	—	`---0 0000`	`---0 0000`
PIE2	—	—	—	EEIE	BCLIE	LVDIE	TMR3IE	—	`---0 0000`	`---0 0000`
IPR2	—	—	—	EEIP	BCLIP	LVDIP	TMR3IP	—	`---1 1111`	`---1 1111`
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`
T1CON	RD16	—	T1CKPS1	T1CKPS0	—	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	`0-00 0000`	`u-uu uuuu`
T3CON	RD16	—	T3CKPS1	T3CKPS0	—	$\overline{\text{T3SYNC}}$	TMR3CS	TMR3ON	`0000 0000`	`uuuu uuuu`

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.

NOTES:

14.0 SINGLE PHASE INDUCTION MOTOR CONTROL KERNEL

The Motor Control kernel of the PIC18FXX39 family uses Programmable Motor Processor Technology (ProMPT) to control the speed of a single phase induction motor, with variable frequency technology. The controller's two PWM modules are used to synthesize a sine wave current through the motor windings. The kernel provides open loop control for a continuous frequency range of 15 Hz to 127 Hz.

14.1 Theory of Operation

The speed of an induction motor is a function of frequency, slip and the number of poles in the motor. They are related by the equation:

$$Speed = (F \times 120 / P) - Slip$$

where $Speed$ and $Slip$ are in RPM, F is the frequency of the input voltage (in Hertz), and P represents the number of motor poles (for this equation, either 2, 4, 6 or 8).

For the purpose of this discussion, slip is assumed to be constant across the motor's useful operating range. Since the rated speed is based on the number of poles (which is fixed at the time of manufacture), this leaves changing the frequency of the supplied voltage as the only way to vary the motor's speed. When the frequency controlling a motor is reduced, however, its impedance is also reduced, resulting in a higher motor current draw.

It can be shown that the voltage applied to the motor is proportional to both the frequency and the current (Equation 14-1). So to keep the current constant at, or below the Full Load Amp rating, the RMS voltage to the motor must be reduced as the frequency is reduced. By varying the supply voltage and frequency at a constant ratio, the motor's speed can be varied with constant current. Maintaining this constant ratio is the function of the Motor Control kernel.

EQUATION 14-1: KEY RELATIONSHIPS IN SINGLE PHASE MOTORS

$$V \propto \phi \times \omega \quad (1\text{-}1)$$

or:

$$V \propto 2\pi f \phi \quad (1\text{-}2)$$

$$I \propto \phi \propto \frac{V}{f} \quad (1\text{-}3)$$

where: V is applied voltage
I is motor current
ϕ is stator flux
f is input frequency

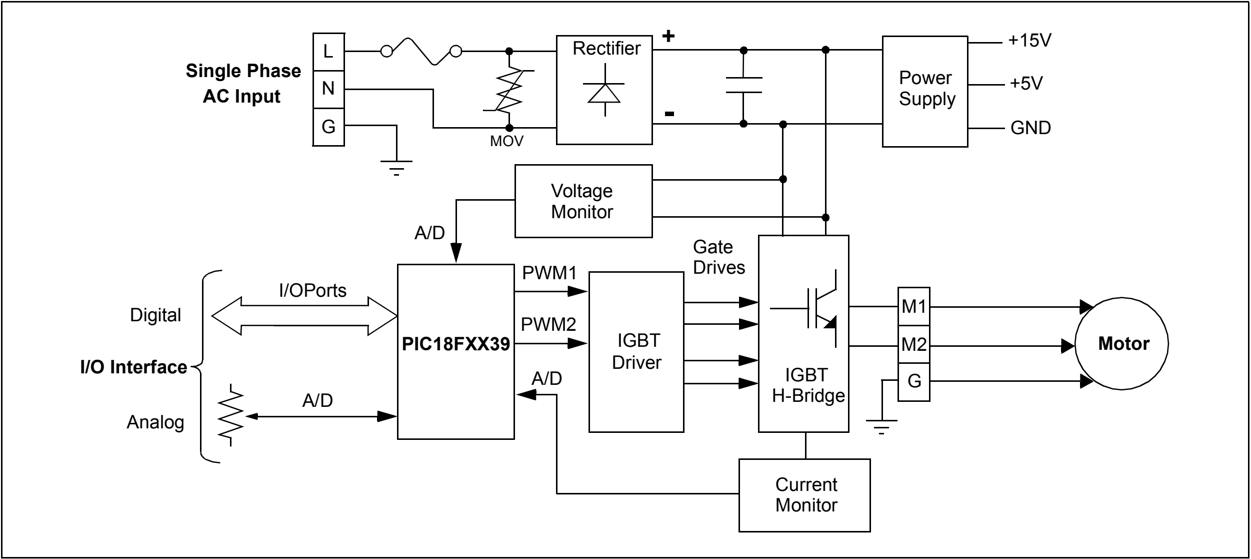
14.2 Typical Hardware Interface

A block diagram for a recommended single phase induction motor control using the PIC18FXX39 is shown in Figure 14-1.

The single phase AC supply is rectified, using a diode bridge and filtered, using a capacitor. The PWM outputs from the PIC18FXX39 synthesize the AC to drive the motor from this DC bus by switching Insulated Gate Bipolar Transistors (IGBTs) on and off. The IGBT gate driver converts the TTL level of PWMs to the required IGBT gate voltage level, and supplies the gate charging current when the IGBT turns on.

The I/O ports of the microcontroller can be used for the external logic controls. The A/D channels can be used for monitoring the DC bus voltage and motor current; a potentiometer can also be connected to one of these channels to provide a variable frequency reference for the motor.

FIGURE 14-1: TYPICAL MOTOR CONTROL SYSTEM USING THE PIC18FXX39



14.3 Software Interface

A sine table, stored in the ProMPT kernel, is used as the basis for synthesizing the DC bus using the PWM modules. The table values are accessed in sequence and scaled based on the frequency or the speed at which the motor is intended to run. The intended frequency input can be from an A/D channel or a digital value.

Parameters in the ProMPT modules can be accessed using the pre-defined Application Program Interface (API) methods. A list of the APIs is given in Section 14.3.3.

For example, to run the motor at 40 Hz, the user would invoke the `PromMPT_SetFrequency` API:

```
i = ProMPT_SetFrequency(40);
```

where `i` is an unsigned character variable. In this case, if `i` = 0 on return, the command has been successfully executed. If the frequency input is out of range, or if there is an error in setting the frequency, `i` is returned with a value of FFh.

Similarly, to check the frequency set by the ProMPT kernel, use the `ProMPT_GetFrequency` API:

```
i = ProMPT_GetFrequency(void);
```

where `i` is an unsigned character variable. Upon return from the ProMPT kernel, `i` will contain the frequency value in the ProMPT kernel.

14.3.1 THE V/F CURVE

The ProMPT kernel contains a default V/F curve stored in memory. The default curve is linear, as shown in Figure 14-2. Table 14-1 shows the data points used to construct the curve.

Users may require a different V/F curve for their application, based on the load on the motor, or based on the characteristics of the motor used. The curve can be changed in the application program using the API method `SetVFCurve(X,Y)`, where `X` is the frequency and `Y` is the level of modulation of the DC bus voltage. As a rule, in customizing the curve, the input frequency corresponding to the point on the V/F curve that gives 100% modulation should match the motor's rated frequency. Similarly, full modulation should occur at the motor's rated input voltage. (See Figure 14-2 for details.)

Examples of the characteristics for V/F curves for typical motor applications are shown in Section 14-2 (page 115).

14.3.2 PARAMETERS DEFINED BY THE ProMPT API METHODS

Frequency: The frequency (in Hz) of the supply current for steady state motor operation.

Modulation: The level of modulation (in percentage) applied to the DC supply voltage by the PWM through the H-bridge to produce AC drive current.

Acceleration rate: The rate of increase of motor speed, achieved by ramping up the supply frequency. Expressed in Hz/s.

Deceleration rate: The rate of decrease of motor speed, achieved by ramping down the supply frequency. Expressed in Hz/s.

Boost: The mode for starting a stopped motor by varying the supply current frequency and modulation until steady state speed is reached. Boost is defined in terms of a frequency, a starting and ending modulation, and a time interval for the transition between the two.

PWM Frequency: The sampling rate (in kHz) at which the PWM module operates.

FIGURE 14-2: DEFAULT V/F CURVE FOR THE ProMPT KERNEL

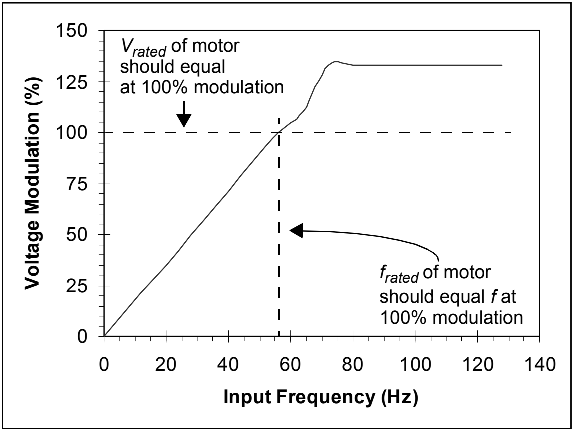


TABLE 14-1: DATA POINTS FOR THE DEFAULT V/F CURVE

Frequency (Hz)	% Modulation
0	0
8	14
16	28
24	42
32	57
40	71
48	86
56	100
64	110
72	133
80	133
88	133
96	133
104	133
112	133
120	133
128	133

TABLE 14-2: ProMPT OUTPUT CHARACTERISTICS FOR VARIOUS V/F CURVES

Motor Type:	**Shaded Pole Blower**
Rated Voltage:	**115V**
Full Load Current:	**3.5/3.25A**
Rated Frequency:	**50/60 Hz**
Rated Speed:	**1570 RPM**
Rated Power:	**1/10 HP**

Input Frequency (Hz)	Measured Frequency (Hz)	Deviation (%)	Measured Output Voltage (RMS)	Measured Output Current (A)	Motor Speed (RPM)
Linear V/F Curve (Pre-programmed)					
15	14.8	1.3	22.8	1.59	348
18	17.8	1.1	28.2	1.75	445
20	19.8	1.0	33.5	1.92	505
25	24.7	1.2	42.0	2.08	651
30	29.7	1.0	52.6	2.26	794
35	34.6	1.1	62.0	2.40	926
40	39.6	1.0	72.3	2.57	1060
45	44.5	1.1	81.3	2.70	1185
50	49.5	1.0	90.7	2.79	1305
55	54.4	1.1	99.6	2.96	1421
60	59.4	1.0	107.8	3.10	1536
65	64.3	1.1	112.3	3.26	1565
70	69.3	1.0	111.5	3.53	1450
75	74.2	1.1	111.3	3.69	1070
Pump V/F Curve					
15	14.8	1.3	15.0	1.00	3.5
18	17.8	1.1	18.4	1.10	396
20	19.8	1.0	21.4	1.23	456
25	24.7	1.2	29.5	1.44	602
30	29.7	1.0	36.6	1.60	722
35	34.6	1.1	44.7	1.79	852
40	39.6	1.0	53.9	2.01	979
45	44.5	1.1	62.9	2.21	1092
50	49.5	1.0	73.4	2.47	1221
55	54.4	1.1	88.2	2.79	1367
60	59.4	1.0	102.0	3.05	1488
65	64.3	1.1	108.8	3.25	1538
70	69.3	1.0	108.0	3.50	1385
75	74.3	0.9	109.1	3.58	994

TABLE 14-2: ProMPT OUTPUT CHARACTERISTICS FOR VARIOUS V/F CURVES (CONTINUED)

Motor Type:	**Shaded Pole Blower**
Rated Voltage:	**115V**
Full Load Current:	**3.5/3.25A**
Rated Frequency:	**50/60 Hz**
Rated Speed:	**1570 RPM**
Rated Power:	**1/10 HP**

Input Frequency (Hz)	Measured Frequency (Hz)	Deviation (%)	Measured Output Voltage (RMS)	Measured Output Current (A)	Motor Speed (RPM)
Strong Fan V/F Curve					
15	14.8	1.3%	6.2	0.45	100
18	17.8	1.1%	8.5	0.57	193
20	19.8	1.0%	11.3	0.69	264
25	24.7	1.2%	17.3	0.94	408
30	29.7	1.0%	24.0	1.17	538
35	34.6	1.1%	31.5	1.43	654
40	39.6	1.0%	38.9	1.66	720
45	44.5	1.1%	49.5	1.96	888
50	49.5	1.0%	61.6	2.26	1040
55	54.4	1.1%	73.5	2.56	1162
60	59.4	1.0%	93.8	2.94	1410
65	64.3	1.1%	106.8	3.24	1534
70	69.3	1.0%	108.9	3.49	1401
75	74.2	1.1%	109.5	3.58	1016
Weak Fan V/F Curve					
15	14.8	1.3%	14.9	0.99	306
18	17.8	1.1%	19.1	1.15	405
20	19.8	1.0%	23.5	1.31	475
25	24.7	1.2%	32.8	1.56	619
30	29.7	1.0%	41.2	1.79	759
35	34.6	1.1%	51.5	2.01	893
40	39.6	1.0%	62.2	2.23	1018
45	44.5	1.1%	73.7	2.47	1155
50	49.4	1.2%	83.0	2.64	1277
55	54.4	1.1%	92.5	2.86	1397
60	59.4	1.0%	103.5	3.06	1498
65	64.3	1.1%	108.0	3.22	1500
70	69.3	1.0%	107.8	3.50	1348
75	74.2	1.1%	108.1	3.55	949

14.3.3 ProMPT API METHODS

There are 27 separate API methods for the ProMPT kernel:

Note: The operation of the Motor Control kernel and its APIs is based on an assumed clock frequency of 20 MHz. Changing the oscillator frequency will change the timing used in the Motor Control kernel accordingly. To achieve the best results in motor control applications, a clock frequency of 20 MHz is highly recommended.

`void ProMPT_ClearTick(void)`

Resources used: 0 stack levels

Description: This function clears the Tick (62.5 ms) timer flag returned by `ProMPT_tick()`. This function must be called by any routine that is used for timing purposes.

`void ProMPT_DisableBoostMode(void)`

Resources used: 0 stack levels

Description: This function disables the Boost mode logic. This method should be called before changing any of the Boost mode parameters.

`void ProMPT_EnableBoostMode(void)`

Resources used: 0 stack levels

Description: This function enables the Boost mode logic. Boost mode is entered when a stopped drive is commanded to start. The drive will immediately go to Boost Frequency and ramp from Start Modulation to End Modulation over the time period, Boost Time.

`unsigned char ProMPT_GetAccelRate(void)`

Resources used: 1 stack level

Range of values: 0 to 255

Description: Returns the current Acceleration Rate in Hz/second.

`unsigned char ProMPT_GetBoostEndModulation(void)`

Resources used: 1 stack level

Range of values: 0 to 200

Description: Returns the current End Modulation (in %) used in the boost logic.

`unsigned char ProMPT_GetBoostFrequency(void)`

Resources used: 1 stack level

Range of values: 0 to 127

Description: Returns the current Boost Frequency in Hz.

`unsigned char ProMPT_GetBoostStartModulation(void)`

Resources used: 1 stack level

Range of values: 0 to `BoostEndModulation`

Description: Returns the Start Modulation (in %) used in the Boost logic.

`unsigned char ProMPT_GetBoostTime()`

Resources used: 1 stack level

Range of values: 0 to 255

Description: Returns the time in seconds for Boost mode.

`unsigned char ProMPT_GetDecelRate()`

Resources used: 1 stack level

Range of values: 0 to 255

Description: Returns the current deceleration rate in Hz/second.

`unsigned char ProMPT_GetFrequency(void)`

Resources used: 1 stack level

Range of values: 0 to 127

Description: Returns the current output frequency in Hz. This may not be the frequency commanded due to Boost or Accel/Decel logic.

`unsigned char ProMPT_GetModulation(void)`

Resources used: Hardware Multiplier; 1 stack level

Range of values: 0 to 200

Description: Returns the current output modulation in %.

`unsigned char ProMPT_GetParameter(unsigned char parameter)`

Resources used: 1 stack level

Description: In addition to its pre-defined API methods, the ProMPT kernel allows the user to custom define up to 16 functions for control or communication purposes not covered by the ProMPT APIs. These parameters are used to communicate with motor control GUI evaluation tools, such as Microchip's DashDriveMP™. This method returns the current value of any one of the parameters.

`unsigned char ProMPT_GetVFCurve(unsigned char point)`

Resources used: Hardware Multiplier; 1 stack level

Description: This function returns one of the 17 modulation values (in %) of the V/F curve. Each point represents a frequency increment of 8 Hz, ranging from point 0 (0 Hz) to point 16 (128 Hz).

`void ProMPT_Init(unsigned char PWMfrequency)`

Resources used: 64 Bytes RAM; Timer2; PWM1 and PWM2; High Priority Interrupt Vector; Hardware Multiplier; fast call/return; FSR 0; TBLPTR; 2 stack levels

`PWMfrequency` values: 0 or 1

Description: This function must be called before all other ProMPT methods, and it must be called only once. This routine configures Timer2 and the PWM outputs.

When `PWMfrequency` is '0', the module's operating frequency is 9.75 kHz. When `PWMfrequency` is '1', the module's operating frequency is 19.53 kHz.

Note: Since the high priority interrupt is used, the fast call/return cannot be used by other routines.

`void ProMPT_SetAccelRate(unsigned char rate)`

Resources used: 0 stack level

`rate` range: 0 to 255

Description: Sets the acceleration to the value of rate in Hz/second. The default setting is 10 Hz/s.

`void ProMPT_SetBoostEndModulation(unsigned char modulation)`

Resources used: Hardware Multiplier; 0 stack levels

`modulation` range: 0 to 200

Description: Sets the End Modulation (in %) for the Boost logic. Boost mode operates at Boost Frequency, and the modulation ramps from `BoostStartModulation` to `BoostEndModulation`. This function should not be called while Boost is enabled.

`unsigned char ProMPT_SetBoostFrequency(unsigned char frequency)`

Resources used: 0 stack levels

`frequency` range: 0 to 127

Description: Sets the frequency the drive goes to in Boost mode. Frequency must be < 128. On exit, w = 0 if the command is successful, or w = FFh if the frequency is out of range. This function should not be called while Boost is enabled.

`void ProMPT_SetBoostStartModulation(unsigned char modulation)`

Resources used: Hardware Multiplier; 0 stack levels

`modulation` range: 0 to `BoostEndModulation`

Description: Sets the Start Modulation (in %) for the Boost logic. Boost mode operates at Boost Frequency, and the modulation ramps from `BoostStartModulation` to `BoostEndModulation`. This function should not be called while Boost is enabled.

`void ProMPT_SetBoostTime(unsigned char time)`

Resources used: Hardware Multiplier; 0 stack levels

`time` range: 0 to 255

Description: Sets the amount of time in seconds for the Boost mode. Boost mode operates at Boost Frequency, and the modulation ramps from `BoostStartModulation` to `BoostEndModulation` over `BoostTime`. This function should not be called while Boost is enabled.

`void ProMPT_SetDecelRate(unsigned char rate)`

Resources used: 0 stack levels

`rate` range: 0 to 255

Description: Sets the deceleration to the value of `rate` in Hz per second. The default setting is 5 Hz/s.

`unsigned char ProMPT_SetFrequency(unsigned char frequency)`

Resources used: 2 stack levels

`frequency` range: 0 to 127

Description: Sets the output frequency of the drive if the drive is running. Frequency is limited to 0 to 127, but should be controlled within the valid operational range of the motor. Modulation is determined from the V/F curve, which is set up with the `ProMPT_SetVFCurve` method. If frequency = 0, the drive will stop. If the drive is stopped and frequency > 0, the drive will start.

`void ProMPT_SetLineVoltage(unsigned char voltage)`

Resources used: Hardware Multiplier; 0 stack levels

`voltage` range: 0 to 255

Description: Sets the line voltage for Automatic Voltage Compensation. The units for `SetLineVoltage` and `SetMotorVoltage` must be the same for accurate operation. The values passed to `SetMotorVoltage` and `SetLineVoltage` can be the same to disable voltage compensation.

`void ProMPT_SetMotorVoltage(unsigned char voltage)`

Resources used: Hardware Multiplier; 0 stack levels

`voltage` range: 0 to 255

Description: Sets the motor rating for Automatic Voltage Compensation. The units for `SetLineVoltage` and `SetMotorVoltage` must be the same for accurate operation. The values passed to `SetMotorVoltage` and `SetLineVoltage` can be the same to disable voltage compensation.

`void ProMPT_SetParameter(unsigned char parameter, unsigned char value)`

Resources used: 0 stack levels

`parameter` range:

Description: In addition to its pre-defined API methods, the ProMPT kernel allows the user to custom define up to 16 functions for control or communication purposes not covered by the ProMPT APIs. This function sets the value of the specified user defined function.

`void ProMPT_SetPWMfrequency(unsigned char PWMfrequency)`

`PWMfrequency` values: 0 or 1

Resources used: Timer2; 1 stack level

Description: This sets and changes the PWM switching frequency. Typically, this is set with the `Init()` function. When `PWMfrequency` is '0', the module's operating frequency is 9.75 kHz. When `PWMfrequency` is '1', the module's operating frequency is 19.53 kHz.

`void ProMPT_SetVFCurve(unsigned char point, unsigned char value)`

Resources used: Hardware Multiplier; 0 stack level

`point` range: 0 to 16 (0 = 0 Hz, 1 = 8 Hz, 2 = 16 Hz……. 17 = 128 Hz)

`value` range: 0 to 200

Description: This sets one of the 17 modulation values (in %) for the V/F curve. Each point represents a frequency increment of 8 Hz, ranging from point 0 (0 Hz) to point 16 (128 Hz).

`unsigned char ProMPT_Tick(void)`

Resources used: 1 stack level

Description: The value of the Tick timer flag becomes '1' every 62.5 ms (1/16 second). This can be used for timing applications. `clearTick` must be called in the timing routine when this is serviced.

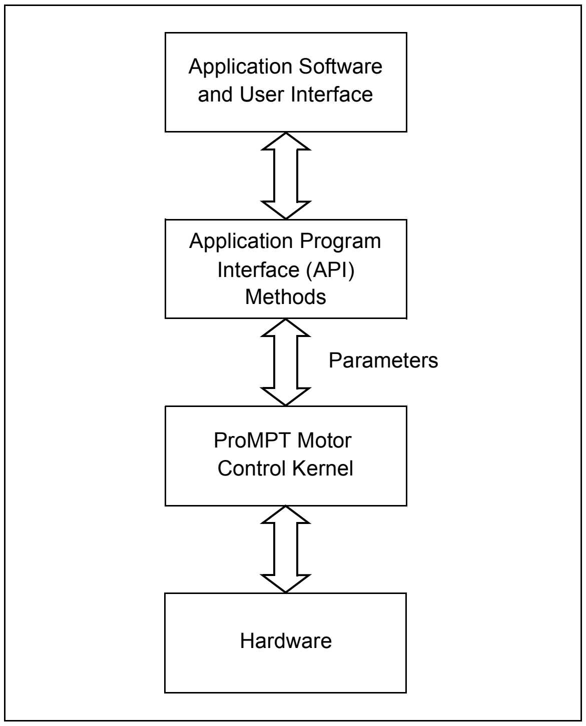
14.4 Developing Applications Using the Motor Control Kernel

The Motor Control kernel allows users to develop their applications without having knowledge of motor control. The key parameters of the motor control kernel can be set and read through the Application Program Interface (API) methods discussed in the previous section.

The overall application can be thought of as a protocol stack, as shown in Figure 14-3. In this case, the API methods reside between the user's application and the ProMPT kernel, and are used to exchange parameter values. The motor control kernel sets the PWM duty cycles based on the inputs from the application software.

A typical motor control routine is shown in Example 14-1. In this case, the motor will run at 20 Hz for 10 seconds, accelerate to 60 Hz at the rate of 10 Hz/s, remain at 60 Hz for 20 seconds, and finally stop.

FIGURE 14-3: LAYERS OF THE MOTOR CONTROL ARCHITECTURE STACK


Application Software and User Interface
Application Program Interface (API) Methods
Parameters
ProMPT Motor Control Kernel
Hardware

EXAMPLE 14-1: MOTOR CONTROL ROUTINE USING THE ProMPT APIs

```
Void main()
{
    unsigned char i;
    unsigned char j;
    ProMPT_Init(0);                      // Initialize the ProMPT block
    i = ProMPT_SetFrequency(10);         // Set motor frequency to 10Hz

    for (i=0;i<161;i++)                 // Set counter for 10 sec @ 1/16 sec per tick
        {
        j = ProMPT_Tick(void);           // Tick of 1/16 sec
        ProMPT_ClearTick(void);          // Clearing the Tick flag
        }

    ProMPT_SetAccelRate(10);            // Set acceleration rate to 10 Hz/sec

    i = ProMPT_SetFrequency(60);        // Set motor frequency to 60 Hz

    for (i=0;i<161;i++)                 // Set counter for 20 Sec @ 1/16 sec per tick
        {                                // (2 loops of 10 Sec each)
        j = ProMPT_Tick(void);           // Tick of 1/16 Sec
        ProMPT_ClearTick(void);          // Clearing the Tick flag
        j = ProMPT_Tick(void);           // Tick of 1/16 Sec
        ProMPT_ClearTick(void);          // Clearing the Tick flag
        }
    i = ProMPT_SetFrequency(0);         // Set motor frequency to 0 Hz (stop)
    while(1);
}                                        // End of the task
```

NOTES:

15.0 PULSE WIDTH MODULATION (PWM) MODULES

PIC18FXX39 devices are equipped with two 10-bit PWM modules. Each contains a register pair (CCPxH:CCPxL), which operates as a Master/Slave Duty Cycle register, and a control register (CCPxCON). The modules use Timer2 (Section 12.0) as their time-base reference. Figure 15-1 shows a simplified block diagram of the module's operation.

This section gives a brief overview of PWM operation as controlled by the Motor Control module (Section 14.0). Operation is described with respect to PWM1, but is equally applicable to PWM2.

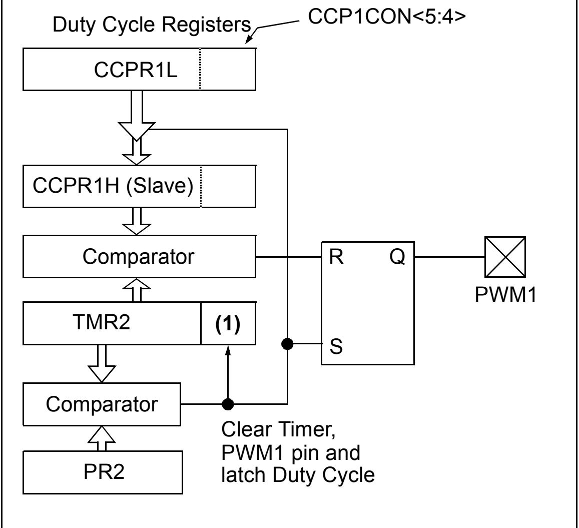
Note:	The PWM modules are used exclusively by the Motor Control module. As such, they are not available to users as a separate resource. Although their locations are shown on the device data memory maps, users should not modify the values of these registers.

15.1 PWM Mode

In Pulse Width Modulation, each PWM pin produces a PWM output with a resolution of up to 10 bits.

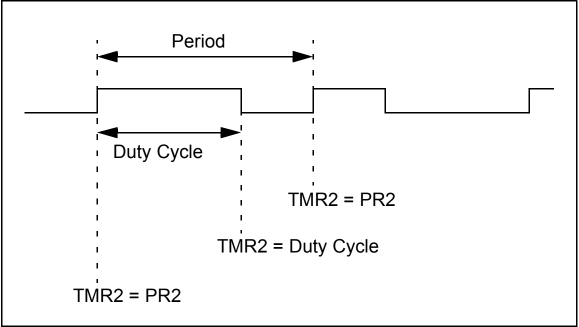
A PWM output (Figure 15-2) has a time-base (period) and a time that the output stays high (duty cycle). The frequency of the PWM is the inverse of the period (1/period).

FIGURE 15-1: SIMPLIFIED PWM BLOCK DIAGRAM



Note 1: 8-bit timer is concatenated with 2-bit internal Q clock, or 2 bits of the prescaler to create a 10-bit time-base.

FIGURE 15-2: PWM OUTPUT



15.1.1 PWM PERIOD

The PWM period is specified when the Motor Control module is initialized. The PWM period can be calculated using the formula:

$$\text{PWM period} = [(\text{PR2}) + 1] \bullet 4 \bullet T_{OSC} \bullet (\text{TMR2 prescale value})$$

PWM frequency is defined as 1 / [PWM period].

The API method `void ProMPT_Init` (page 118) sets the required PWM frequency in the application. The parameter `PWMfrequency` determines the operating frequency of the module. When it is '0', the PWM frequency set in the Motor Control module is 9.75 kHz; when it is '1', the set PWM frequency is 19.53 kHz.

When TMR2 is equal to PR2, the following three events occur on the next increment cycle:

- TMR2 is cleared
- The PWM1 pin is set (exception: if PWM duty cycle = 0%, the PWM1 pin will not be set)
- The PWM duty cycle is latched from CCPR1L into CCPR1H

Note:	The Timer2 postscaler (see Section 12.0) is not used in the determination of the PWM frequency. The postscaler could be used to have a servo update rate at a different frequency than the PWM output.

15.1.2 PWM DUTY CYCLE

The PWM duty cycle is set by the Motor Control module when it writes a 10-bit value to the CCPR1L and CCP1CON registers, where CCPR1L contains the eight Most Significant bits and CCP1CON<5:4> contains the two Least Significant bits. The duty cycle time is given by the equation:

$$\text{PWM duty cycle} = (\text{10-bit CCP register value}) \cdot T_{OSC} \cdot (\text{TMR2 prescale value})$$

where TOSC and the duty cycle are in the same unit of time.

The CCPR1H register and a 2-bit internal latch are used to double-buffer the PWM duty cycle. This buffering is essential for glitchless PWM operation. At the same time, the value of TMR2 is concatenated with either an internal 2-bit Q clock, or 2 bits of the TMR2 prescaler. When the CCPR1H:latch pair value matches that of the TMR2:latch pair, the PWM1 pin is cleared.

The maximum PWM resolution (bits) for a given PWM frequency is given by the equation:

$$\text{PWM Resolution (max)} = \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)}\text{bits}$$

where FPWM is the PWM frequency, or (1/PWM period).

Note:	If the PWM duty cycle value is longer than the PWM period, the PWM1 pin will not be cleared.

TABLE 15-1: 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	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	0000 0000	0000 0000
TMR2*	*	*	*	*	*	*	*	*	0000 0000	0000 0000
PR2*	*	*	*	*	*	*	*	*	1111 1111	1111 1111
T2CON*	*	*	*	*	*	*	*	*	-000 0000	-000 0000
CCPR1L*	*	*	*	*	*	*	*	*	xxxx xxxx	uuuu uuuu
CCPR1H	PWM Register1 (MSB) (read-only)								xxxx xxxx	uuuu uuuu
CCP1CON*	—	—	*	*	*	*	*	*	--00 0000	--00 0000
CCPR2L*	*	*	*	*	*	*	*	*	xxxx xxxx	uuuu uuuu
CCPR2H*	PWM Register2 (MSB) (read-only)								xxxx xxxx	uuuu uuuu
CCP2CON*	—	—	*	*	*	*	*	*	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0' unless otherwise noted. Shaded cells are not used by PWM and Timer2.
* These registers are retained to maintain compatibility with PIC18FXX2 devices; however, the indicated bits are reserved in PIC18FXX39 devices. Users should not alter the values of these bits.
Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18F2X39 devices; always maintain these bits clear.

16.0 MASTER SYNCHRONOUS SERIAL PORT (MSSP) MODULE

16.1 Master SSP (MSSP) Module Overview

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

16.2 Control Registers

The MSSP module has three associated registers. These include a status register (SSPSTAT) and two control registers (SSPCON1 and SSPCON2). The use of these registers and their individual configuration bits differ significantly, depending on whether the MSSP module is operated in SPI or I^2C mode.

Additional details are provided under the individual sections.

16.3 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:

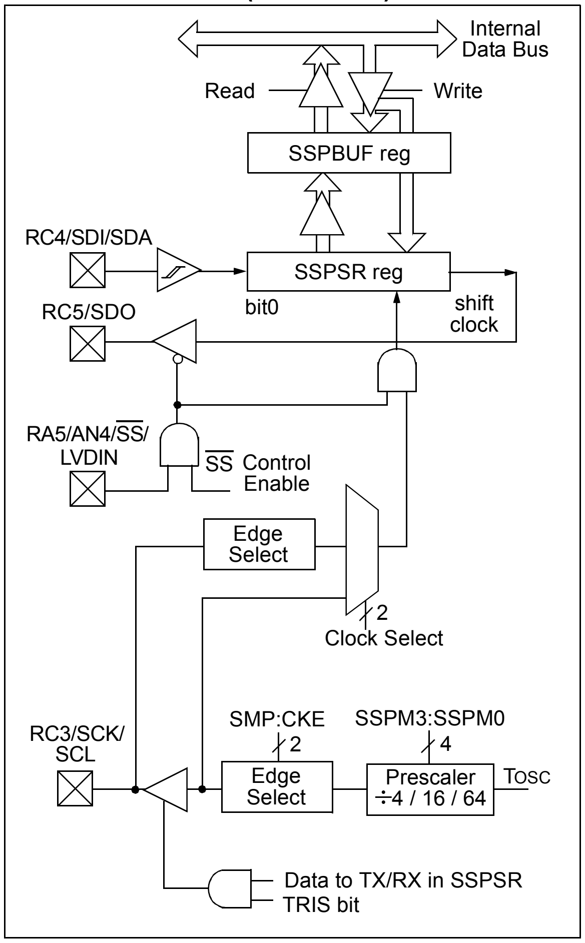
- Serial Data Out (SDO) - RC5/SDO
- Serial Data In (SDI) - RC4/SDI/SDA
- Serial Clock (SCK) - RC3/SCK/SCL

Additionally, a fourth pin may be used when in a Slave mode of operation:

- Slave Select ($\overline{SS}$) - RA5/AN4/$\overline{SS}$/LVDIN

Figure 16-1 shows the block diagram of the MSSP module when operating in SPI mode.

FIGURE 16-1: MSSP BLOCK DIAGRAM (SPI MODE)



16.3.1 REGISTERS

The MSSP module has four registers for SPI mode operation. These are:

- MSSP Control Register1 (SSPCON1)
- MSSP Status Register (SSPSTAT)
- Serial Receive/Transmit Buffer (SSPBUF)
- MSSP Shift Register (SSPSR) - Not directly accessible

SSPCON1 and SSPSTAT are the control and status registers in SPI mode operation. The SSPCON1 register is readable and writable. The lower 6 bits of the SSPSTAT are read only. The upper two bits of the SSPSTAT are read/write.

SSPSR is the shift register used for shifting data in or out. SSPBUF is the buffer register to which data bytes are written to or read from.

In receive operations, SSPSR and SSPBUF together create a double-buffered receiver. When SSPSR receives a complete byte, it is transferred to SSPBUF and the SSPIF interrupt is set.

During transmission, the SSPBUF is not double-buffered. A write to SSPBUF will write to both SSPBUF and SSPSR.

REGISTER 16-1: SSPSTAT: MSSP STATUS REGISTER (SPI MODE)

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

bit 6 **CKE:** SPI Clock Edge Select bit
When CKP = 0:
`1` = Data transmitted on rising edge of SCK
`0` = Data transmitted on falling edge of SCK
When 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
Used in I^2C mode only

bit 4 **P:** STOP bit
Used in I^2C mode only. This bit is cleared when the MSSP module is disabled, SSPEN is cleared.

bit 3 **S:** START bit
Used in I^2C mode only

bit 2 **$R/\overline{W}$:** Read/Write bit information
Used in I^2C mode only

bit 1 **UA:** Update Address
Used in I^2C mode only

bit 0 **BF:** Buffer Full Status bit (Receive mode only)
`1` = Receive complete, SSPBUF is full
`0` = Receive not complete, SSPBUF is empty

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

REGISTER 16-2: SSPCON1: MSSP CONTROL REGISTER 1 (SPI MODE)

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 (Transmit mode only)

`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

<u>SPI Slave mode:</u>

`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.The user must read the SSPBUF, even if only transmitting data, to avoid setting overflow (must be cleared in software).

`0` = No overflow

Note: In Master mode, the overflow bit is not set since each new reception (and transmission) is initiated by writing to the SSPBUF register.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit

`1` = Enables serial port and configures SCK, SDO, SDI, and $\overline{SS}$ as serial port pins

`0` = Disables serial port and configures these pins as I/O port pins

Note: When enabled, these pins must be properly configured as input or output.

bit 4 **CKP:** Clock Polarity Select bit

`1` = IDLE state for clock is a high level

`0` = IDLE state for clock is a low level

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits

`0101` = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control disabled, $\overline{SS}$ can be used as I/O pin

`0100` = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control enabled

`0011` = Reserved

`0010` = SPI Master mode, clock = FOSC/64

`0001` = SPI Master mode, clock = FOSC/16

`0000` = SPI Master mode, clock = FOSC/4

Note: Bit combinations not specifically listed here are either reserved, or implemented in I^{2}C mode only.

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

16.3.2 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)

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<0>), 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<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 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 16-1 shows the loading of the SSPBUF (SSPSR) for data transmission.

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.

EXAMPLE 16-1: LOADING THE SSPBUF (SSPSR) REGISTER

```
LOOP BTFSS SSPSTAT, BF    ;Has data been received(transmit complete)?
     BRA   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
```

16.3.3 ENABLING SPI I/O

To enable the serial port, SSP Enable bit, SSPEN (SSPCON1<5>), 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 TRISC<5> bit cleared
- SCK (Master mode) must have TRISC<3> bit cleared
- SCK (Slave mode) must have TRISC<3> bit set
- $\overline{SS}$ must have TRISC<4> bit set

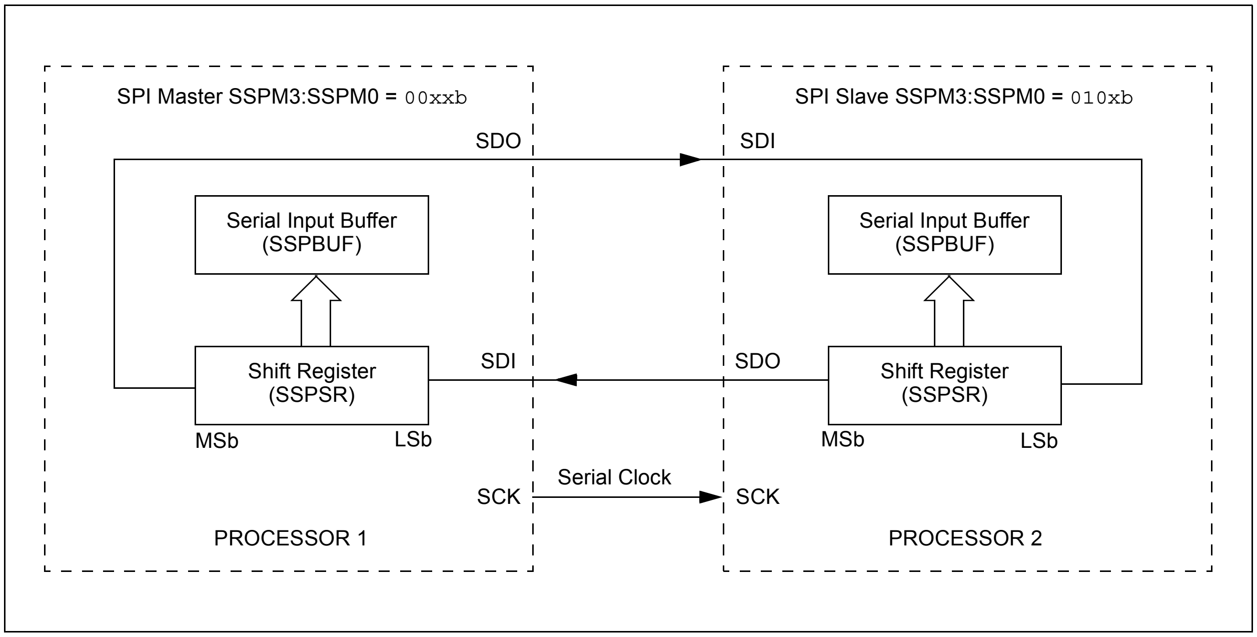
Any serial port function that is not desired may be overridden by programming the corresponding data direction (TRIS) register to the opposite value.

16.3.4 TYPICAL CONNECTION

Figure 16-2 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 opposite edge of the clock. Both processors should be programmed to the 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 16-2: SPI MASTER/SLAVE CONNECTION


SPI Master SSPM3:SSPM0 = 00xxb
SPI Slave SSPM3:SSPM0 = 010xb
SDO
SDI
Serial Input Buffer (SSPBUF)
Serial Input Buffer (SSPBUF)
Shift Register (SSPSR)
SDI
SDO
Shift Register (SSPSR)
MSb
LSb
MSb
LSb
Serial Clock
SCK
SCK
PROCESSOR 1
PROCESSOR 2

16.3.5 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 16-2) 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.

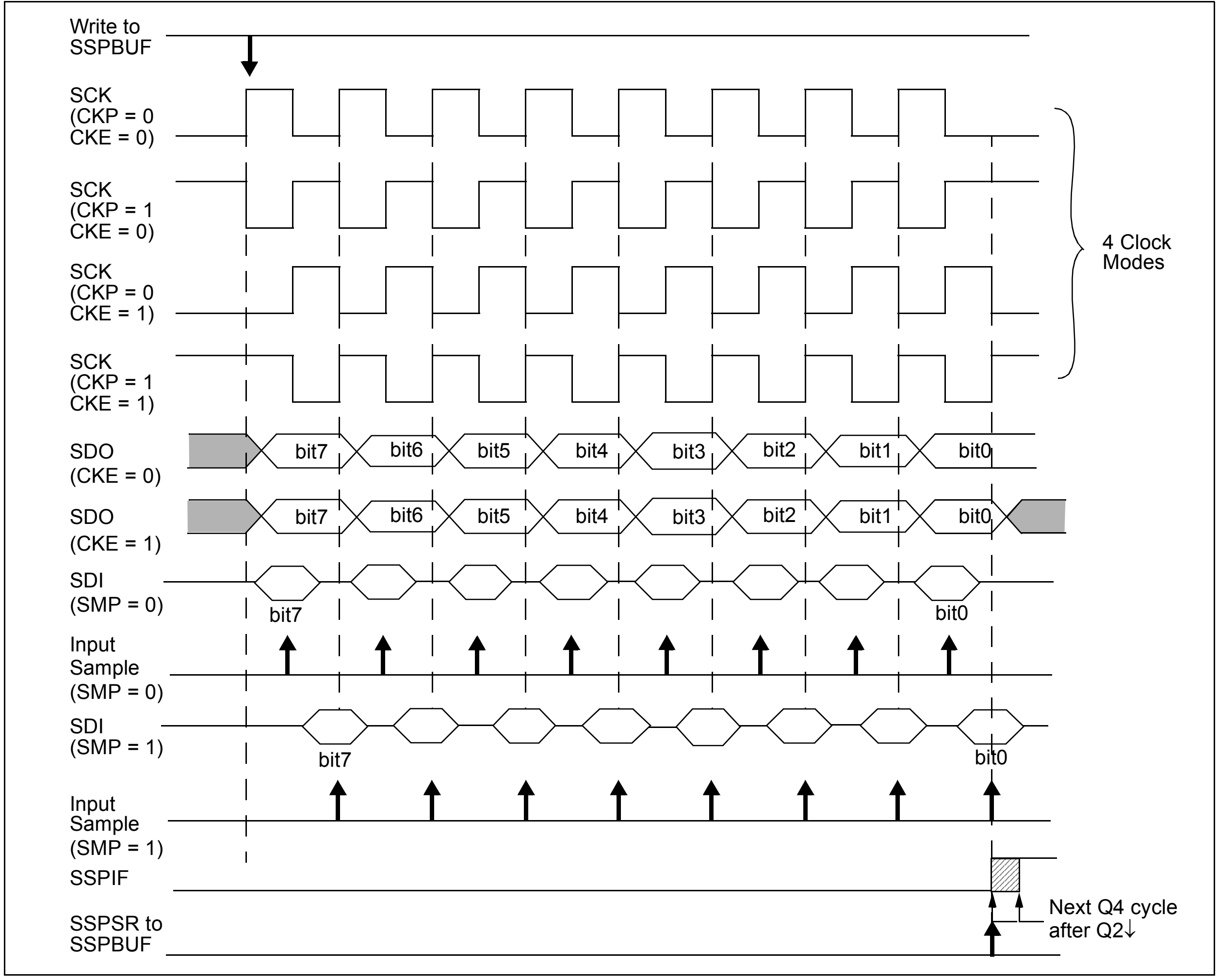
The clock polarity is selected by appropriately programming the CKP bit (SSPCON1<4>). This then, would give waveforms for SPI communication as shown in Figure 16-3, Figure 16-5, and Figure 16-6, where the MSB is transmitted first. In Master mode, the SPI clock rate (bit rate) is user-programmable to be one of the following:

- FOSC/4 (or TCY)
- FOSC/16 (or 4 • TCY)
- FOSC/64 (or 16 • TCY)

This allows a maximum data rate (at 40 MHz) of 10.00 Mbps.

Figure 16-3 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 16-3: SPI MODE WAVEFORM (MASTER MODE)



16.3.6 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.

16.3.7 SLAVE SELECT SYNCHRONIZATION

The $\overline{SS}$ pin allows a Synchronous Slave mode. The SPI must be in Slave mode with $\overline{SS}$ pin control enabled (SSPCON1<3:0> = 04h). The pin must not be driven low for the $\overline{SS}$ pin to function as an input. The Data Latch must be high. 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.

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.

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 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 16-4: SLAVE SYNCHRONIZATION WAVEFORM

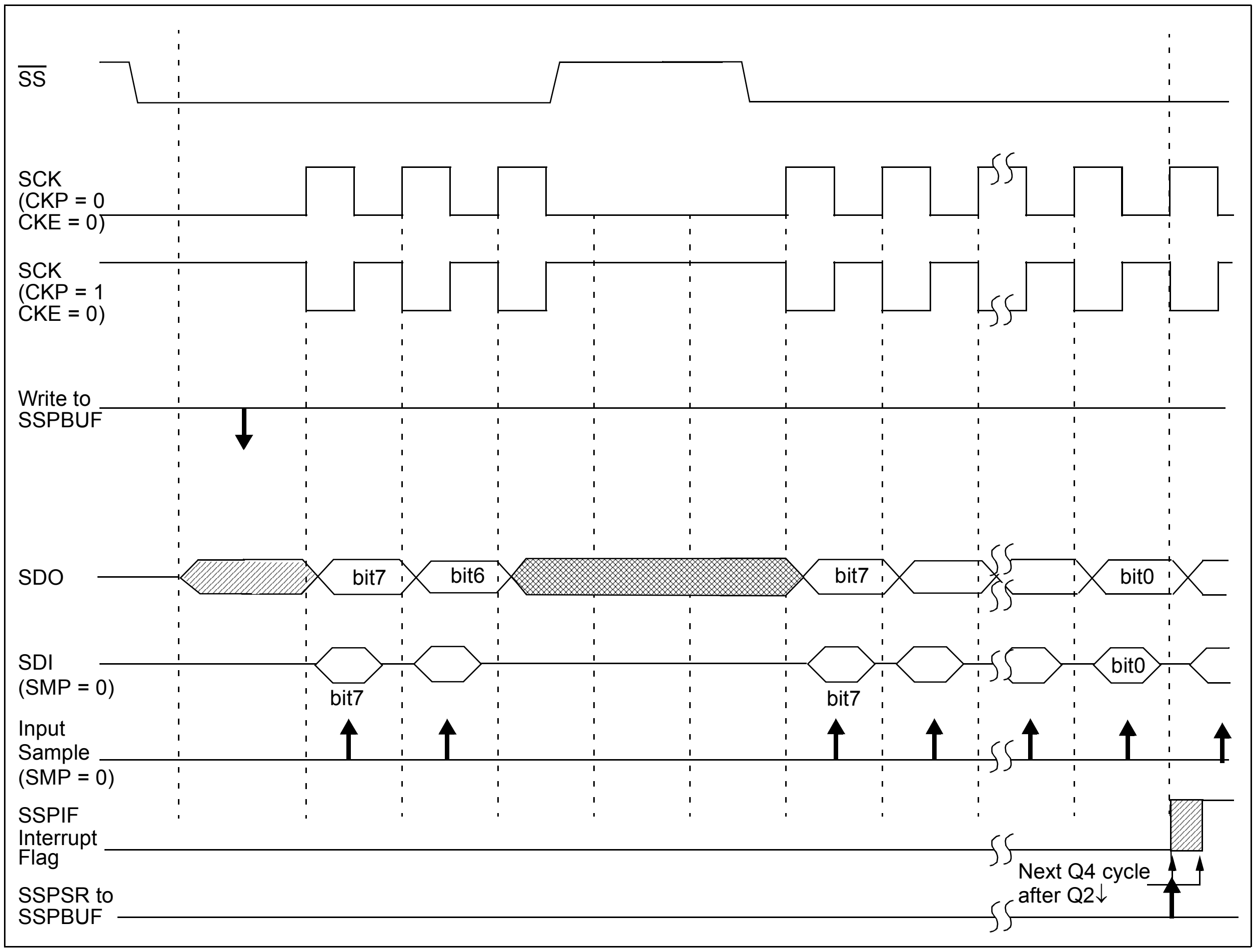


FIGURE 16-5: SPI MODE WAVEFORM (SLAVE MODE WITH CKE = 0)

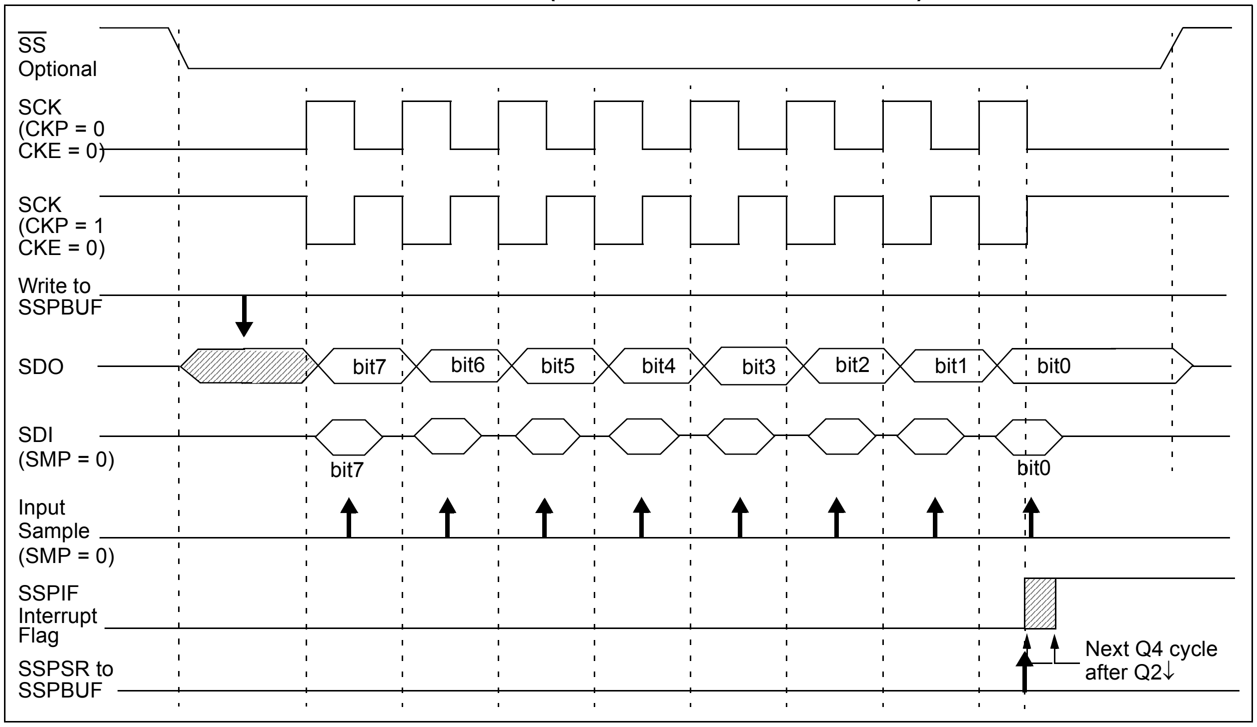
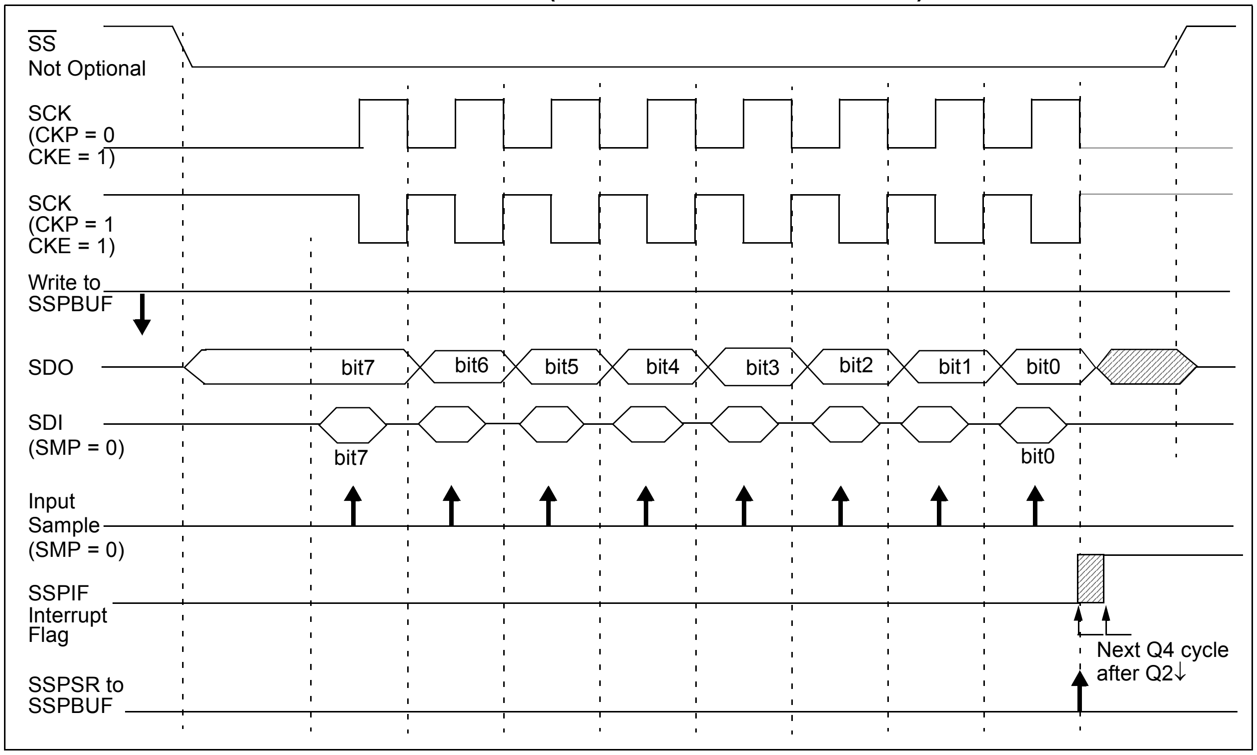


FIGURE 16-6: SPI MODE WAVEFORM (SLAVE MODE WITH CKE = 1)



16.3.8 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 MSSP interrupt flag bit will be set and if enabled, will wake the device from SLEEP.

16.3.9 EFFECTS OF A RESET

A RESET disables the MSSP module and terminates the current transfer.

16.3.10 BUS MODE COMPATIBILITY

Table 16-1 shows the compatibility between the standard SPI modes and the states the CKP and CKE control bits.

TABLE 16-1: 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 an SMP bit which controls when the data is sampled.

TABLE 16-2: 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	—	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	0000 0000	0000 0000
TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	*	*	TRISC0	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							-111 1111	-111 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.
* Reserved bits; do not modify.

Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18C2X2 devices; always maintain these bits clear.

16.4 I^{2}C Mode

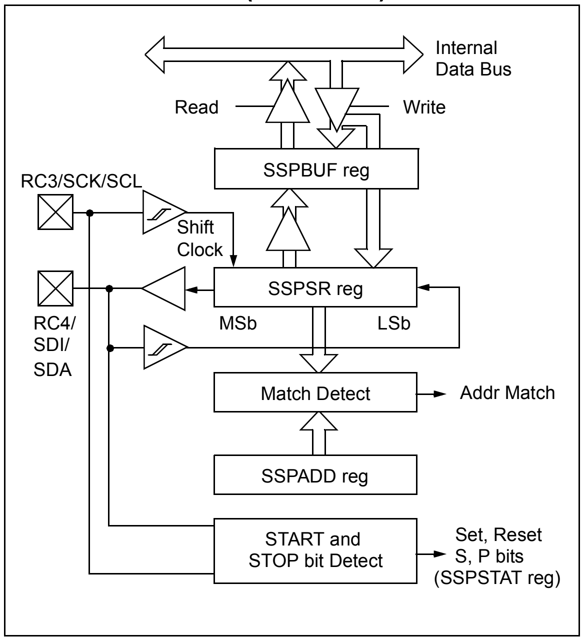
The MSSP module in I^{2}C mode fully implements all master and slave functions (including general call support) and provides interrupts on START and STOP bits in hardware to determine a free bus (multi-master function). The MSSP module implements the Standard mode specifications, as well as 7-bit and 10-bit addressing.

Two pins are used for data transfer:

- Serial clock (SCL) - RC3/SCK/SCL
- Serial data (SDA) - RC4/SDI/SDA

The user must configure these pins as inputs or outputs through the TRISC<4:3> bits.

FIGURE 16-7: MSSP BLOCK DIAGRAM (I^{2}C MODE)



16.4.1 REGISTERS

The MSSP module has six registers for I^{2}C operation. These are:

- 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)

SSPCON, SSPCON2 and SSPSTAT are the control and status registers in I^{2}C mode operation. The SSPCON and SSPCON2 registers are readable and writable. The lower 6 bits of the SSPSTAT are read only. The upper two bits of the SSPSTAT are read/write.

SSPSR is the shift register used for shifting data in or out. SSPBUF is the buffer register to which data bytes are written to or read from.

SSPADD register holds the slave device address when the SSP is configured in I^{2}C Slave mode. When the SSP is configured in Master mode, the lower seven bits of SSPADD act as the baud rate generator reload value.

In receive operations, SSPSR and SSPBUF together, create a double-buffered receiver. When SSPSR receives a complete byte, it is transferred to SSPBUF and the SSPIF interrupt is set.

During transmission, the SSPBUF is not double-buffered. A write to SSPBUF will write to both SSPBUF and SSPSR.

REGISTER 16-3: SSPSTAT: MSSP STATUS REGISTER (I^2C MODE)

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:** Slew Rate Control bit
In 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:** SMBus Select bit
In Master or Slave mode:
1 = Enable SMBus specific inputs
0 = Disable SMBus specific inputs

bit 5 **D/$\overline{A}$:** Data/$\overline{\text{Address}}$ bit
In Master mode:
Reserved
In Slave mode:
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
1 = Indicates that a STOP bit has been detected last
0 = STOP bit was not detected last

Note: This bit is cleared on RESET and when SSPEN is cleared.

bit 3 **S:** START bit
1 = Indicates that a START bit has been detected last
0 = START bit was not detected last

Note: This bit is cleared on RESET and when SSPEN is cleared.

bit 2 **R/$\overline{W}$:** Read/Write bit Information (I^2C mode only)
In Slave mode:
1 = Read
0 = Write

Note: 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 Master mode:
1 = Transmit is in progress
0 = Transmit is not in progress

Note: ORing 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 Slave 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
In Transmit mode:
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty
In Receive mode:
1 = Data transmit in progress (does not include the $\overline{\text{ACK}}$ and STOP bits), SSPBUF is full
0 = Data transmit complete (does not include the $\overline{\text{ACK}}$ and STOP bits), SSPBUF is empty

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

REGISTER 16-4: SSPCON1: MSSP CONTROL REGISTER 1 (I^2C MODE)

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

In Master Transmit mode:
`1` = A write to the SSPBUF register was attempted while the I^2C conditions were not valid for a transmission to be started (must be cleared in software)
`0` = No collision

In Slave Transmit mode:
`1` = The SSPBUF register is written while it is still transmitting the previous word (must be cleared in software)
`0` = No collision

In Receive mode (Master or Slave modes):
This is a "don't care" bit

bit 6 **SSPOV:** Receive Overflow Indicator bit

In Receive mode:
`1` = A byte is received while the SSPBUF register is still holding the previous byte (must be cleared in software)
`0` = No overflow

In Transmit mode:
This is a "don't care" bit in Transmit mode

bit 5 **SSPEN:** Synchronous Serial Port Enable bit
`1` = Enables the serial port and configures the SDA and SCL pins as the serial port pins
`0` = Disables serial port and configures these pins as I/O port pins

Note: When enabled, the SDA and SCL pins must be properly configured as input or output.

bit 4 **CKP:** SCK Release Control bit

In Slave mode:
`1` = Release clock
`0` = Holds clock low (clock stretch), used to ensure data setup time

In Master mode:
Unused in this mode

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits
`1111` = I^2C Slave mode, 10-bit address with START and STOP bit interrupts enabled
`1110` = I^2C Slave mode, 7-bit address with START and STOP bit interrupts enabled
`1011` = I^2C Firmware Controlled Master mode (Slave IDLE)
`1000` = I^2C Master mode, clock = FOSC / (4 * (SSPADD+1))
`0111` = I^2C Slave mode, 10-bit address
`0110` = I^2C Slave mode, 7-bit address

Note: Bit combinations not specifically listed here are either reserved, or implemented in SPI mode only.

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

REGISTER 16-5: SSPCON2: MSSP CONTROL REGISTER 2 (I^2C MODE)

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 (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 (Master Transmit mode only)
1 = Acknowledge was not received from slave
0 = Acknowledge was received from slave

bit 5 **ACKDT:** Acknowledge Data bit (Master Receive mode only)
1 = Not Acknowledge
0 = Acknowledge

Note: Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.

bit 4 **ACKEN:** Acknowledge Sequence Enable bit (Master Receive mode only)
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 (Master mode only)
1 = Enables Receive mode for I^2C
0 = Receive IDLE

bit 2 **PEN:** STOP Condition Enable bit (Master mode only)
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 (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/Stretch Enabled bit
In Master mode:
1 = Initiate START condition on SDA and SCL pins. Automatically cleared by hardware.
0 = START condition IDLE
In Slave mode:
1 = Clock stretching is enabled for both Slave Transmit and Slave Receive (stretch enabled)
0 = Clock stretching is enabled for slave transmit only (Legacy mode)

Note: For bits ACKEN, RCEN, PEN, RSEN, SEN: If the I^2C module is not in the IDLE mode, this bit may not be set (no spooling) and the SSPBUF may not be written (or writes to the SSPBUF are disabled).

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

16.4.2 OPERATION

The MSSP module functions are enabled by setting MSSP Enable bit, SSPEN (SSPCON<5>).

The SSPCON1 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 Master mode, clock = OSC/4 (SSPADD +1)
- I^2C Slave mode (7-bit address)
- I^2C Slave mode (10-bit address)
- 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

Selection of any I^2C mode, with the SSPEN bit set, forces the SCL and SDA pins to be open drain, provided these pins are programmed to inputs by setting the appropriate TRISC bits. To ensure proper operation of the module, pull-up resistors must be provided externally to the SCL and SDA pins.

16.4.3 SLAVE MODE

In Slave mode, the SCL and SDA pins must be configured as inputs (TRISC<4:3> set). The MSSP module will override the input state with the output data when required (slave-transmitter).

The I^2C Slave mode hardware will always generate an interrupt on an address match. Through the mode select bits, the user can also choose to interrupt on START and STOP bits

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 load the SSPBUF register with the received value currently in the SSPSR register.

Any combination of the following conditions will cause the MSSP module not to give this $\overline{ACK}$ pulse:

- The buffer full bit BF (SSPSTAT<0>) was set before the transfer was received.
- The overflow bit SSPOV (SSPCON<6>) was set before the transfer was received.

In this case, the SSPSR register value is not loaded into the SSPBUF, but bit SSPIF (PIR1<3>) is set. 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 for proper operation. The high and low times of the I^2C specification, as well as the requirement of the MSSP module, are shown in timing parameter 100 and parameter 101.

16.4.3.1 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. 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:

1. The SSPSR register value is loaded into the SSPBUF register.
2. The buffer full bit BF is set.
3. An $\overline{ACK}$ pulse is generated.
4. MSSP interrupt flag bit, SSPIF (PIR1<3>) 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. Bit R/$\overline{W}$ (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 '`11110 A9 A8 0`', where '`A9`' and '`A8`' are the two MSbs of the address. The sequence of events for 10-bit address is as follows, with steps 7 through 9 for the slave-transmitter:

1. Receive first (high) byte of Address (bits SSPIF, BF and bit UA (SSPSTAT<1>) are set).
2. Update the SSPADD register with second (low) byte of Address (clears bit UA and releases the SCL line).
3. Read the SSPBUF register (clears bit BF) and clear flag bit SSPIF.
4. Receive second (low) byte of Address (bits SSPIF, BF, and UA are set).
5. Update the SSPADD register with the first (high) byte of Address. If match releases SCL line, this will clear bit UA.
6. Read the SSPBUF register (clears bit BF) and clear flag bit SSPIF.
7. Receive Repeated START condition.
8. Receive first (high) byte of Address (bits SSPIF and BF are set).
9. Read the SSPBUF register (clears bit BF) and clear flag bit SSPIF.

16.4.3.2 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 and the SDA line is held low ($\overline{ACK}$).

When the address byte overflow condition exists, then the no Acknowledge ($\overline{ACK}$) pulse is given. An overflow condition is defined as either bit BF (SSPSTAT<0>) is set, or bit SSPOV (SSPCON1<6>) is set.

An MSSP interrupt is generated for each data transfer byte. Flag bit SSPIF (PIR1<3>) must be cleared in software. The SSPSTAT register is used to determine the status of the byte.

If SEN is enabled (SSPCON1<0> = 1), RC3/SCK/SCL will be held low (clock stretch) following each data transfer. The clock must be released by setting bit CKP (SSPCON<4>). See Section 16.4.4 (“Clock Stretching”), for more detail.

16.4.3.3 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 pin RC3/SCK/SCL is held low, regardless of SEN (see “Clock Stretching”, Section 16.4.4, for more detail). By stretching the clock, the master will be unable to assert another clock pulse until the slave is done preparing the transmit data. The transmit data must be loaded into the SSPBUF register, which also loads the SSPSR register. Then, pin RC3/SCK/SCL should be enabled by setting bit CKP (SSPCON1<4>). 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 16-9).

The $\overline{ACK}$ pulse from the master-receiver is latched on the rising edge of the ninth SCL input pulse. If the SDA line is high (not $\overline{ACK}$), then the data transfer is complete. In this case, when the $\overline{ACK}$ is latched by the slave, the slave logic is reset (resets SSPSTAT register) and the slave monitors for another occurrence of the START bit. If the SDA line was low ($\overline{ACK}$), the next transmit data must be loaded into the SSPBUF register. Again, pin RC3/SCK/SCL must be enabled by setting bit CKP.

An MSSP interrupt is generated for each data transfer byte. The SSPIF bit must be cleared in software and the SSPSTAT register is used to determine the status of the byte. The SSPIF bit is set on the falling edge of the ninth clock pulse.

FIGURE 16-8: I^{2}C SLAVE MODE TIMING WITH SEN = 0 (RECEPTION, 7-BIT ADDRESS)

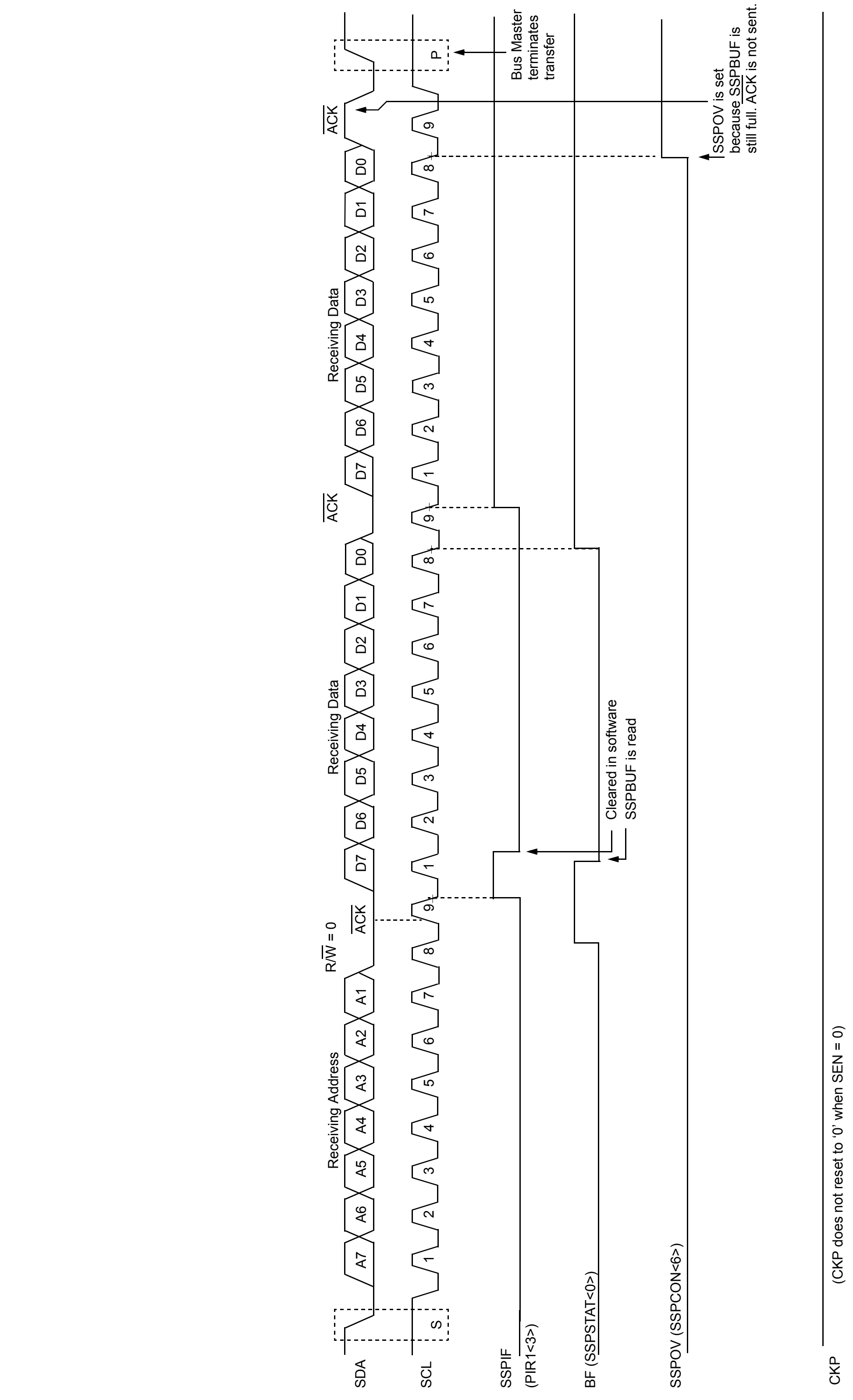


FIGURE 16-9: I^2C SLAVE MODE TIMING (TRANSMISSION, 7-BIT ADDRESS)

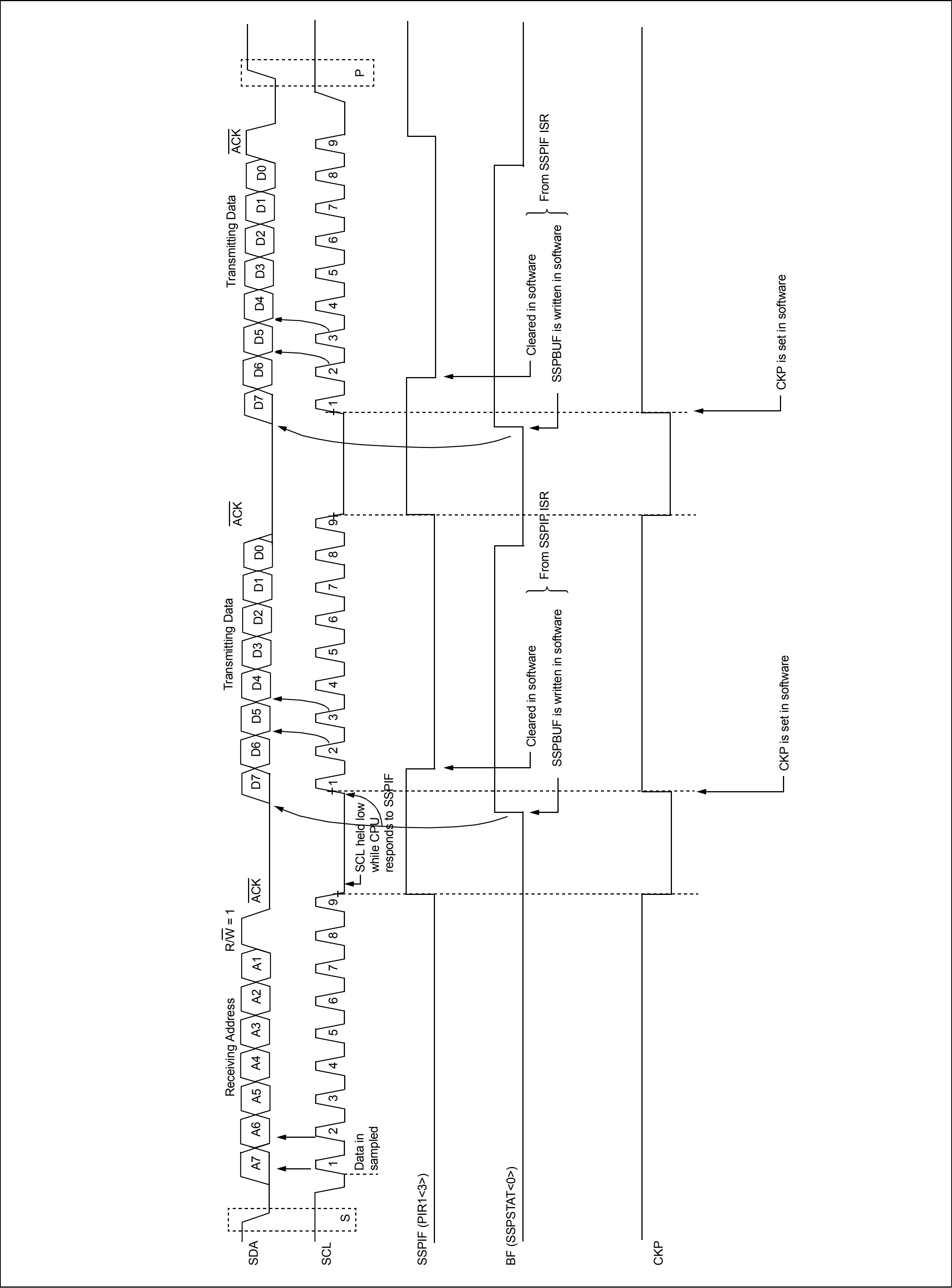


FIGURE 16-10: I^{2}C SLAVE MODE TIMING WITH SEN = 0 (RECEPTION, 10-BIT ADDRESS)

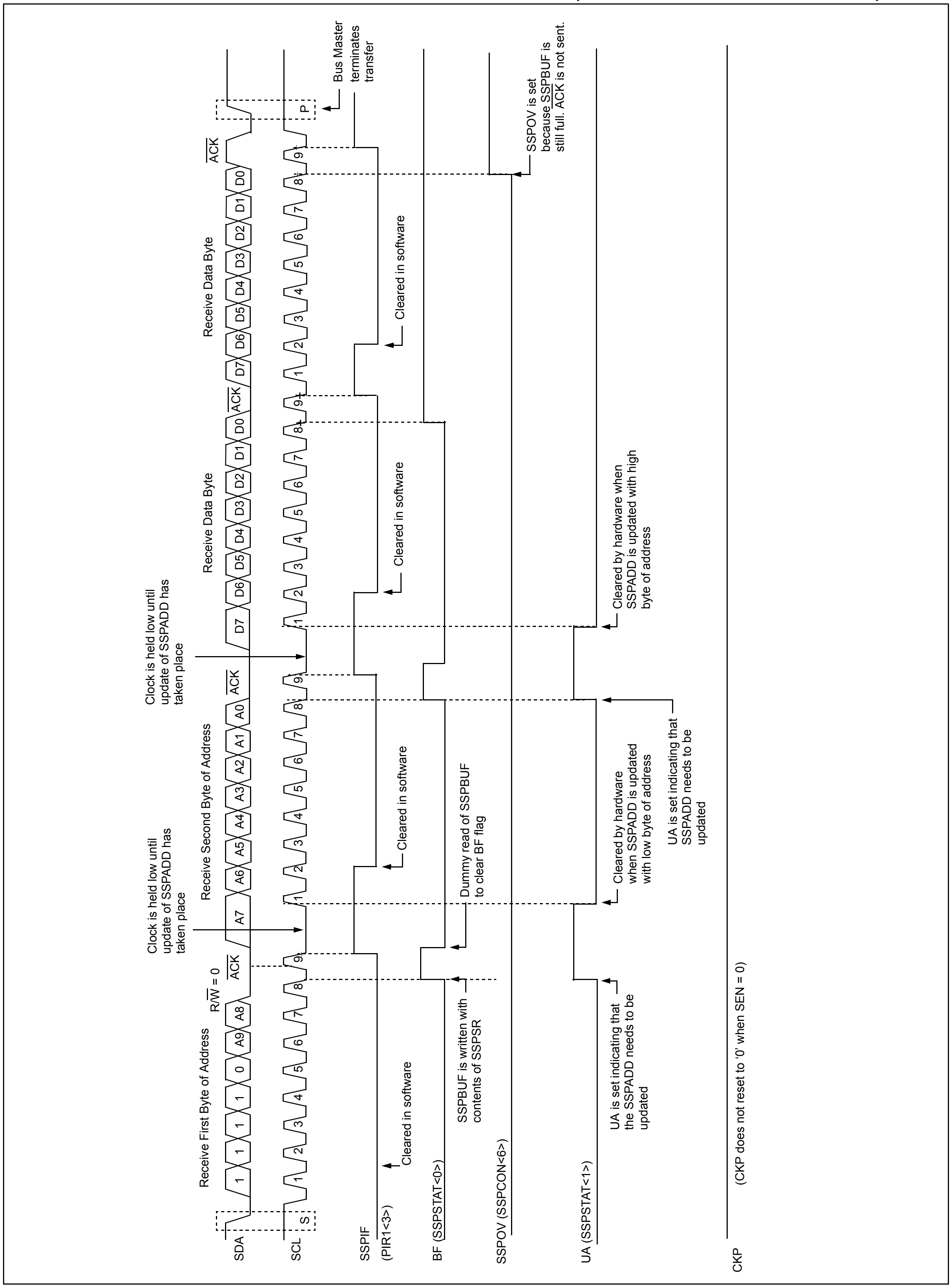
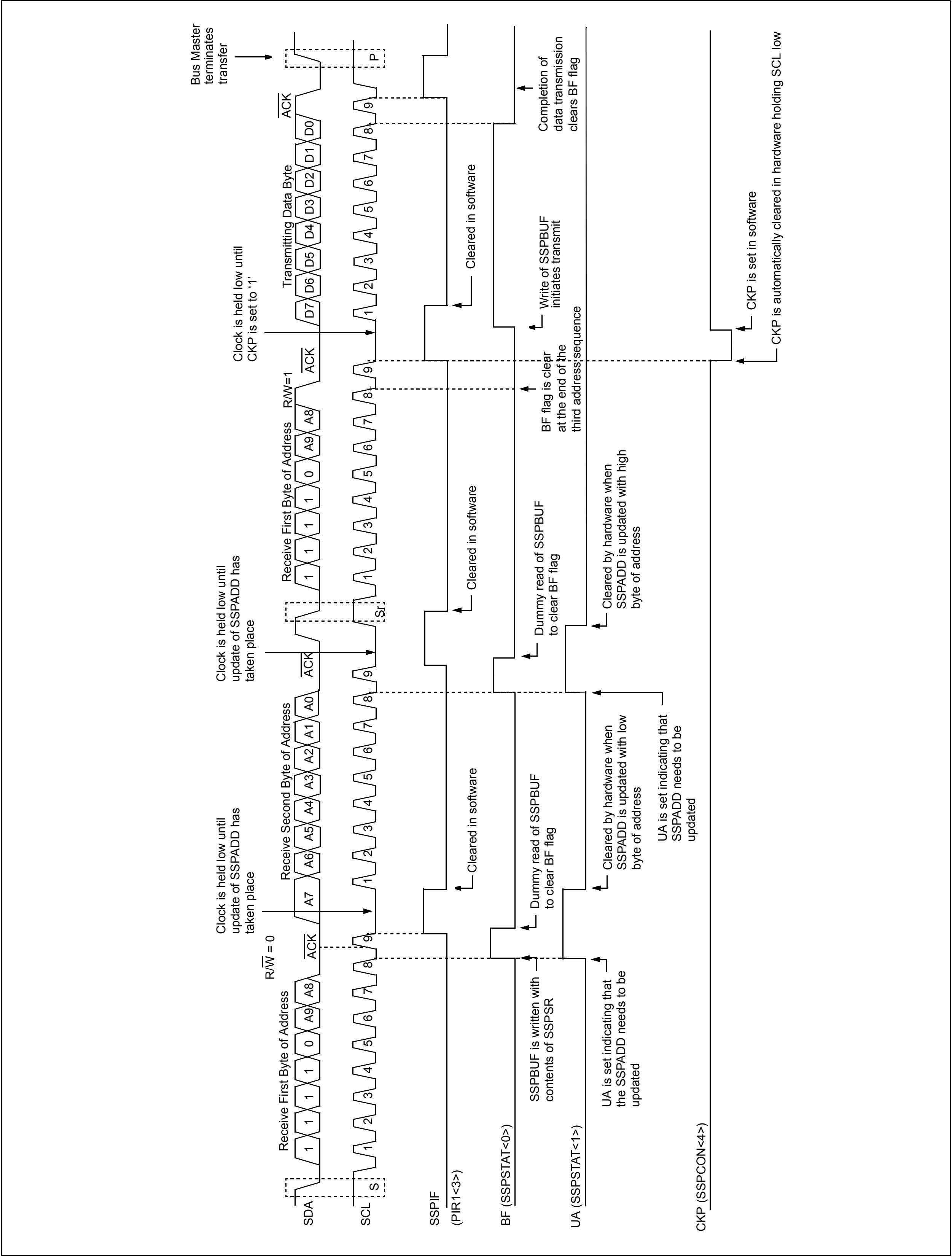

SDA
SCL
SSPIF
(PIR1<3>)
BF (SSPSTAT<0>)
SSPOV (SSPCON<6>)
UA (SSPSTAT<1>)
CKP
S
P
Receive First Byte of Address
R/W = 0
ACK
Receive Second Byte of Address
Receive Data Byte
Receive Data Byte
Clock is held low until update of SSPADD has taken place
Clock is held low until update of SSPADD has taken place
Cleared in software
Cleared in software
Cleared in software
Cleared in software
SSPBUF is written with contents of SSPSR
Dummy read of SSPBUF to clear BF flag
UA is set indicating that the SSPADD needs to be updated
Cleared by hardware when SSPADD is updated with low byte of address
UA is set indicating that SSPADD needs to be updated
Cleared by hardware when SSPADD is updated with high byte of address
SSPOV is set because SSPBUF is still full. ACK is not sent.
Bus Master terminates transfer
(CKP does not reset to '0' when SEN = 0)

FIGURE 16-11: I^{2}C SLAVE MODE TIMING (TRANSMISSION, 10-BIT ADDRESS)


SDA
SCL
SSPIF
(PIR1<3>)
BF (SSPSTAT<0>)
UA (SSPSTAT<1>)
CKP (SSPCON<4>)
S
Sr
P
Receive First Byte of Address
R/W = 0
ACK
Receive Second Byte of Address
Receive First Byte of Address
R/W=1
Transmitting Data Byte
Bus Master terminates transfer
Clock is held low until update of SSPADD has taken place
Clock is held low until update of SSPADD has taken place
Clock is held low until CKP is set to '1'
Cleared in software
Cleared in software
Cleared in software
SSPBUF is written with contents of SSPSR
Dummy read of SSPBUF to clear BF flag
Dummy read of SSPBUF to clear BF flag
BF flag is clear at the end of the third address sequence
Write of SSPBUF initiates transmit
Completion of data transmission clears BF flag
UA is set indicating that the SSPADD needs to be updated
Cleared by hardware when SSPADD is updated with low byte of address
UA is set indicating that SSPADD needs to be updated
Cleared by hardware when SSPADD is updated with high byte of address
CKP is set in software
CKP is automatically cleared in hardware holding SCL low

16.4.4 CLOCK STRETCHING

Both 7- and 10-bit Slave modes implement automatic clock stretching during a transmit sequence.

The SEN bit (SSPCON2<0>) allows clock stretching to be enabled during receives. Setting SEN will cause the SCL pin to be held low at the end of each data receive sequence.

16.4.4.1 Clock Stretching for 7-bit Slave Receive Mode (SEN = 1)

In 7-bit Slave Receive mode, on the falling edge of the ninth clock at the end of the $\overline{\text{ACK}}$ sequence, if the BF bit is set, the CKP bit in the SSPCON1 register is automatically cleared, forcing the SCL output to be held low. The CKP being cleared to '0' will assert the SCL line low. The CKP bit must be set in the user's ISR before reception is allowed to continue. By holding the SCL line low, the user has time to service the ISR and read the contents of the SSPBUF before the master device can initiate another receive sequence. This will prevent buffer overruns from occurring (see Figure 16-13).

Note 1: If the user reads the contents of the SSPBUF before the falling edge of the ninth clock, thus clearing the BF bit, the CKP bit will not be cleared and clock stretching will not occur.

2: The CKP bit can be set in software, regardless of the state of the BF bit. The user should be careful to clear the BF bit in the ISR before the next receive sequence, in order to prevent an overflow condition.

16.4.4.2 Clock Stretching for 10-bit Slave Receive Mode (SEN = 1)

In 10-bit Slave Receive mode, during the address sequence, clock stretching automatically takes place but CKP is not cleared. During this time, if the UA bit is set after the ninth clock, clock stretching is initiated. The UA bit is set after receiving the upper byte of the 10-bit address, and following the receive of the second byte of the 10-bit address with the R/$\overline{\text{W}}$ bit cleared to '0'. The release of the clock line occurs upon updating SSPADD. Clock stretching will occur on each data receive sequence, as described in 7-bit mode.

Note: If the user polls the UA bit and clears it by updating the SSPADD register before the falling edge of the ninth clock occurs, and if the user hasn't cleared the BF bit by reading the SSPBUF register before that time, then the CKP bit will still NOT be asserted low. Clock stretching on the basis of the state of the BF bit only occurs during a data sequence, not an address sequence.

16.4.4.3 Clock Stretching for 7-bit Slave Transmit Mode

7-bit Slave Transmit mode implements clock stretching by clearing the CKP bit after the falling edge of the ninth clock, if the BF bit is clear. This occurs, regardless of the state of the SEN bit.

The user's ISR must set the CKP bit before transmission is allowed to continue. By holding the SCL line low, the user has time to service the ISR and load the contents of the SSPBUF before the master device can initiate another transmit sequence (see Figure 16-9).

Note 1: If the user loads the contents of SSPBUF, setting the BF bit before the falling edge of the ninth clock, the CKP bit will not be cleared and clock stretching will not occur.

2: The CKP bit can be set in software, regardless of the state of the BF bit.

16.4.4.4 Clock Stretching for 10-bit Slave Transmit Mode

In 10-bit Slave Transmit mode, clock stretching is controlled during the first two address sequences by the state of the UA bit, just as it is in 10-bit Slave Receive mode. The first two addresses are followed by a third address sequence, which contains the high order bits of the 10-bit address and the R/$\overline{\text{W}}$ bit set to '1'. After the third address sequence is performed, the UA bit is not set, the module is now configured in Transmit mode, and clock stretching is controlled by the BF flag, as in 7-bit Slave Transmit mode (see Figure 16-11).

16.4.4.5 Clock Synchronization and the CKP bit

If a user clears the CKP bit, the SCL output is forced to '0'. Setting the CKP bit will not assert the SCL output low until the SCL output is already sampled low. If the user attempts to drive SCL low, the CKP bit will not assert the SCL line until an external I^2C master device has already asserted the SCL line. The SCL output will remain low until the CKP bit is set, and all other devices on the I^2C bus have de-asserted SCL. This ensures that a write to the CKP bit will not violate the minimum high time requirement for SCL (see Figure 16-12).

FIGURE 16-12: CLOCK SYNCHRONIZATION TIMING

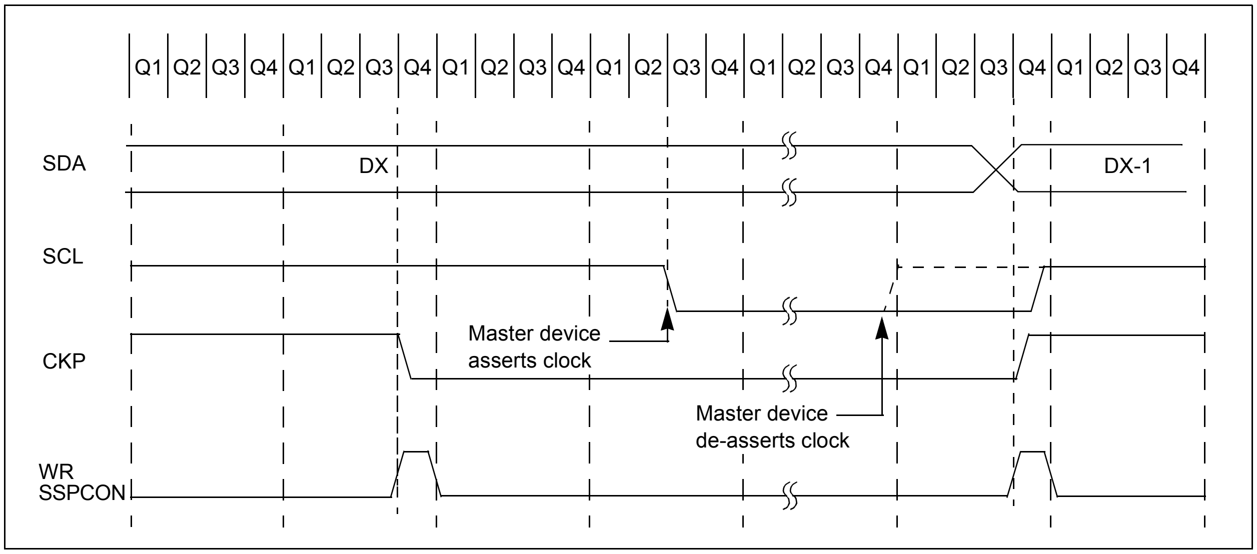

Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4
SDA
DX
DX-1
SCL
CKP
Master device asserts clock
Master device de-asserts clock
WR SSPCON

FIGURE 16-13: I^2C SLAVE MODE TIMING WITH SEN = 1 (RECEPTION, 7-BIT ADDRESS)

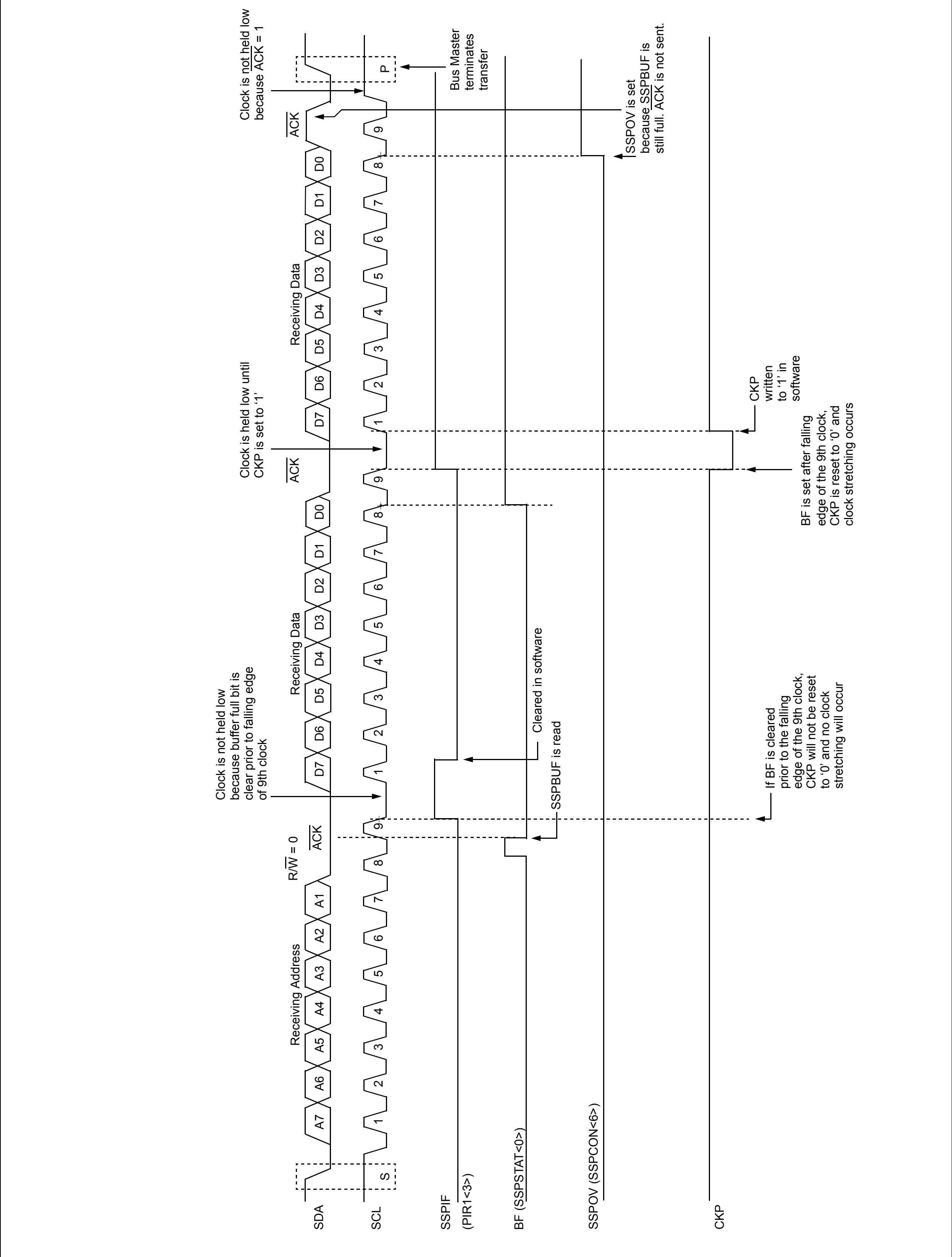
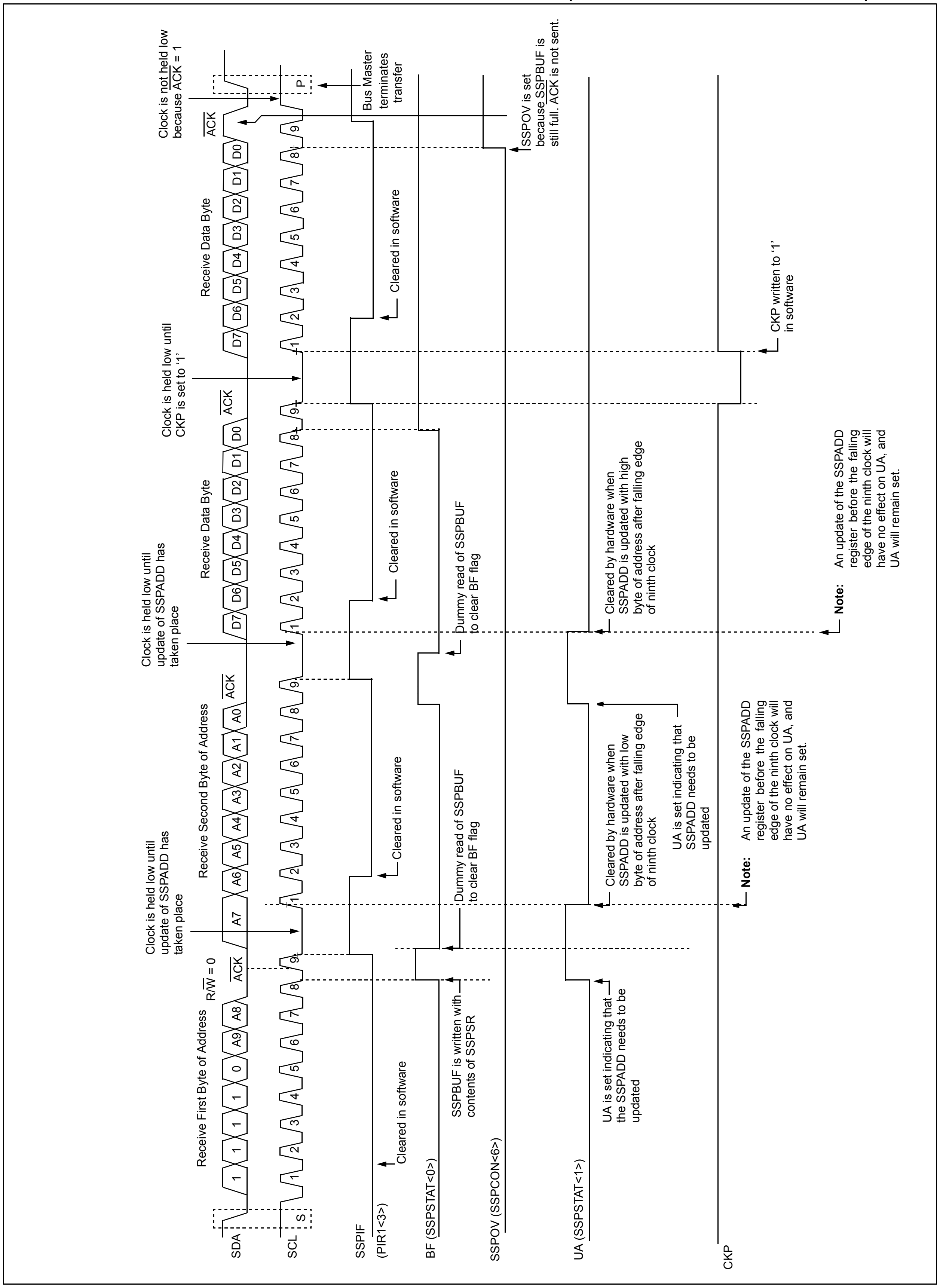

SDA
SCL
SSPIF (PIR1<3>)
BF (SSPSTAT<0>)
SSPOV (SSPCON<6>)
CKP
S
P
A7
A6
A5
A4
A3
A2
A1
R/W = 0
ACK
D7
D6
D5
D4
D3
D2
D1
D0
Receiving Address
Receiving Data
Receiving Data
Clock is not held low because buffer full bit is clear prior to falling edge of 9th clock
Clock is held low until CKP is set to '1'
Clock is not held low because ACK = 1
Bus Master terminates transfer
SSPOV is set because SSPBUF is still full. ACK is not sent.
Cleared in software
SSPBUF is read
If BF is cleared prior to the falling edge of the 9th clock, CKP will not be reset to '0' and no clock stretching will occur
BF is set after falling edge of the 9th clock, CKP is reset to '0' and clock stretching occurs
CKP written to '1' in software

FIGURE 16-14: I²C SLAVE MODE TIMING WITH SEN = 1 (RECEPTION, 10-BIT ADDRESS)


SDA
SCL
SSPIF
(PIR1<3>)
BF (SSPSTAT<0>)
SSPOV (SSPCON<6>)
UA (SSPSTAT<1>)
CKP
S
P
Receive First Byte of Address
R/W = 0
ACK
Receive Second Byte of Address
Receive Data Byte
Receive Data Byte
1 1 1 1 0 A9 A8
A7 A6 A5 A4 A3 A2 A1 A0
D7 D6 D5 D4 D3 D2 D1 D0
1 2 3 4 5 6 7 8 9
Clock is held low until update of SSPADD has taken place
Clock is held low until update of SSPADD has taken place
Clock is held low until CKP is set to '1'
Clock is not held low because ACK = 1
Cleared in software
Cleared in software
Cleared in software
Cleared in software
SSPBUF is written with contents of SSPSR
Dummy read of SSPBUF to clear BF flag
Dummy read of SSPBUF to clear BF flag
Bus Master terminates transfer
SSPOV is set because SSPBUF is still full. ACK is not sent.
UA is set indicating that the SSPADD needs to be updated
Cleared by hardware when SSPADD is updated with low byte of address after falling edge of ninth clock
UA is set indicating that SSPADD needs to be updated
Cleared by hardware when SSPADD is updated with high byte of address after falling edge of ninth clock
Note: An update of the SSPADD register before the falling edge of the ninth clock will have no effect on UA, and UA will remain set.
Note: An update of the SSPADD register before the falling edge of the ninth clock will have no effect on UA, and UA will remain set.
CKP written to '1' in software

16.4.5 GENERAL CALL ADDRESS SUPPORT

The addressing procedure for the I²C 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, in theory, respond with an Acknowledge.

The general call address is one of eight addresses reserved for specific purposes by the I²C protocol. It consists of all ‘0’s with $R/\overline{W} = 0$.

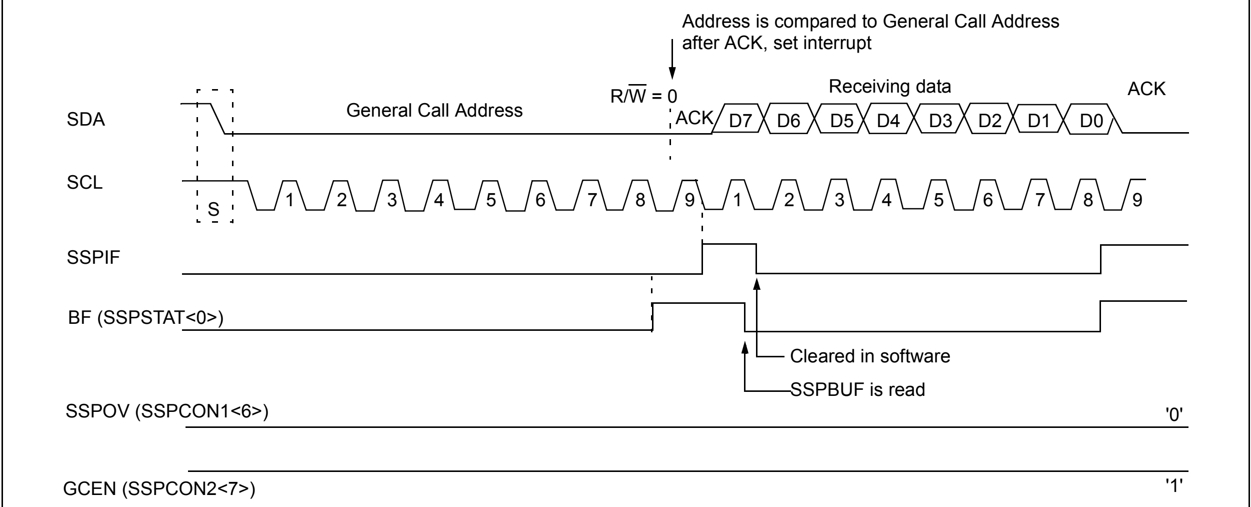
The general call address is recognized when the General Call Enable bit (GCEN) is enabled (SSPCON2<7> set). Following a START bit detect, 8-bits are shifted into the SSPSR and the address is compared against the SSPADD. It is also compared to the general call address and fixed in hardware.

If the general call address matches, the SSPSR is transferred to the SSPBUF, the BF flag bit is set (eighth bit), and on the falling edge of the ninth bit ($\overline{ACK}$ 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. The value can be used to determine if the address was device specific or a general call address.

In 10-bit mode, the SSPADD is required to be updated for the second half of the address to match, and the UA bit is set (SSPSTAT<1>). If the general call address is sampled when the GCEN bit is set, while the slave is configured in 10-bit Address mode, then the second half of the address is not necessary, the UA bit will not be set, and the slave will begin receiving data after the Acknowledge (Figure 16-15).

FIGURE 16-15: SLAVE MODE GENERAL CALL ADDRESS SEQUENCE (7 OR 10-BIT ADDRESS MODE)



16.4.6 MASTER MODE

Master mode is enabled by setting and clearing the appropriate SSPM bits in SSPCON1 and by setting the SSPEN bit. In Master mode, the SCL and SDA lines are manipulated by the MSSP hardware.

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 MSSP module is disabled. Control of the I^{2}C bus may be taken when the P bit is set or the bus is IDLE, with both the S and P bits clear.

In Firmware Controlled Master mode, user code conducts all I^{2}C bus operations based on START and STOP bit conditions.

Once Master mode is enabled, the user has six options.

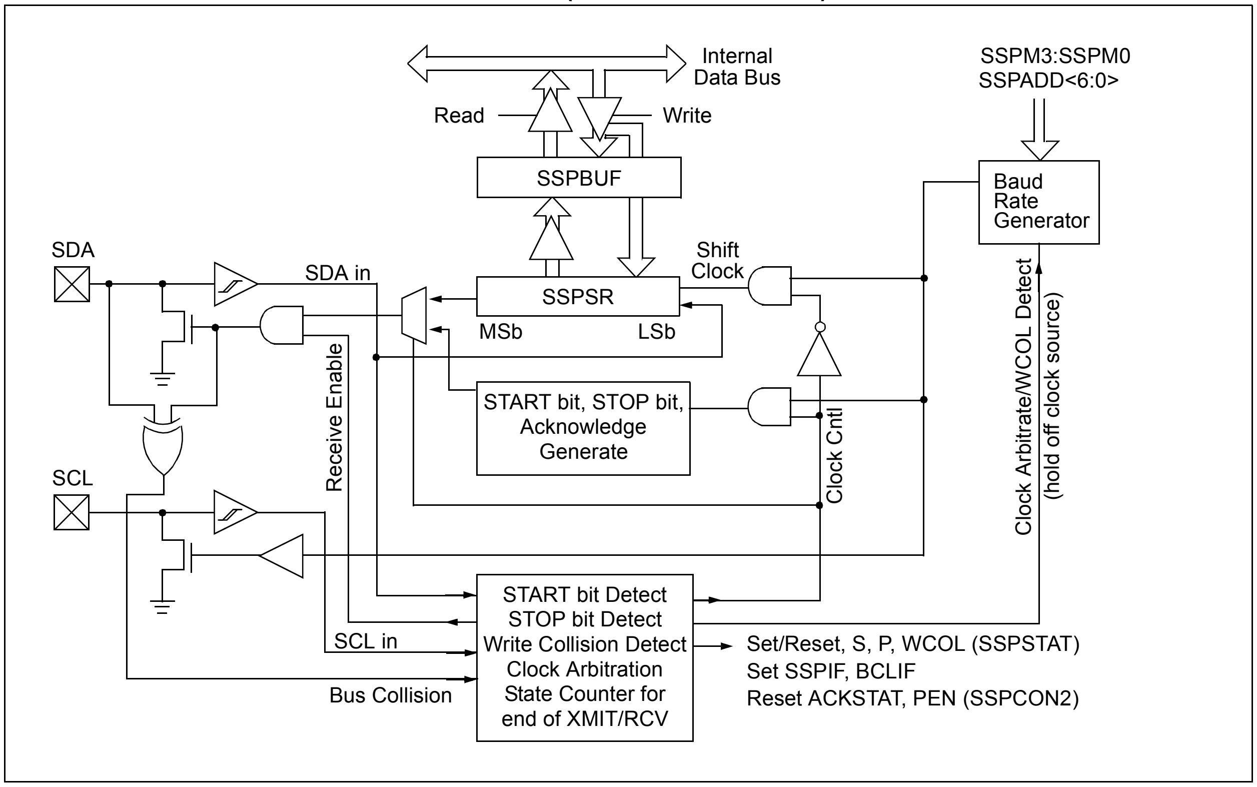
1. Assert a START condition on SDA and SCL.
2. Assert a Repeated START condition on SDA and SCL.
3. Write to the SSPBUF register initiating transmission of data/address.
4. Configure the I^{2}C port to receive data.
5. Generate an Acknowledge condition at the end of a received byte of data.
6. Generate a STOP condition on SDA and SCL.

Note: The MSSP Module, when configured in I^{2}C 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 initiate 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 a write to the SSPBUF did not occur.

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 16-16: MSSP BLOCK DIAGRAM (I^{2}C MASTER MODE)



16.4.6.1 I^{2}C 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 I^{2}C 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 the receive bit. Serial data is received via SDA, while SCL 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 the SPI mode operation is used to set the SCL clock frequency for either 100 kHz, 400 kHz or 1 MHz I^{2}C operation. See Section 16.4.7 (“Baud Rate Generator”), for more detail.

A typical transmit sequence would go as follows:

1. The user generates a START condition by setting the START enable bit, SEN (SSPCON2<0>).
2. SSPIF is set. The MSSP module will wait the required start time before any other operation takes place.
3. The user loads the SSPBUF with the slave address to transmit.
4. Address is shifted out the SDA pin until all 8 bits are transmitted.
5. The MSSP module shifts in the ACK bit from the slave device and writes its value into the SSPCON2 register (SSPCON2<6>).
6. The MSSP module generates an interrupt at the end of the ninth clock cycle by setting the SSPIF bit.
7. The user loads the SSPBUF with eight bits of data.
8. Data is shifted out the SDA pin until all 8 bits are transmitted.
9. The MSSP module shifts in the ACK bit from the slave device and writes its value into the SSPCON2 register (SSPCON2<6>).
10. The MSSP module generates an interrupt at the end of the ninth clock cycle by setting the SSPIF bit.
11. The user generates a STOP condition by setting the STOP enable bit PEN (SSPCON2<2>).
12. Interrupt is generated once the STOP condition is complete.

16.4.7 BAUD RATE GENERATOR

In I^{2}C Master mode, the baud rate generator (BRG) reload value is placed in the lower 7 bits of the SSPADD register (Figure 16-17). When a write occurs to SSPBUF, the baud rate generator will automatically begin counting. 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^{2}C Master mode, the BRG is reloaded automatically.

Once the given operation is complete (i.e., transmission of the last data bit is followed by $\overline{\text{ACK}}$), the internal clock will automatically stop counting and the SCL pin will remain in its last state.

Table 15-3 demonstrates clock rates based on instruction cycles and the BRG value loaded into SSPADD.

FIGURE 16-17: BAUD RATE GENERATOR BLOCK DIAGRAM

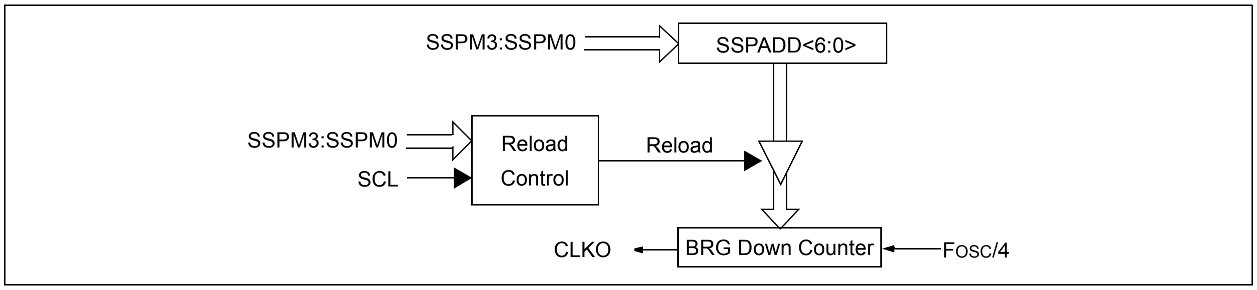


TABLE 16-3: I^{2}C CLOCK RATE W/BRG

FCY	FCY*2	BRG Value	FSCL(2) (2 Rollovers of BRG)
10 MHz	20 MHz	19h	400 kHz(1)
10 MHz	20 MHz	20h	312.5 kHz
10 MHz	20 MHz	3Fh	100 kHz
4 MHz	8 MHz	0Ah	400 kHz(1)
4 MHz	8 MHz	0Dh	308 kHz
4 MHz	8 MHz	28h	100 kHz
1 MHz	2 MHz	03h	333 kHz(1)
1 MHz	2 MHz	0Ah	100kHz
1 MHz	2 MHz	00h	1 MHz(1)

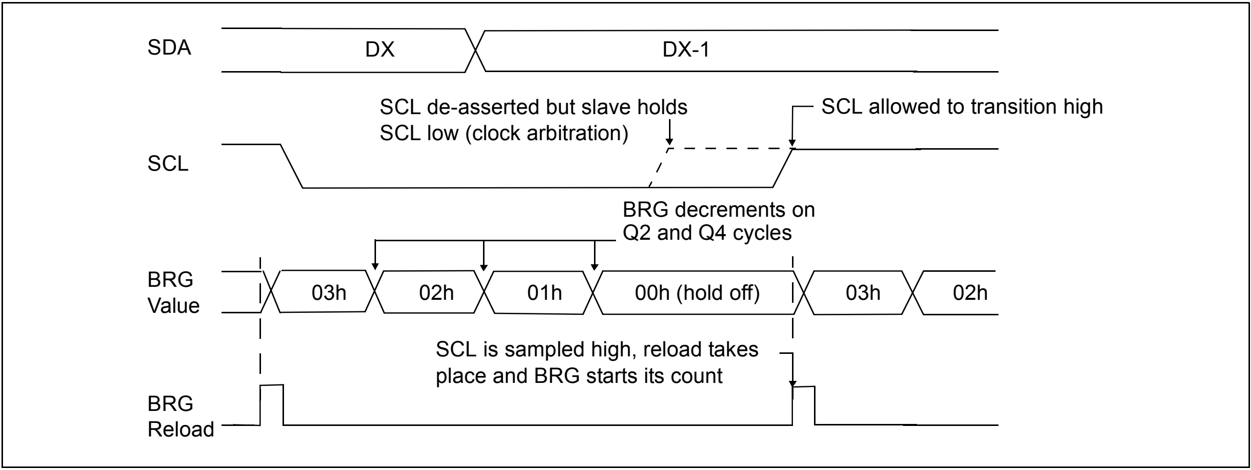
Note 1: The I^{2}C interface does not conform to the 400 kHz I^{2}C specification (which applies to rates greater than 100 kHz) in all details, but may be used with care where higher rates are required by the application.

2: Actual frequency will depend on bus conditions. Theoretically, bus conditions will add rise time and extend low time of clock period, producing the effective frequency.

16.4.7.1 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 16-18).

FIGURE 16-18: BAUD RATE GENERATOR TIMING WITH CLOCK ARBITRATION


SDA
DX
DX-1
SCL de-asserted but slave holds SCL low (clock arbitration)
SCL allowed to transition high
SCL
BRG decrements on Q2 and Q4 cycles
BRG Value
03h
02h
01h
00h (hold off)
03h
02h
SCL is sampled high, reload takes place and BRG starts its count
BRG Reload

16.4.8 I^{2}C MASTER MODE START CONDITION TIMING

To initiate a START condition, the user sets the START condition enable bit, SEN (SSPCON2<0>). If the SDA and SCL pins are sampled high, the baud rate generator is reloaded with the contents of SSPADD<6:0> and starts its count. If SCL and SDA are both sampled high when the baud rate generator times out (TBRG), the SDA pin is driven low. The action of the SDA being driven low, while SCL is high, is the START condition and causes the S bit (SSPSTAT<3>) 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<0>) will be automatically cleared by hardware, the baud rate generator is suspended, leaving the SDA line held low and the START condition is complete.

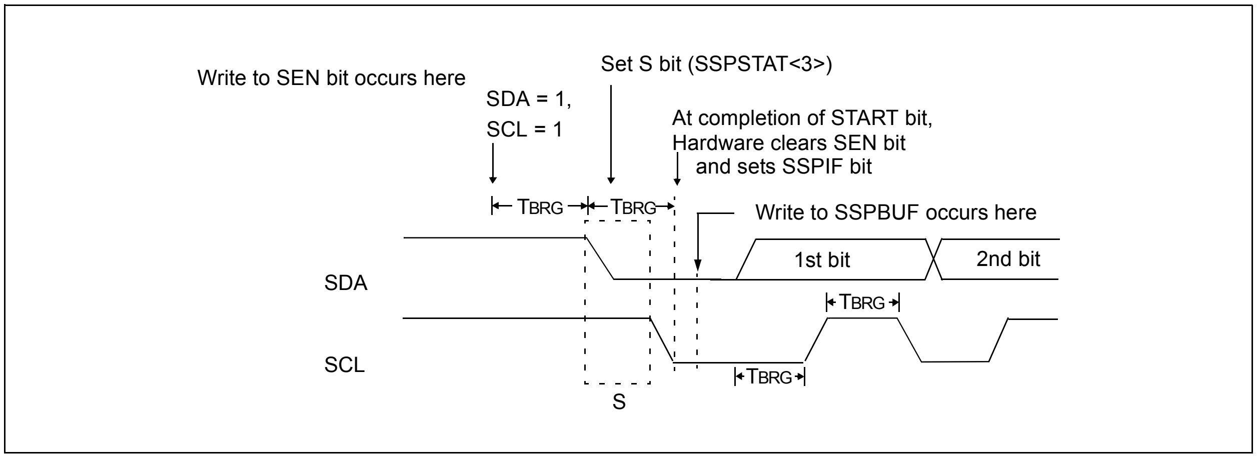
Note:	If at the beginning of the START condition, the SDA and SCL pins are already sampled low, or if during the START condition the SCL line is sampled low before the SDA line is driven low, a bus collision occurs, the Bus Collision Interrupt Flag, BCLIF is set, the START condition is aborted, and the I^{2}C module is reset into its IDLE state.

16.4.8.1 WCOL Status Flag

If the user writes the SSPBUF when a START sequence is in progress, the 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 16-19: FIRST START BIT TIMING



16.4.9 I^{2}C MASTER MODE REPEATED START CONDITION TIMING

A Repeated START condition occurs when the RSEN bit (SSPCON2<1>) is programmed high and the I^{2}C 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 SCL is sampled high, the baud rate generator is reloaded 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<1>) will be automatically cleared and the baud rate generator will not be 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<3>) 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.

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".

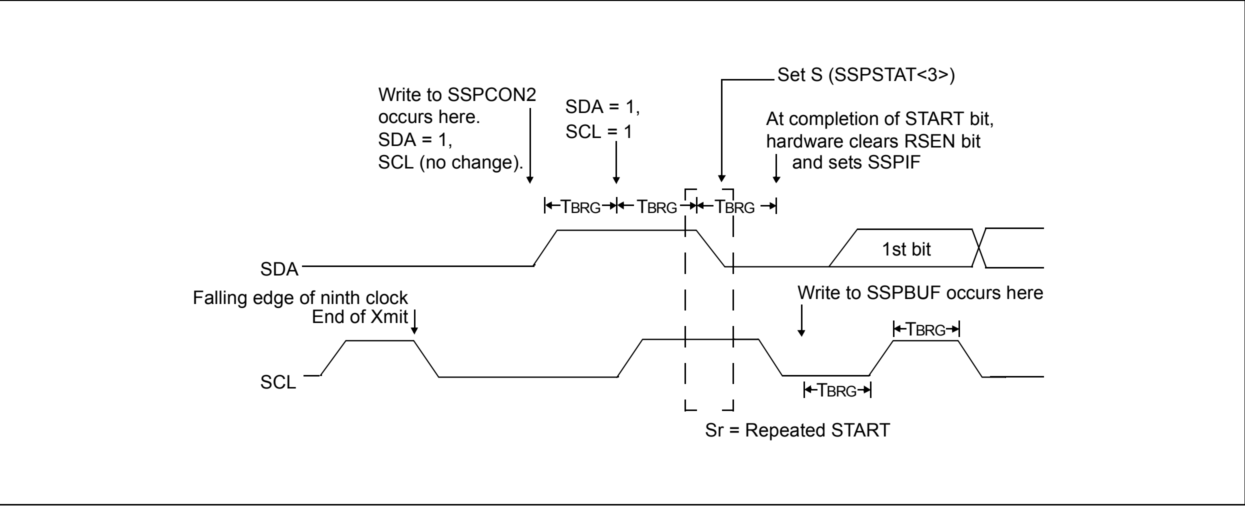
Immediately following the SSPIF bit getting set, the user may write the SSPBUF with the 7-bit address in 7-bit mode, or the default first address in 10-bit mode. After the first eight bits are transmitted and an ACK is received, the user may then transmit an additional eight bits of address (10-bit mode) or eight bits of data (7-bit mode).

16.4.9.1 WCOL Status Flag

If the user writes the SSPBUF when a Repeated START sequence is in progress, the 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 16-20: REPEAT START CONDITION WAVEFORM


Write to SSPCON2 occurs here. SDA = 1, SCL (no change).
SDA = 1, SCL = 1
Set S (SSPSTAT<3>)
At completion of START bit, hardware clears RSEN bit and sets SSPIF
TBRG
TBRG
TBRG
SDA
1st bit
Falling edge of ninth clock End of Xmit
Write to SSPBUF occurs here
TBRG
SCL
TBRG
Sr = Repeated START

16.4.10 I^{2}C 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 the 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). SCL is held low for one baud rate generator rollover count (TBRG). Data should be valid before SCL is released high (see data setup time specification parameter 107). 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 flag is cleared and the master releases SDA. This allows the slave device being addressed to respond with an $\overline{ACK}$ bit during the ninth bit time, if an address match occurred, 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 SCL low and SDA unchanged (Figure 16-21).

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<6>). 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.

16.4.10.1 BF Status Flag

In Transmit mode, the BF bit (SSPSTAT<0>) is set when the CPU writes to SSPBUF and is cleared when all 8 bits are shifted out.

16.4.10.2 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), the WCOL is set and the contents of the buffer are unchanged (the write doesn't occur).

WCOL must be cleared in software.

16.4.10.3 ACKSTAT Status Flag

In Transmit mode, the ACKSTAT bit (SSPCON2<6>) 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.

16.4.11 I^{2}C MASTER MODE RECEPTION

Master mode reception is enabled by programming the receive enable bit, RCEN (SSPCON2<3>).

Note:	In the MSSP module, the RCEN bit must be set after the ACK sequence 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<4>).

16.4.11.1 BF Status Flag

In receive operation, 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.

16.4.11.2 SSPOV Status Flag

In receive operation, 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.

16.4.11.3 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), the WCOL bit is set and the contents of the buffer are unchanged (the write doesn't occur).

FIGURE 16-21: I^{2}C MASTER MODE WAVEFORM (TRANSMISSION, 7 OR 10-BIT ADDRESS)

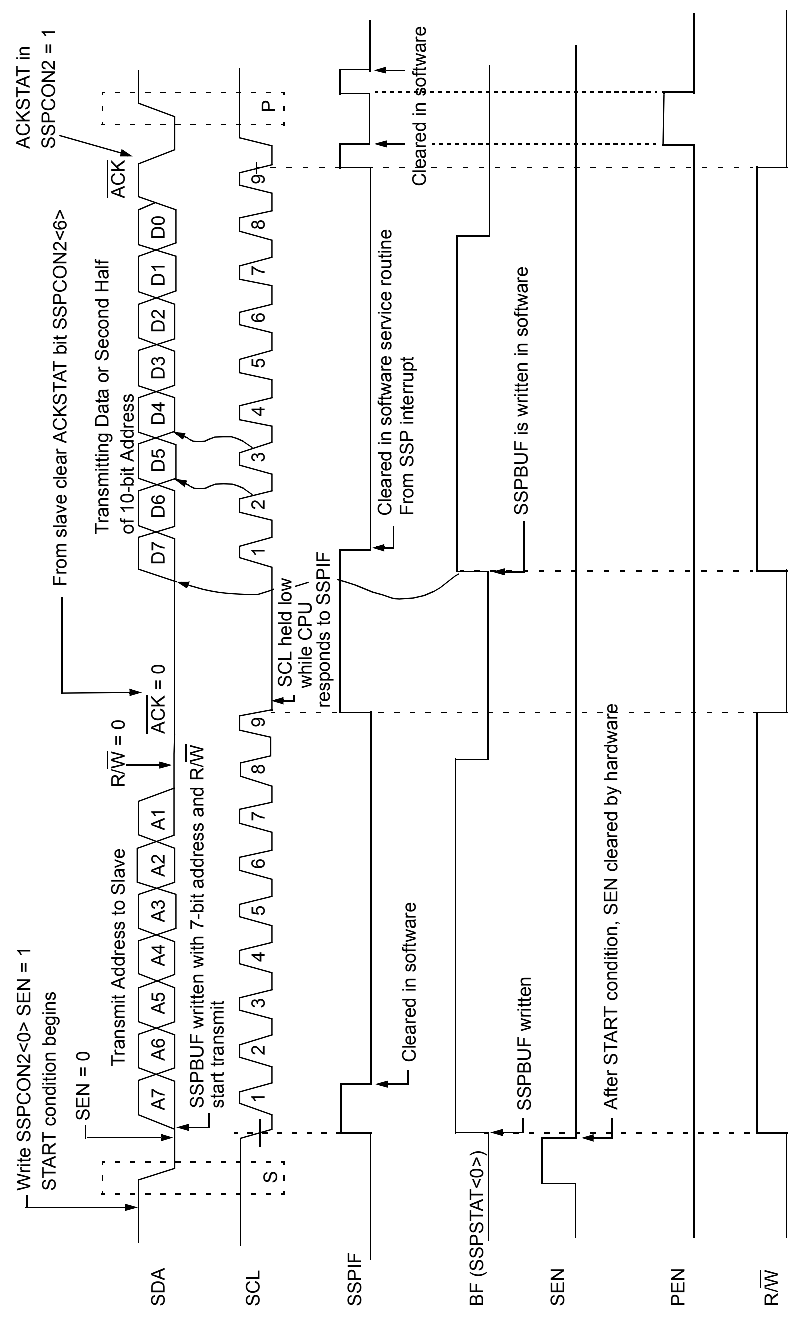
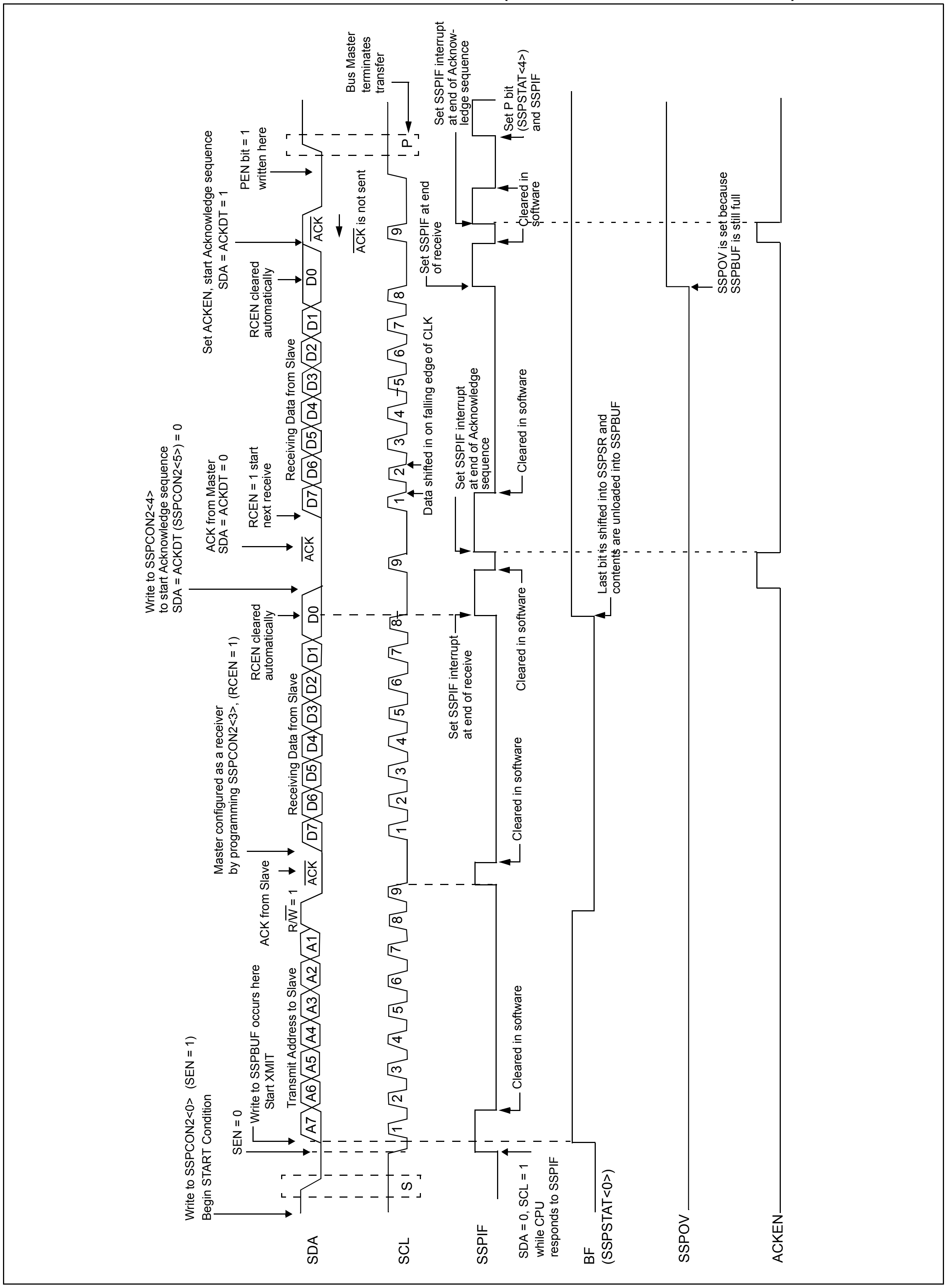


FIGURE 16-22: I²C MASTER MODE WAVEFORM (RECEPTION, 7-BIT ADDRESS)


SDA
SCL
SSPIF
SDA = 0, SCL = 1 while CPU responds to SSPIF
BF (SSPSTAT<0>)
SSPOV
ACKEN
Write to SSPCON2<0> (SEN = 1) Begin START Condition
SEN = 0
Write to SSPBUF occurs here Start XMIT
Transmit Address to Slave
ACK from Slave
R/W = 1
Master configured as a receiver by programming SSPCON2<3>, (RCEN = 1)
Receiving Data from Slave
RCEN cleared automatically
Write to SSPCON2<4> to start Acknowledge sequence SDA = ACKDT (SSPCON2<5>) = 0
ACK from Master SDA = ACKDT = 0
RCEN = 1 start next receive
Receiving Data from Slave
RCEN cleared automatically
Set ACKEN, start Acknowledge sequence SDA = ACKDT = 1
PEN bit = 1 written here
ACK is not sent
Bus Master terminates transfer
S
P
A7 A6 A5 A4 A3 A2 A1
ACK
D7 D6 D5 D4 D3 D2 D1 D0
ACK
D7 D6 D5 D4 D3 D2 D1 D0
ACK
1 2 3 4 5 6 7 8 9
Cleared in software
Cleared in software
Set SSPIF interrupt at end of receive
Cleared in software
Cleared in software
Set SSPIF interrupt at end of Acknowledge sequence
Data shifted in on falling edge of CLK
Cleared in software
Set SSPIF at end of receive
Set SSPIF interrupt at end of Acknowledge sequence
Cleared in software
Set P bit (SSPSTAT<4>) and SSPIF
Last bit is shifted into SSPSR and contents are unloaded into SSPBUF
SSPOV is set because SSPBUF is still full

16.4.12 ACKNOWLEDGE SEQUENCE TIMING

An Acknowledge sequence is enabled by setting the Acknowledge sequence enable bit, ACKEN (SSPCON2<4>). When this bit is set, the SCL pin is pulled low and the contents of the Acknowledge data bit are 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 16-23).

16.4.12.1 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).

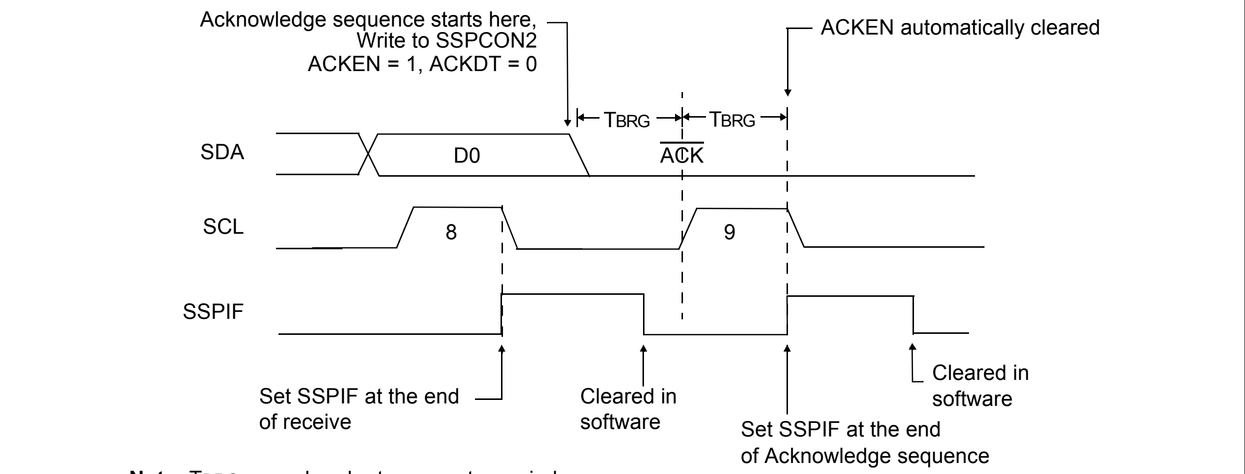
16.4.13 STOP CONDITION TIMING

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<2>). At the end of a receive/transmit the SCL line 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 SCL is high, the P bit (SSPSTAT<4>) is set. A TBRG later, the PEN bit is cleared and the SSPIF bit is set (Figure 16-24).

16.4.13.1 WCOL Status Flag

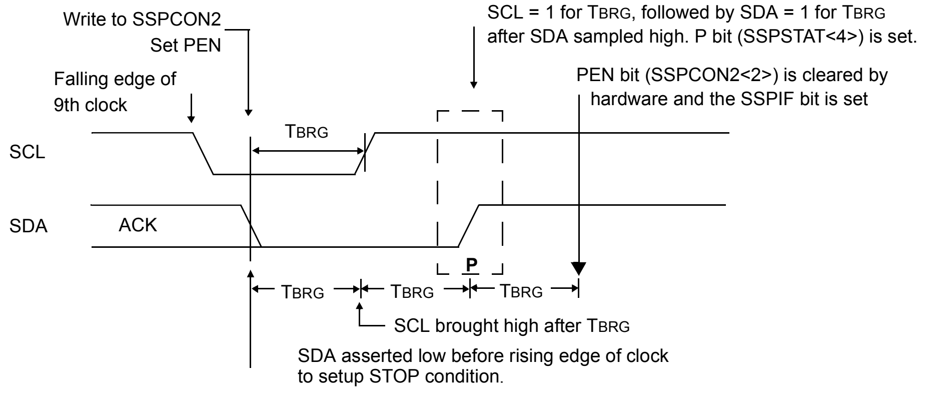
If the user writes the SSPBUF when a STOP sequence is in progress, then the WCOL bit is set and the contents of the buffer are unchanged (the write doesn't occur).

FIGURE 16-23: ACKNOWLEDGE SEQUENCE WAVEFORM


Acknowledge sequence starts here,
Write to SSPCON2
ACKEN = 1, ACKDT = 0
ACKEN automatically cleared
TBRG
TBRG
SDA
D0
ACK
SCL
8
9
SSPIF
Set SSPIF at the end of receive
Cleared in software
Set SSPIF at the end of Acknowledge sequence
Cleared in software

Note: TBRG = one baud rate generator period.

FIGURE 16-24: STOP CONDITION RECEIVE OR TRANSMIT MODE


Write to SSPCON2
Set PEN
Falling edge of 9th clock
SCL = 1 for TBRG, followed by SDA = 1 for TBRG after SDA sampled high. P bit (SSPSTAT<4>) is set.
PEN bit (SSPCON2<2>) is cleared by hardware and the SSPIF bit is set
TBRG
SCL
SDA
ACK
P
TBRG
TBRG
TBRG
SCL brought high after TBRG
SDA asserted low before rising edge of clock to setup STOP condition.

Note: TBRG = one baud rate generator period.

16.4.14 SLEEP OPERATION

While in SLEEP mode, the I^{2}C module can receive addresses or data, and when an address match or complete byte transfer occurs, wake the processor from SLEEP (if the MSSP interrupt is enabled).

16.4.15 EFFECT OF A RESET

A RESET disables the MSSP module and terminates the current transfer.

16.4.16 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^{2}C 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 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

16.4.17 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^{2}C port to its IDLE state (Figure 16-25).

If a transmit was in progress when the bus collision occurred, the transmission is halted, the BF flag is cleared, the SDA and SCL lines are de-asserted, and the SSPBUF can be written to. When the user services the bus collision Interrupt Service Routine, and if the I^{2}C bus is free, the user can resume communication by asserting a START condition.

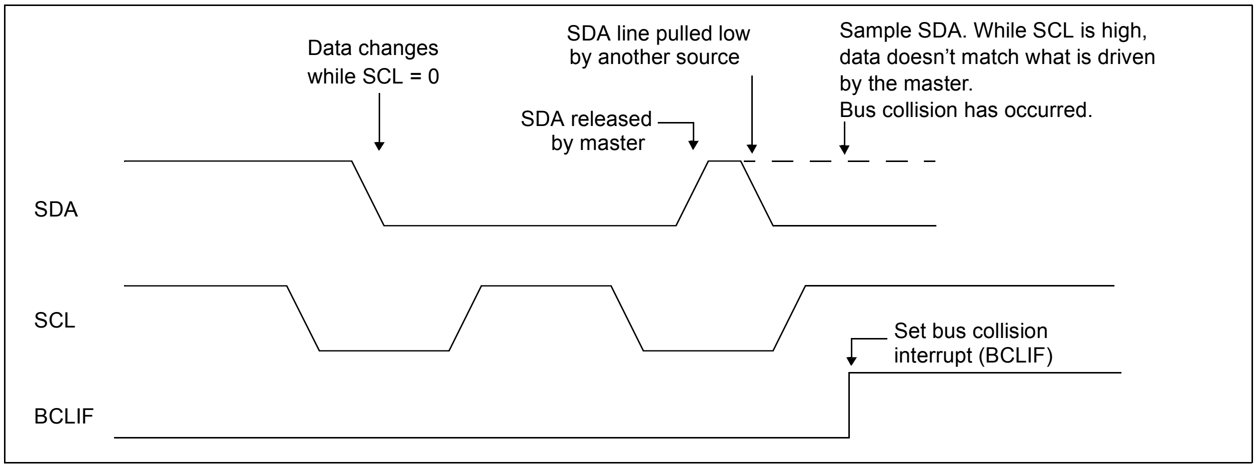
If a START, Repeated START, STOP, or Acknowledge condition was in progress when the bus collision occurred, the condition is aborted, the SDA and SCL lines are de-asserted, and the respective control bits in the SSPCON2 register are cleared. When the user services the bus collision Interrupt Service Routine, and if the I^{2}C bus is free, the user can resume communication by asserting a START condition.

The master will continue to monitor the SDA and SCL pins. 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 the 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^{2}C 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 16-25: BUS COLLISION TIMING FOR TRANSMIT AND ACKNOWLEDGE


Data changes while SCL = 0
SDA line pulled low by another source
Sample SDA. While SCL is high, data doesn't match what is driven by the master.
Bus collision has occurred.
SDA released by master
SDA
SCL
Set bus collision interrupt (BCLIF)
BCLIF

16.4.17.1 Bus Collision During a START Condition

During a START condition, a bus collision occurs if:

a) SDA or SCL are sampled low at the beginning of the START condition (Figure 16-26).

b) SCL is sampled low before SDA is asserted low (Figure 16-27).

During a START condition, both the SDA and the SCL pins are monitored.

If the SDA pin is already low, or the SCL pin is already low, then all of the following occur:

- the START condition is aborted,
- the BCLIF flag is set, and
- the MSSP module is reset to its IDLE state (Figure 16-26).

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 ‘0’. 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 16-28). 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 ‘0’, and during this time, if the SCL pins are 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. If the address is the same, arbitration must be allowed to continue into the data portion, Repeated START or STOP conditions.

FIGURE 16-26: BUS COLLISION DURING START CONDITION (SDA ONLY)

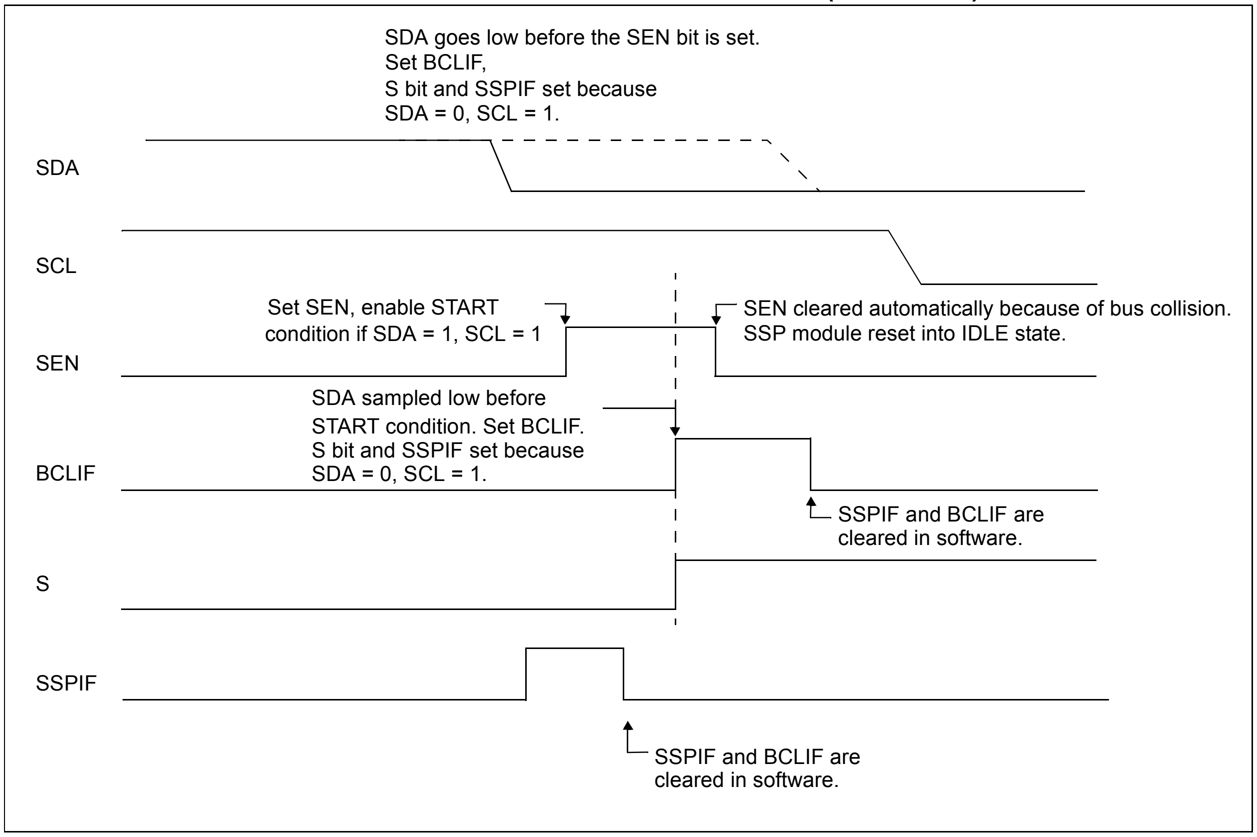

SDA goes low before the SEN bit is set.
Set BCLIF,
S bit and SSPIF set because
SDA = 0, SCL = 1.
SDA
SCL
Set SEN, enable START condition if SDA = 1, SCL = 1
SEN cleared automatically because of bus collision. SSP module reset into IDLE state.
SEN
SDA sampled low before START condition. Set BCLIF. S bit and SSPIF set because SDA = 0, SCL = 1.
BCLIF
SSPIF and BCLIF are cleared in software.
S
SSPIF
SSPIF and BCLIF are cleared in software.

FIGURE 16-27: BUS COLLISION DURING START CONDITION (SCL = 0)

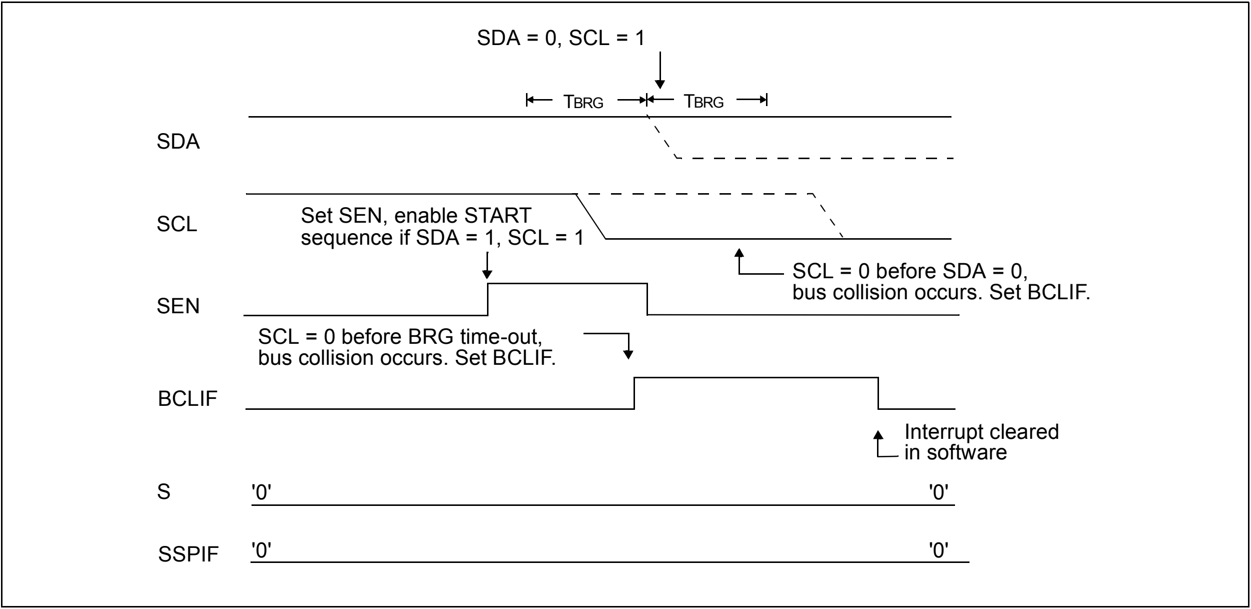
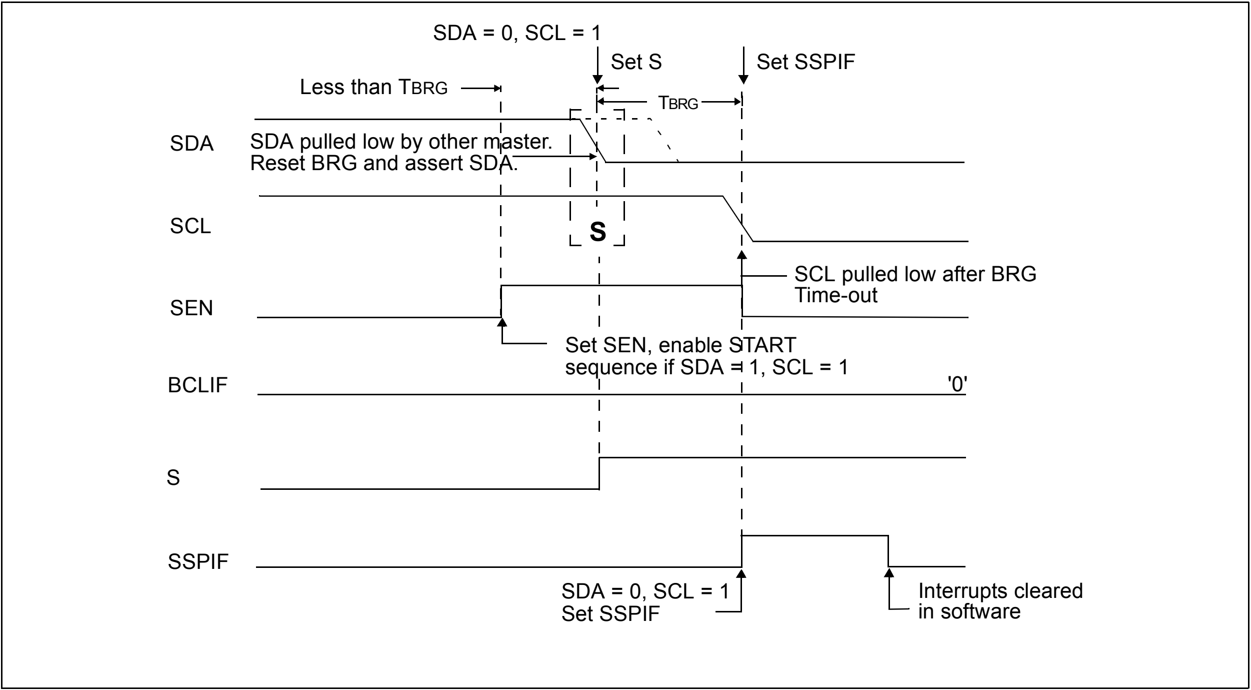

SDA = 0, SCL = 1
TBRG
TBRG
SDA
SCL
Set SEN, enable START sequence if SDA = 1, SCL = 1
SCL = 0 before SDA = 0, bus collision occurs. Set BCLIF.
SEN
SCL = 0 before BRG time-out, bus collision occurs. Set BCLIF.
BCLIF
Interrupt cleared in software
S
'0'
'0'
SSPIF
'0'
'0'

FIGURE 16-28: BRG RESET DUE TO SDA ARBITRATION DURING START CONDITION


SDA = 0, SCL = 1
Set S
Set SSPIF
Less than TBRG
TBRG
SDA
SDA pulled low by other master. Reset BRG and assert SDA.
SCL
S
SCL pulled low after BRG Time-out
SEN
Set SEN, enable START sequence if SDA = 1, SCL = 1
BCLIF
'0'
S
SSPIF
SDA = 0, SCL = 1 Set SSPIF
Interrupts cleared in software

16.4.17.2 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', Figure 16-29). If SDA is sampled high, the BRG is reloaded and begins counting. If SDA 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.

If SCL goes from high to low before the BRG times out and SDA has not already been asserted, a bus collision occurs. In this case, another master is attempting to transmit a data '1' during the Repeated START condition, see Figure 16-30.

If, at the end of the BRG time-out, both SCL and SDA 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 16-29: BUS COLLISION DURING A REPEATED START CONDITION (CASE 1)

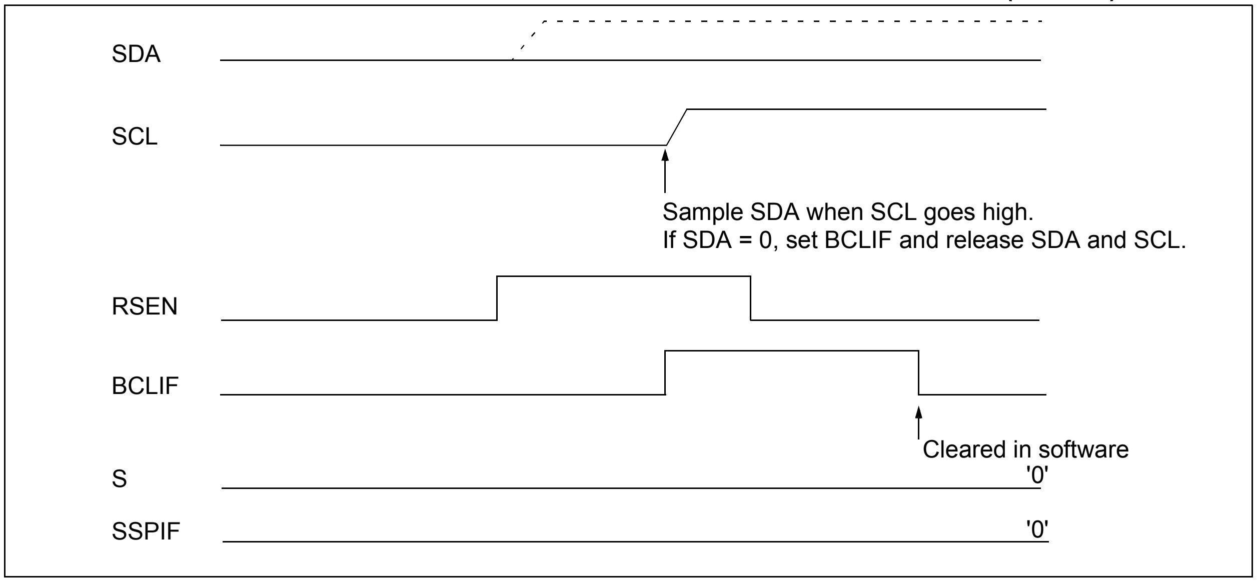
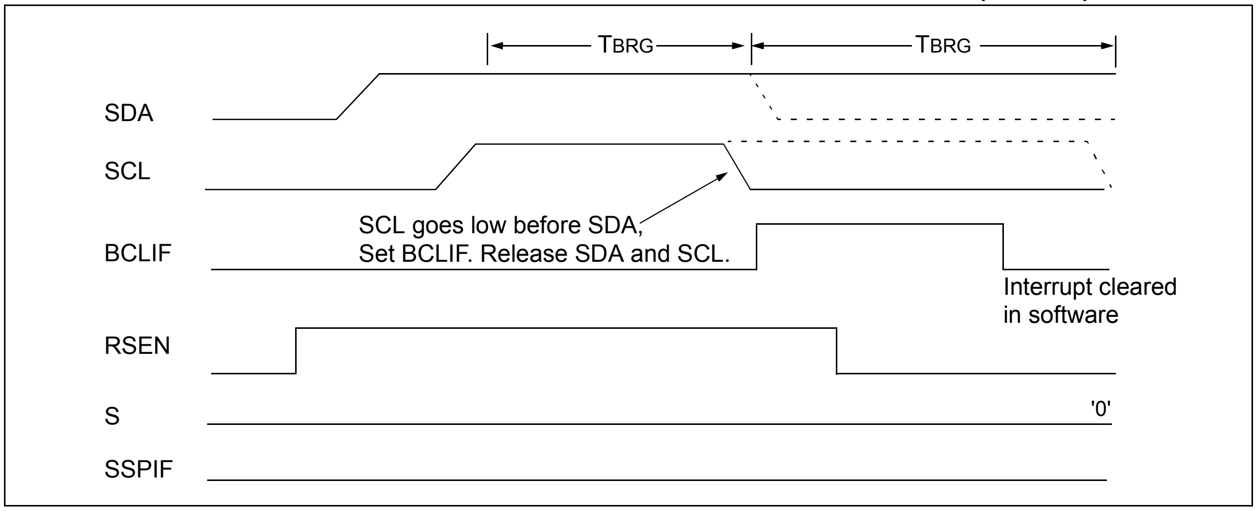

SDA
SCL
Sample SDA when SCL goes high.
If SDA = 0, set BCLIF and release SDA and SCL.
RSEN
BCLIF
Cleared in software
S
'0'
SSPIF
'0'

FIGURE 16-30: BUS COLLISION DURING REPEATED START CONDITION (CASE 2)



16.4.17.3 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 allowed 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 16-31). 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 16-32).

FIGURE 16-31: BUS COLLISION DURING A STOP CONDITION (CASE 1)

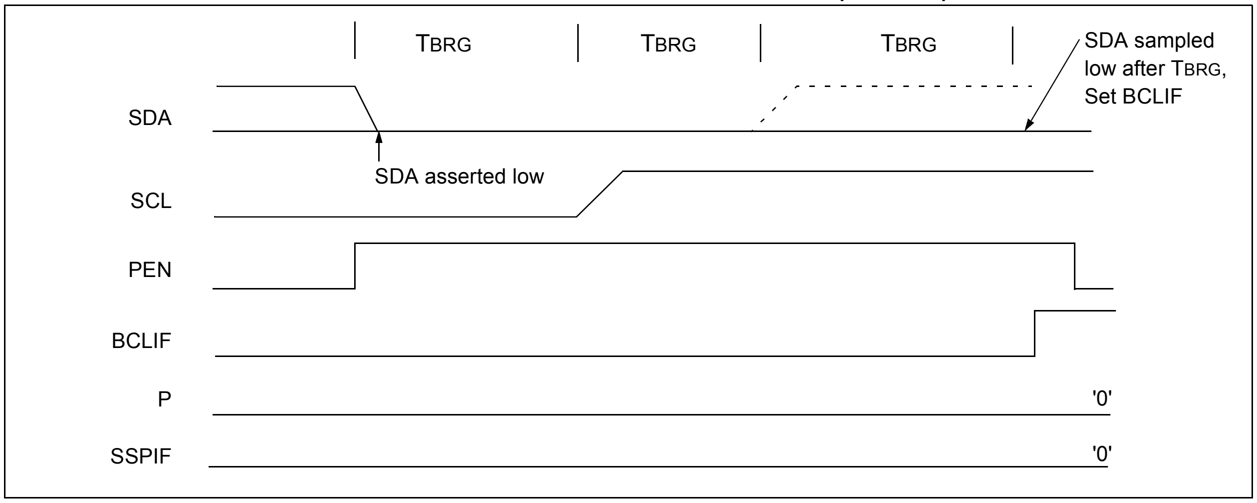
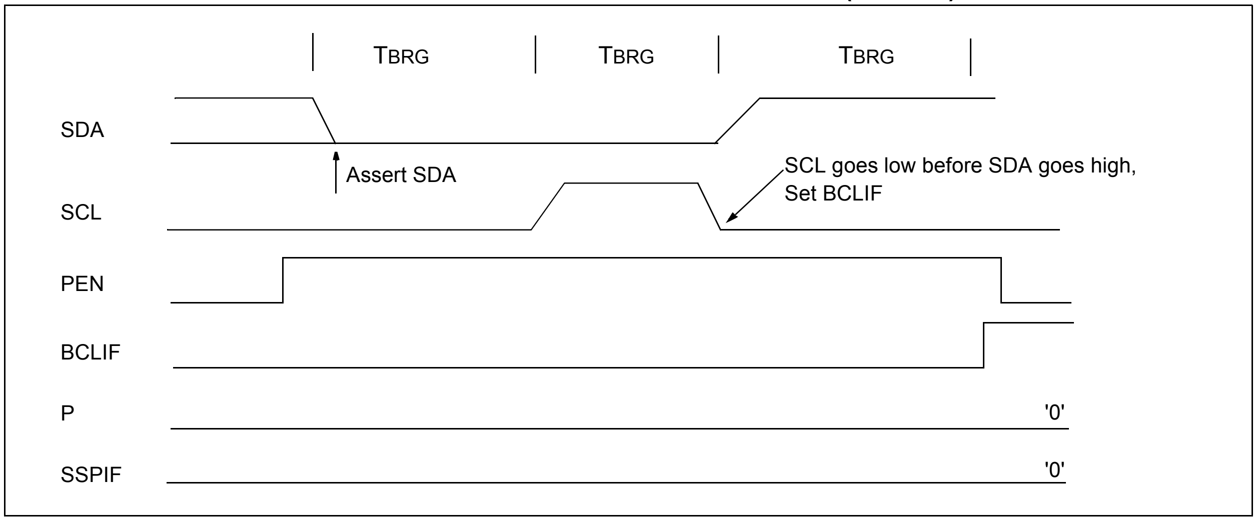


FIGURE 16-32: BUS COLLISION DURING A STOP CONDITION (CASE 2)



NOTES:

17.0 ADDRESSABLE UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER (USART)

The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules. (USART is also known as a Serial Communications Interface or SCI.) The 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 USART can be configured in the following modes:

- Asynchronous (full-duplex)
- Synchronous - Master (half-duplex)
- Synchronous - Slave (half-duplex)

In order to configure pins RC6/TX/CK and RC7/RX/DT as the Universal Synchronous Asynchronous Receiver Transmitter:

- bit SPEN (RCSTA<7>) must be set (= 1),
- bit TRISC<6> must be cleared (= 0), and
- bit TRISC<7> must be set (= 1).

Register 17-1 shows the Transmit Status and Control Register (TXSTA) and Register 17-2 shows the Receive Status and Control Register (RCSTA).

REGISTER 17-1: 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
Asynchronous mode:
Don’t care
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: SREN/CREN overrides TXEN in SYNC mode.

bit 4 **SYNC**: USART Mode Select bit
`1` = Synchronous mode
`0` = Asynchronous mode

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

bit 2 **BRGH**: High Baud Rate Select bit
Asynchronous mode:
`1` = High speed
`0` = Low speed
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
Can be Address/Data bit or a parity bit

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

REGISTER 17-2: 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
Asynchronous mode:
Don’t care
Synchronous mode - Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode - Slave:
Don’t care

bit 4 **CREN**: Continuous Receive Enable bit
Asynchronous mode:
1 = Enables receiver
0 = Disables receiver
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN**: Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and load 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

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
This can be Address/Data bit or a parity bit, and must be calculated by user firmware

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

17.1 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, bit BRGH (TXSTA<2>) also controls the baud rate. In Synchronous mode, bit BRGH is ignored. Table 17-1 shows the formula for computation of the baud rate for different USART modes, which 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 17-1. From this, the error in baud rate can be determined.

Example 17-1 shows the calculation of the baud rate error for the following conditions:

- FOSC = 16 MHz
- Desired Baud Rate = 9600
- BRGH = 0
- SYNC = 0

It may be advantageous to use the high baud rate (BRGH = 1) even for slower baud clocks. This is because the FOSC/(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.

17.1.1 SAMPLING

The data on the RC7/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.

EXAMPLE 17-1: CALCULATING BAUD RATE ERROR

Desired Baud Rate = $F_{OSC} / (64 (X + 1))$

Solving for X:

$$X = ((F_{OSC} / \text{Desired Baud Rate}) / 64) - 1$$
$$X = ((16000000 / 9600) / 64) - 1$$
$$X = [25.042] = 25$$

Calculated Baud Rate = $16000000 / (64 (25 + 1))$ = 9615

$$\text{Error} = \frac{(\text{Calculated Baud Rate} - \text{Desired Baud Rate})}{\text{Desired Baud Rate}} = (9615 - 9600) / 9600 = 0.16\%$$

TABLE 17-1: 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))	N/A

Legend: X = value in SPBRG (0 to 255)

TABLE 17-2: 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 -00x	0000 -00x
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 17-3: BAUD RATES FOR SYNCHRONOUS MODE

BAUD RATE (Kbps)	Fosc = 40 MHz KBAUD	% ERROR	SPBRG value (decimal)	33 MHz KBAUD	% ERROR	SPBRG value (decimal)	25 MHz KBAUD	% ERROR	SPBRG value (decimal)	20 MHz KBAUD	% ERROR	SPBRG value (decimal)
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 KBAUD	% ERROR	SPBRG value (decimal)	10 MHz KBAUD	% ERROR	SPBRG value (decimal)	7.15909 MHz KBAUD	% ERROR	SPBRG value (decimal)	5.0688 MHz KBAUD	% ERROR	SPBRG value (decimal)
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 KBAUD	% ERROR	SPBRG value (decimal)	3.579545 MHz KBAUD	% ERROR	SPBRG value (decimal)	1 MHz KBAUD	% ERROR	SPBRG value (decimal)	32.768 kHz KBAUD	% ERROR	SPBRG value (decimal)
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 17-4: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (Kbps)	Fosc = 40 MHz			33 MHz			25 MHz			20 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
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			10 MHz			7.15909 MHz			5.0688 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
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			3.579545 MHz			1 MHz			32.768 kHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
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 17-5: 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

17.2 USART Asynchronous Mode

In this mode, the USART uses standard non-return-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 bit BRGH (TXSTA<2>). Parity is not supported by the hardware, but can be implemented in software (and stored as the ninth data bit). Asynchronous mode is stopped during SLEEP.

Asynchronous mode is selected by clearing bit SYNC (TXSTA<4>).

The USART Asynchronous module consists of the following important elements:

- Baud Rate Generator
- Sampling Circuit
- Asynchronous Transmitter
- Asynchronous Receiver

17.2.1 USART ASYNCHRONOUS TRANSMITTER

The USART transmitter block diagram is shown in Figure 17-1. The heart of the transmitter is the Transmit (serial) Shift Register (TSR). The shift register obtains its data from the read/write 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 flag bit TXIF (PIR1<4>) is set. This interrupt can be enabled/disabled by setting/clearing enable bit TXIE (PIE1<4>). Flag bit TXIF will be set, regardless of the state of enable bit TXIE and cannot be cleared in software. It will reset only when new data is loaded into the TXREG register. While flag bit TXIF indicated the status of the TXREG register, another bit, TRMT (TXSTA<1>), shows the status of the TSR register. Status bit TRMT 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: Flag bit TXIF is set when enable bit TXEN is set.

To set up an asynchronous transmission:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is desired, set bit BRGH (Section 17.1).
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, set enable bit TXIE.
4. If 9-bit transmission is desired, set transmit bit TX9. Can be used as address/data bit.
5. Enable the transmission by setting bit TXEN, which will also set bit TXIF.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Load data to the TXREG register (starts transmission).

Note: TXIF is not cleared immediately upon loading data into the transmit buffer TXREG. The flag bit becomes valid in the second instruction cycle following the load instruction.

FIGURE 17-1: USART TRANSMIT BLOCK DIAGRAM

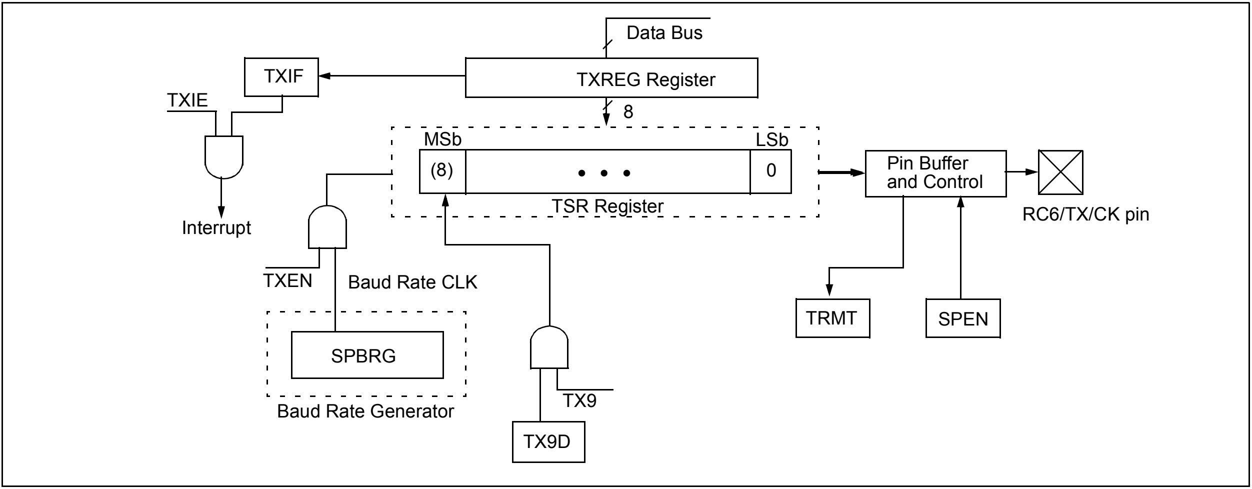


FIGURE 17-2: ASYNCHRONOUS TRANSMISSION

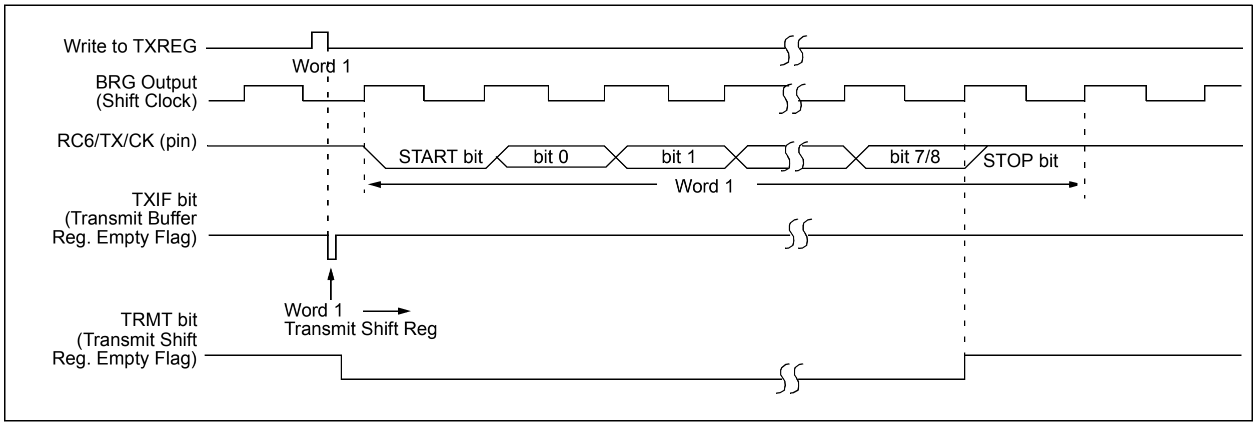
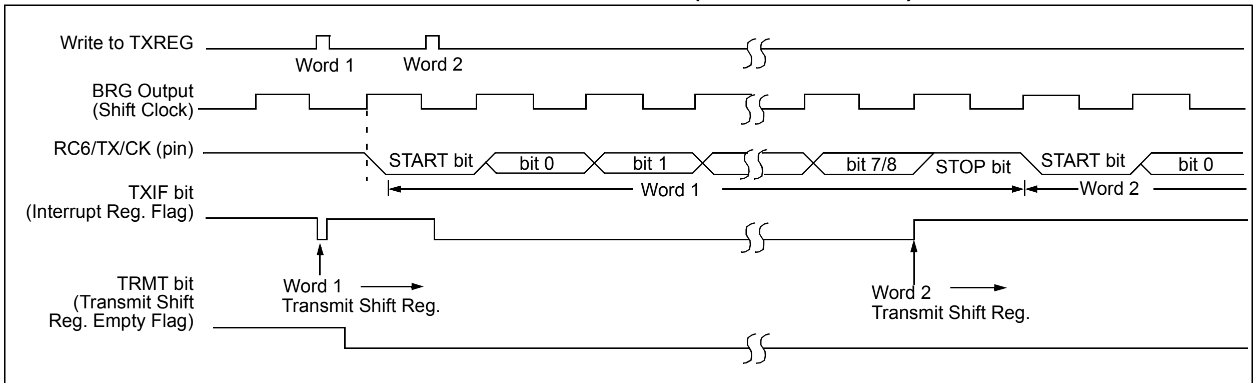

Write to TXREG
Word 1
BRG Output (Shift Clock)
RC6/TX/CK (pin)
START bit
bit 0
bit 1
bit 7/8
STOP bit
Word 1
TXIF bit (Transmit Buffer Reg. Empty Flag)
Word 1 → Transmit Shift Reg
TRMT bit (Transmit Shift Reg. Empty Flag)

FIGURE 17-3: ASYNCHRONOUS TRANSMISSION (BACK TO BACK)


Write to TXREG
Word 1
Word 2
BRG Output (Shift Clock)
RC6/TX/CK (pin)
START bit
bit 0
bit 1
bit 7/8
STOP bit
START bit
bit 0
Word 1
Word 2
TXIF bit (Interrupt Reg. Flag)
Word 1 → Transmit Shift Reg.
Word 2 → Transmit Shift Reg.
TRMT bit (Transmit Shift Reg. Empty Flag)

Note: This timing diagram shows two consecutive transmissions.

TABLE 17-6: 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`
PIR1	PSPIF[(1)]	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF	TMR1IF	`0000 0000`	`0000 0000`
PIE1	PSPIE[(1)]	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	`0000 0000`	`0000 0000`
IPR1	PSPIP[(1)]	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	`0000 0000`	`0000 0000`
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	`0000 -00x`	`0000 -00x`
TXREG	USART Transmit Register								`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 PIC18F2X39 devices; always maintain these bits clear.

17.2.2 USART ASYNCHRONOUS RECEIVER

The receiver block diagram is shown in Figure 17-4. The data is received on the RC7/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.

To set 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 (Section 17.1).
2. Enable the asynchronous serial port by clearing bit SYNC and setting bit SPEN.
3. If interrupts are desired, set enable bit RCIE.
4. If 9-bit reception is desired, set bit RX9.
5. Enable the reception by setting bit CREN.
6. Flag bit RCIF will be set when reception is complete and an interrupt will be generated if enable bit RCIE was set.
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 enable bit CREN.
10. If using interrupts, ensure that the GIE and PEIE bits in the INTCON register (INTCON<7:6>) are set.

17.2.3 SETTING UP 9-BIT MODE WITH ADDRESS DETECT

This mode would typically be used in RS-485 systems. To set up an Asynchronous Reception with Address Detect Enable:

1. Initialize the SPBRG register for the appropriate baud rate. If a high speed baud rate is required, set the BRGH bit.
2. Enable the asynchronous serial port by clearing the SYNC bit and setting the SPEN bit.
3. If interrupts are required, set the RCEN bit and select the desired priority level with the RCIP bit.
4. Set the RX9 bit to enable 9-bit reception.
5. Set the ADDEN bit to enable address detect.
6. Enable reception by setting the CREN bit.
7. The RCIF bit will be set when reception is complete. The interrupt will be acknowledged if the RCIE and GIE bits are set.
8. Read the RCSTA register to determine if any error occurred during reception, as well as read bit 9 of data (if applicable).
9. Read RCREG to determine if the device is being addressed.
10. If any error occurred, clear the CREN bit.
11. If the device has been addressed, clear the ADDEN bit to allow all received data into the receive buffer and interrupt the CPU.

FIGURE 17-4: USART RECEIVE BLOCK DIAGRAM

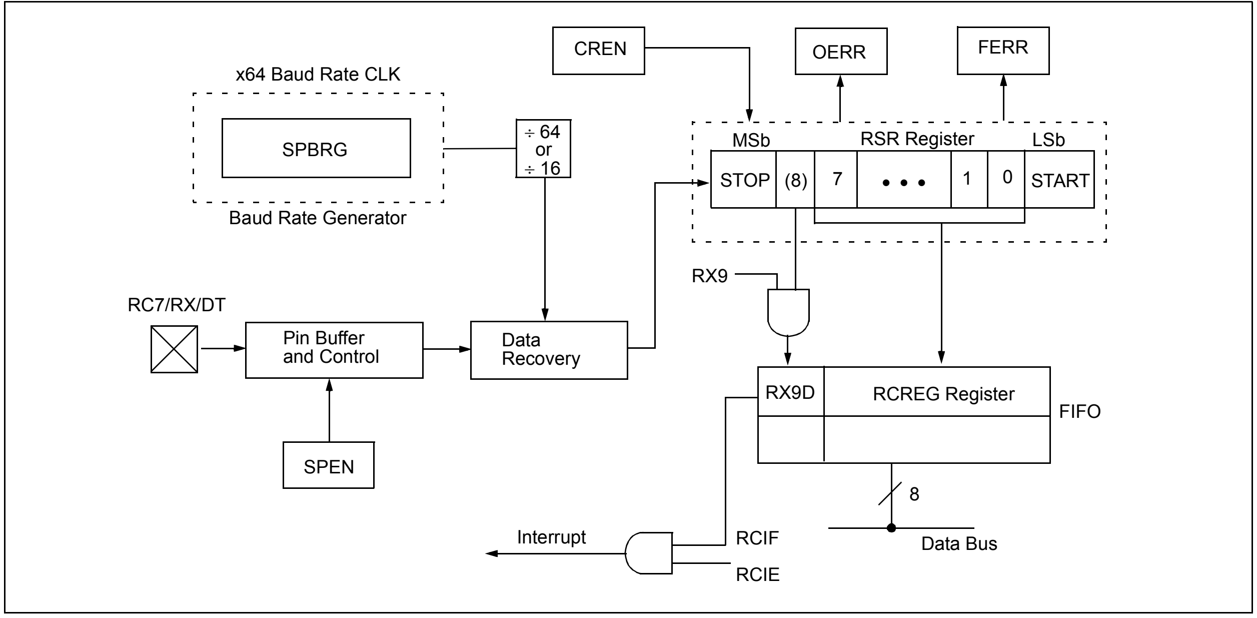
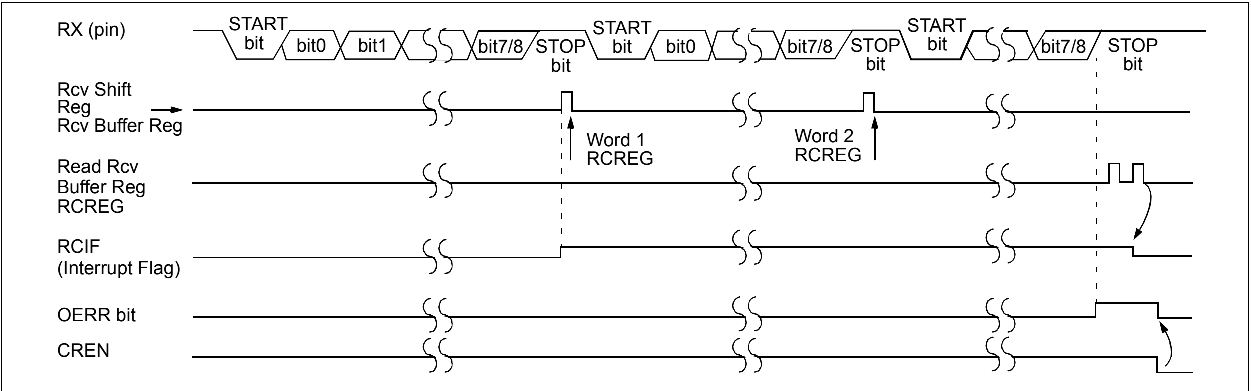


FIGURE 17-5: ASYNCHRONOUS RECEPTION



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.

TABLE 17-7: 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
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	0000 0000	0000 0000
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
RCREG	USART Receive Register								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 PSPIF, PSPIE and PSPIP bits are reserved on the PIC18F2X39 devices; always maintain these bits clear.

17.3 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 bit SYNC (TXSTA<4>). In addition, enable bit SPEN (RCSTA<7>) is set in order to configure the RC6/TX/CK and RC7/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 bit CSRC (TXSTA<7>).

17.3.1 USART SYNCHRONOUS MASTER TRANSMISSION

The USART transmitter block diagram is shown in Figure 17-1. The heart of the transmitter is the Transmit (serial) 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 interrupt bit TXIF (PIR1<4>) is set. The interrupt can be enabled/disabled by setting/clearing enable bit TXIE (PIE1<4>). Flag bit TXIF will be set, regardless of the state of enable bit TXIE, and cannot be cleared in software. It will reset only when new data is loaded into the TXREG register. While flag bit TXIF indicates the status of the TXREG register, another bit TRMT (TXSTA<1>) shows the status of the TSR register. TRMT is a read only bit, which 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.

To set up a Synchronous Master Transmission:

1. Initialize the SPBRG register for the appropriate baud rate (Section 17.1).
2. Enable the synchronous master serial port by setting bits SYNC, SPEN, and CSRC.
3. If interrupts are desired, set enable bit TXIE.
4. If 9-bit transmission is desired, set bit TX9.
5. Enable the transmission by setting bit TXEN.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Start transmission by loading data to the TXREG register.

Note: TXIF is not cleared immediately upon loading data into the transmit buffer TXREG. The flag bit becomes valid in the second instruction cycle following the load instruction.

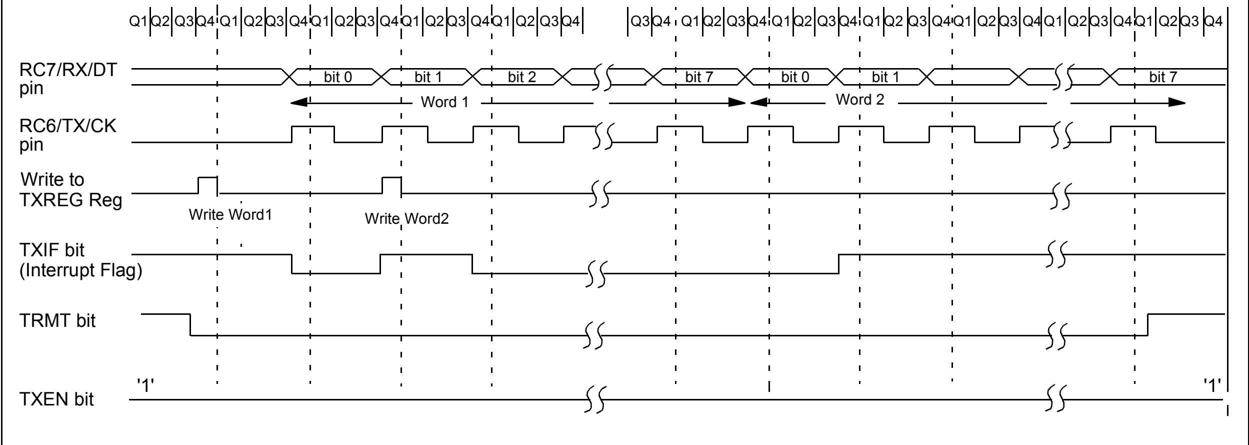
TABLE 17-8: REGISTERS ASSOCIATED WITH SYNCHRONOUS MASTER TRANSMISSION

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
PIR1	PSPIF[(1)]	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE[(1)]	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP[(1)]	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	0000 0000	0000 0000
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
TXREG	USART Transmit Register								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.

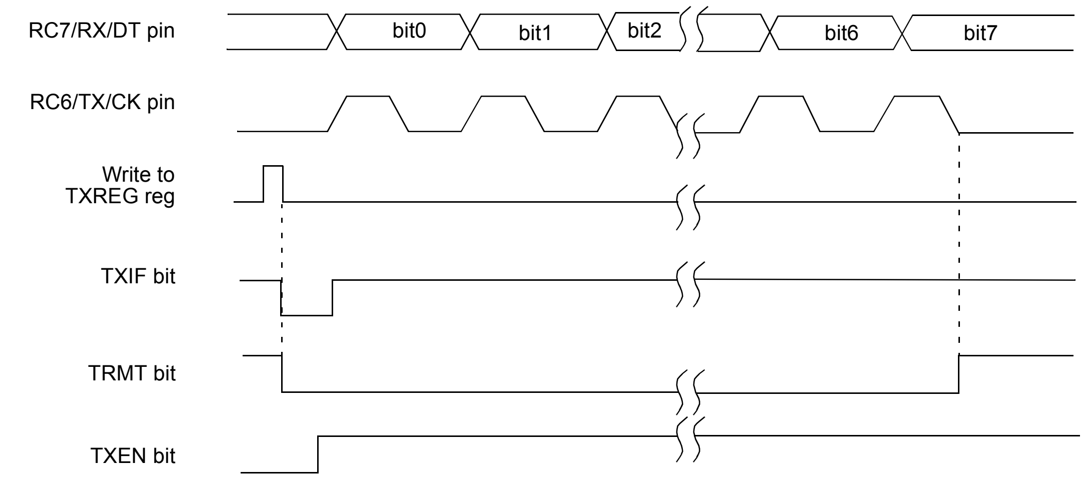
Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18F2X39 devices; always maintain these bits clear.

FIGURE 17-6: SYNCHRONOUS TRANSMISSION


Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4
RC7/RX/DT pin
bit 0
bit 1
bit 2
bit 7
bit 0
bit 1
bit 7
Word 1
Word 2
RC6/TX/CK pin
Write to TXREG Reg
Write Word1
Write Word2
TXIF bit (Interrupt Flag)
TRMT bit
TXEN bit
'1'
'1'

Note: Sync Master mode; SPBRG = '0'. Continuous transmission of two 8-bit words.

FIGURE 17-7: SYNCHRONOUS TRANSMISSION (THROUGH TXEN)



17.3.2 USART SYNCHRONOUS MASTER RECEPTION

Once Synchronous mode is selected, reception is enabled by setting either enable bit SREN (RCSTA<5>), or enable bit CREN (RCSTA<4>). Data is sampled on the RC7/RX/DT pin on the falling edge of the clock. If enable bit SREN is set, only a single word is received. If enable bit CREN is set, the reception is continuous until CREN is cleared. If both bits are set, then CREN takes precedence.

To set up a Synchronous Master Reception:

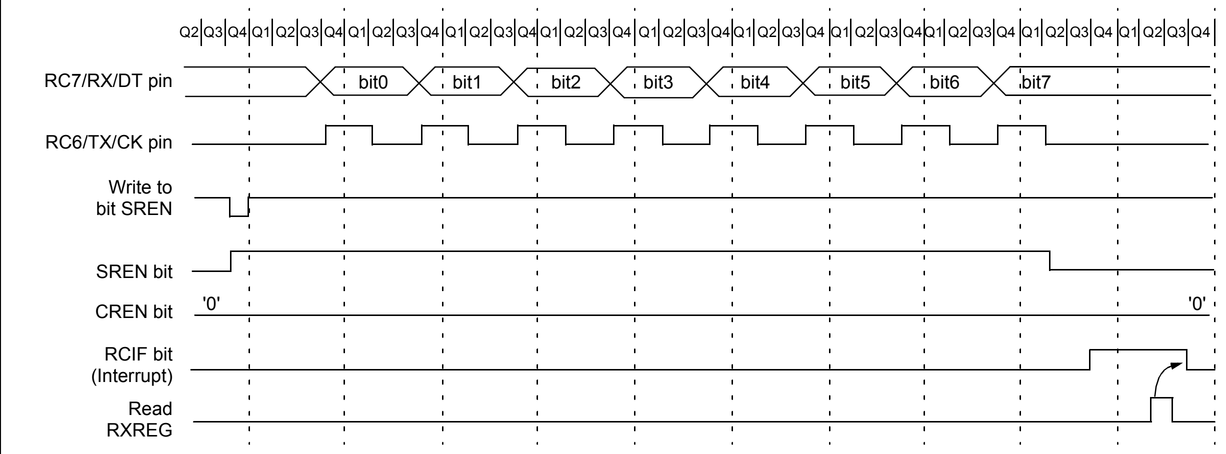
1. Initialize the SPBRG register for the appropriate baud rate (Section 17.1).
2. Enable the synchronous master serial port by setting bits SYNC, SPEN and CSRC.
3. Ensure bits CREN and SREN are clear.
4. If interrupts are desired, set enable bit RCIE.
5. If 9-bit reception is desired, set bit RX9.
6. If a single reception is required, set bit SREN. For continuous reception, set bit CREN.
7. Interrupt flag bit RCIF will be set when reception is complete and an interrupt will be generated if the enable bit RCIE was 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 bit CREN.
11. If using interrupts, ensure that the GIE and PEIE bits in the INTCON register (INTCON<7:6>) are set.

TABLE 17-9: 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
PIR1	PSPIF[(1)]	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE[(1)]	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP[(1)]	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	0000 0000	0000 0000
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
RCREG	USART Receive Register								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 PSPIF, PSPIE and PSPIP bits are reserved on the PIC18F2X39 devices; always maintain these bits clear.

FIGURE 17-8: SYNCHRONOUS RECEPTION (MASTER MODE, SREN)


Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4
RC7/RX/DT pin
bit0
bit1
bit2
bit3
bit4
bit5
bit6
bit7
RC6/TX/CK pin
Write to bit SREN
SREN bit
CREN bit
'0'
'0'
RCIF bit (Interrupt)
Read RXREG

Note: Timing diagram demonstrates Sync Master mode with bit SREN = '1' and bit BRGH = '0'.

17.4 USART Synchronous Slave Mode

Synchronous Slave mode differs from the Master mode in the fact that the shift clock is supplied externally at the RC6/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 bit CSRC (TXSTA<7>).

17.4.1 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) Flag bit TXIF 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 flag bit TXIF will now be set.
e) If enable bit TXIE is set, the interrupt will wake the chip from SLEEP. If the global interrupt is enabled, the program will branch to the interrupt vector.

To set up a Synchronous Slave Transmission:

1. Enable the synchronous slave serial port by setting bits SYNC and SPEN and clearing bit CSRC.
2. Clear bits CREN and SREN.
3. If interrupts are desired, set enable bit TXIE.
4. If 9-bit transmission is desired, set bit TX9.
5. Enable the transmission by setting enable bit TXEN.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Start transmission by loading data to the TXREG register.
8. If using interrupts, ensure that the GIE and PEIE bits in the INTCON register (INTCON<7:6>) are set.

TABLE 17-10: 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
PIR1	PSPIF[(1)]	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE[(1)]	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP[(1)]	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	0000 0000	0000 0000
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
TXREG	USART Transmit Register								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 PSPIF, PSPIE and PSPIP bits are reserved on the PIC18F2X39 devices; always maintain these bits clear.

17.4.2 USART SYNCHRONOUS SLAVE RECEPTION

The operation of the Synchronous Master and Slave modes is identical, except in the case of the SLEEP mode and bit SREN, which is a “don't care” in Slave mode.

If receive is enabled by setting bit CREN 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 enable bit RCIE bit is set, the interrupt generated will wake the chip from SLEEP. If the global interrupt is enabled, the program will branch to the interrupt vector.

To set up a Synchronous Slave Reception:

1. Enable the synchronous master serial port by setting bits SYNC and SPEN and clearing bit CSRC.
2. If interrupts are desired, set enable bit RCIE.
3. If 9-bit reception is desired, set bit RX9.
4. To enable reception, set enable bit CREN.
5. Flag bit RCIF will be set when reception is complete. An interrupt will be generated if enable bit RCIE was 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 bit CREN.
9. If using interrupts, ensure that the GIE and PEIE bits in the INTCON register (INTCON<7:6>) are set.

TABLE 17-11: 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`
PIR1	PSPIF[1]	ADIF	RCIF	TXIF	SSPIF	—	TMR2IF	TMR1IF	`0000 0000`	`0000 0000`
PIE1	PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	`0000 0000`	`0000 0000`
IPR1	PSPIP[1]	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	`0000 0000`	`0000 0000`
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	`0000 -00x`	`0000 -00x`
RCREG	USART Receive Register								`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 PSPIF, PSPIE and PSPIP bits are reserved on the PIC18F2X39 devices; always maintain these bits clear.

18.0 COMPATIBLE 10-BIT ANALOG-TO-DIGITAL CONVERTER (A/D) MODULE

The Analog-to-Digital (A/D) converter module has five inputs for the PIC18F2X39 devices and eight for the PIC18F4X39 devices. This module has the ADCON0 and ADCON1 register definitions that are compatible with the mid-range A/D module.

The A/D allows conversion of an analog input signal to a corresponding 10-bit digital number.

The A/D module has four registers:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register 0 (ADCON0)
- A/D Control Register 1 (ADCON1)

The ADCON0 register, shown in Register 18-1, controls the operation of the A/D module. The ADCON1 register, shown in Register 18-2, configures the functions of the port pins.

REGISTER 18-1: ADCON0 REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
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)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits
000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)(1)
110 = Channel 6 (AN6)(1)
111 = Channel 7 (AN7)(1)

Note 1: These channels are unimplemented on PIC18F2X39 devices. 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 the A/D conversion, which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

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

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	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

REGISTER 18-2: 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 bit
`1` = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
`0` = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
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)

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C / R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8 / 0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7 / 1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5 / 0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4 / 1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3 / 0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	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	VDD	VSS	6 / 0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	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	VDD	VSS	1 / 0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1 / 2

A = Analog input D = Digital I/O
C/R = # 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	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: On any device RESET, the port pins that are multiplexed with analog functions (ANx) are forced to be an analog input.

The analog reference voltage is software selectable to either the device's positive and negative supply voltage (VDD and VSS), or the voltage level on the RA3/AN3/VREF+ pin and RA2/AN2/VREF- pin.

The A/D converter has a unique feature of being able to operate while the device is in SLEEP mode. To operate in SLEEP, the A/D conversion clock must be derived from the A/D's internal RC oscillator.

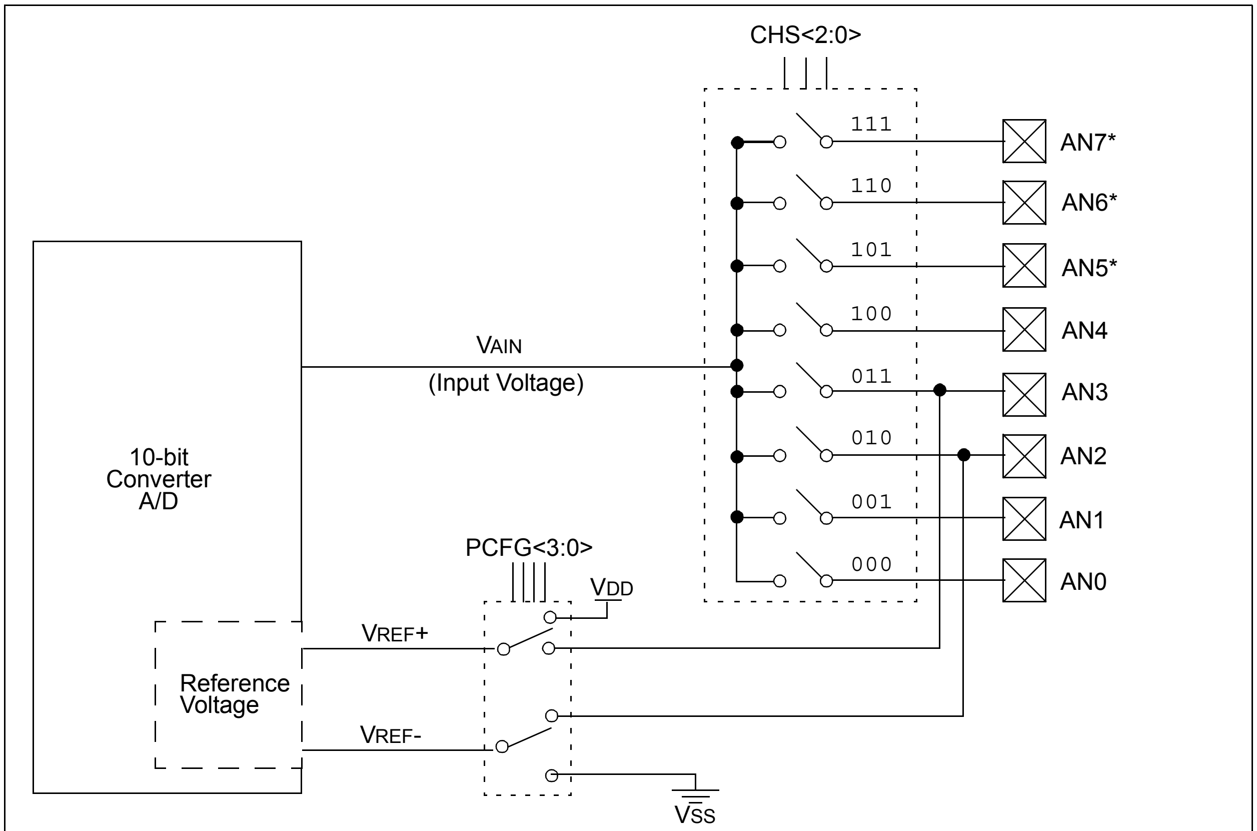
The output of the sample and hold is the input into the converter, which generates the result via successive approximation.

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.

Each port pin associated with the A/D converter can be configured as an analog input (RA3 can also be a voltage reference) or as a digital I/O.

The ADRESH and ADRESL registers contain the result of the A/D conversion. When the A/D conversion is complete, the result is loaded into the ADRESH/ADRESL registers, the GO/$\overline{\text{DONE}}$ bit (ADCON0<2>) is cleared, and A/D interrupt flag bit, ADIF is set. The block diagram of the A/D module is shown in Figure 18-1.

FIGURE 18-1: A/D BLOCK DIAGRAM



* These channels are implemented only on the PIC18F4X39 devices.

The value that is in the ADRESH/ADRESL registers is not modified for a Power-on Reset. The ADRESH/ADRESL registers will contain unknown data after a Power-on Reset.

After the A/D module has been configured as desired, the selected channel must be acquired before the conversion is started. The analog input channels must have their corresponding TRIS bits selected as an input. To determine acquisition time, see Section 18.1. 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 ADIF bit
   - Set ADIE bit
   - Set GIE bit
   - Set PEIE bit
3. Wait the required acquisition time.
4. Start conversion:
   - Set GO/$\overline{\text{DONE}}$ bit (ADCON0)
5. Wait for A/D conversion to complete, by either:
   - Polling for the GO/$\overline{\text{DONE}}$ bit to be cleared (interrupts disabled)

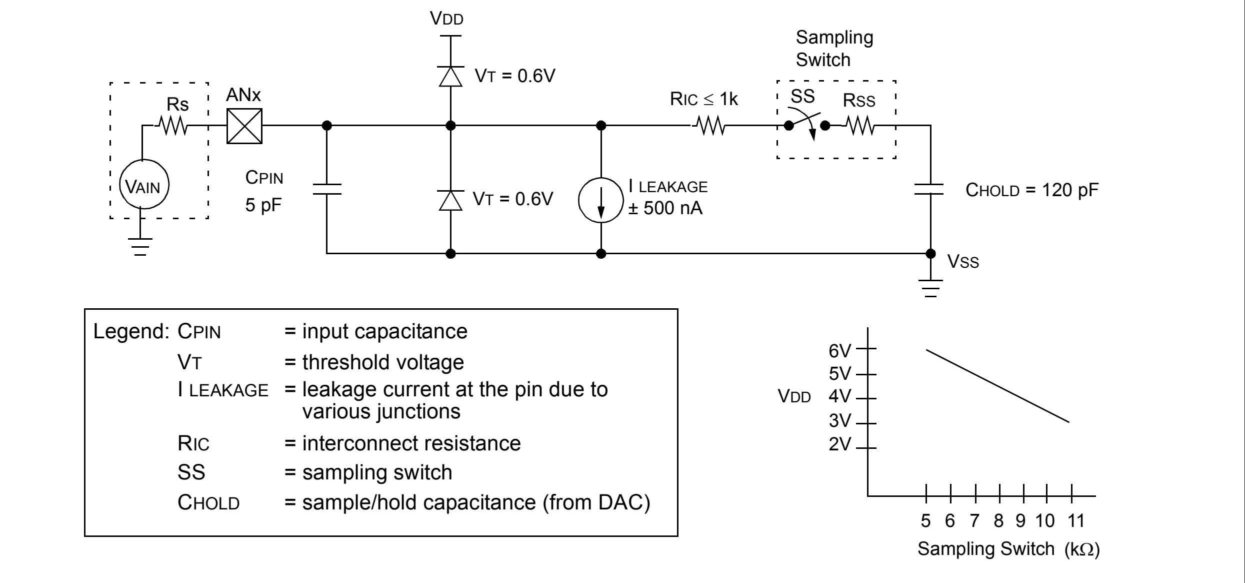
   OR
   - Waiting for the A/D interrupt
6. Read A/D Result registers (ADRESH/ADRESL); clear bit ADIF if required.
7. For next conversion, go to step 1 or step 2 as required. The A/D conversion time per bit is defined as T_{AD}. A minimum wait of 2 T_{AD} is required before the next acquisition starts.

18.1 A/D Acquisition Requirements

For the A/D converter to meet its specified accuracy, the charge holding capacitor (C_{HOLD}) must be allowed to fully charge to the input channel voltage level. The analog input model is shown in Figure 18-2. The source impedance (R_S) and the internal sampling switch (R_{SS}) impedance directly affect the time required to charge the capacitor C_{HOLD}. The sampling switch (R_{SS}) impedance varies over the device voltage (V_{DD}). 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**. After the analog input channel is selected (changed), this acquisition must be done before the conversion can be started.

Note:	When the conversion is started, the holding capacitor is disconnected from the input pin.

FIGURE 18-2: ANALOG INPUT MODEL



To calculate the minimum acquisition time, Equation 18-1 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.

EQUATION 18-1: ACQUISITION TIME

$$T_{ACQ} = \text{Amplifier Settling Time} + \text{Holding Capacitor Charging Time} + \text{Temperature Coefficient}$$
$$= T_{AMP} + T_C + T_{COFF}$$

EQUATION 18-2: A/D MINIMUM CHARGING TIME

$$V_{HOLD} = (V_{REF} - (V_{REF}/2048)) \bullet (1 - e^{(-T_C/C_{HOLD}(R_{IC} + R_{SS} + R_S))})$$
or
$$T_C = -(120\text{ pF})(1\text{ k}\Omega + R_{SS} + R_S)\ \ln(1/2048)$$

Example 18-1 shows the calculation of the minimum required acquisition time, T_{ACQ}. This calculation is based on the following application system assumptions:

• C_{HOLD}	=	120 pF
• Rs	=	2.5 kΩ
• Conversion Error	≤	1/2 LSb
• V_{DD}	=	5V → Rss = 7 kΩ
• Temperature	=	50°C (system max.)
• V_{HOLD}	=	0V @ time = 0

EXAMPLE 18-1: CALCULATING THE MINIMUM REQUIRED ACQUISITION TIME

$T_{ACQ} = T_{AMP} + T_C + T_{COFF}$

Temperature coefficient is only required for temperatures > 25°C.

$T_{ACQ} = 2\ \mu s + T_C + [(\text{Temp} - 25°C)(0.05\ \mu s/°C)]$

$$\begin{aligned}T_C = & -C_{HOLD}\ (R_{IC} + R_{SS} + R_S)\ \ln(1/2048)\\ & -120\text{ pF}\ (1\text{ k}\Omega + 7\text{ k}\Omega + 2.5\text{ k}\Omega)\ \ln(0.0004883)\\ & -120\text{ pF}\ (10.5\text{ k}\Omega)\ \ln(0.0004883)\\ & -1.26\ \mu s\ (-7.6246)\\ & 9.61\ \mu s\end{aligned}$$

$$\begin{aligned}T_{ACQ} = & 2\ \mu s + 9.61\ \mu s + [(50°C - 25°C)(0.05\ \mu s/°C)]\\ & 11.61\ \mu s + 1.25\ \mu s\\ & 12.86\ \mu s\end{aligned}$$

18.2 Selecting the A/D Conversion Clock

The A/D conversion time per bit is defined as TAD. The A/D conversion requires 12 TAD per 10-bit conversion. The source of the A/D conversion clock is software selectable. The seven possible options for TAD are:

- 2 TOSC
- 4 TOSC
- 8 TOSC
- 16 TOSC
- 32 TOSC
- 64 TOSC
- Internal A/D module RC oscillator (2-6 μs)

For correct A/D conversions, the A/D conversion clock (TAD) must be selected to ensure a minimum TAD time of 1.6 μs.

Table 18-1 shows the resultant TAD times derived from the device operating frequencies and the A/D clock source selected.

18.3 Configuring Analog Port Pins

The ADCON1, TRISA and TRISE 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 (VOH or VOL) will be converted.

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, all pins configured as analog input channels 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 AN4:AN0 pins) may cause the input buffer to consume current that is out of the device's specification.

TABLE 18-1: TAD vs. DEVICE OPERATING FREQUENCIES

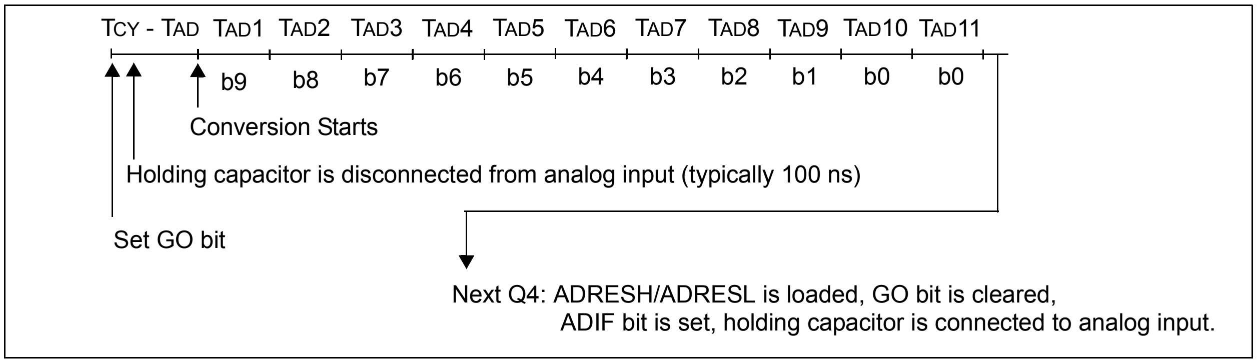
AD Clock Source (TAD)		Maximum Device Frequency	
Operation	**ADCS2:ADCS0**	**PIC18FXX39**	**PIC18LFXX39**
2 TOSC	`000`	1.25 MHz	666 kHz
4 TOSC	`100`	2.50 MHz	1.33 MHz
8 TOSC	`001`	5.00 MHz	2.67 MHz
16 TOSC	`101`	10.00 MHz	5.33 MHz
32 TOSC	`010`	20.00 MHz	10.67 MHz
64 TOSC	`110`	40.00 MHz	21.33 MHz
RC	`011`	—	—

18.4 A/D Conversions

Figure 18-3 shows the operation of the A/D converter after the GO bit has been set. 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 2 TAD wait is required before the next acquisition is started. After this 2 TAD wait, acquisition on the selected channel is automatically started. The GO/$\overline{\text{DONE}}$ bit can then be set to start the conversion.

Note:	The GO/$\overline{\text{DONE}}$ bit should **NOT** be set in the same instruction that turns on the A/D.

FIGURE 18-3: A/D CONVERSION TAD CYCLES


TCY - TAD
TAD1
TAD2
TAD3
TAD4
TAD5
TAD6
TAD7
TAD8
TAD9
TAD10
TAD11
b9
b8
b7
b6
b5
b4
b3
b2
b1
b0
b0
Conversion Starts
Holding capacitor is disconnected from analog input (typically 100 ns)
Set GO bit
Next Q4: ADRESH/ADRESL is loaded, GO bit is cleared,
ADIF bit is set, holding capacitor is connected to analog input.

18.4.1 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 18-4 shows the operation of the A/D result justification. The extra bits are loaded with ‘0’s. When an A/D result will not overwrite these locations (A/D disable), these registers may be used as two general purpose 8-bit registers.

FIGURE 18-4: A/D RESULT JUSTIFICATION

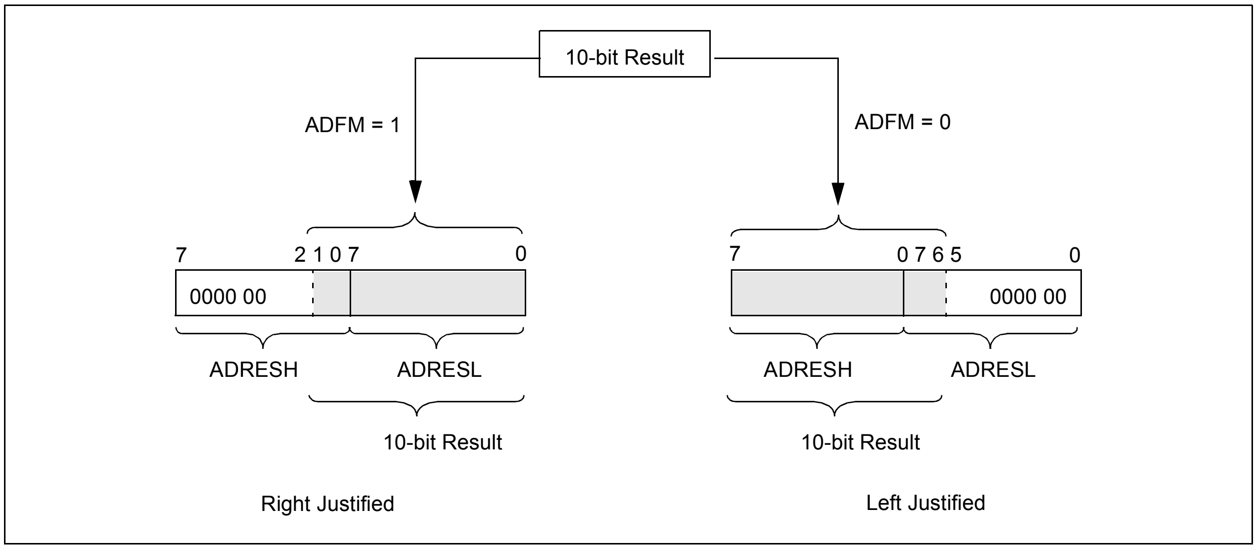


TABLE 18-2: SUMMARY OF A/D REGISTERS

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	—	TMR2IF	TMR1IF	`0000 0000`	`0000 0000`
PIE1	PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	—	TMR2IE	TMR1IE	`0000 0000`	`0000 0000`
IPR1	PSPIP[1]	ADIP	RCIP	TXIP	SSPIP	—	TMR2IP	TMR1IP	`0000 0000`	`0000 0000`
PIR2	—	—	—	EEIF	BCLIF	LVDIF	TMR3IF	—	`---0 0000`	`---0 0000`
PIE2	—	—	—	EEIE	BCLIE	LVDIE	TMR3IE	—	`---0 0000`	`---0 0000`
IPR2	—	—	—	EEIP	BCLIP	LVDIP	TMR3IP	—	`---1 1111`	`---1 0000`
ADRESH	A/D Result Register								`xxxx xxxx`	`uuuu uuuu`
ADRESL	A/D Result Register								`xxxx xxxx`	`uuuu uuuu`
ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON	`0000 00-0`	`0000 00-0`
ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	`---- -000`	`---- -000`
PORTA	—	RA6	RA5	RA4	RA3	RA2	RA1	RA0	`--0x 0000`	`--0u 0000`
TRISA	—	PORTA Data Direction Register							`--11 1111`	`--11 1111`
PORTE	—	—	—	—	—	RE2	RE1	RE0	`---- -000`	`---- -000`
LATE	—	—	—	—	—	LATE2	LATE1	LATE0	`---- -xxx`	`---- -uuu`
TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			`0000 -111`	`0000 -111`

Legend: `x` = unknown, `u` = unchanged, - = unimplemented, read as '0'. Shaded cells are not used for A/D conversion.
Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on the PIC18F2X39 devices; always maintain these bits clear.

19.0 LOW VOLTAGE DETECT

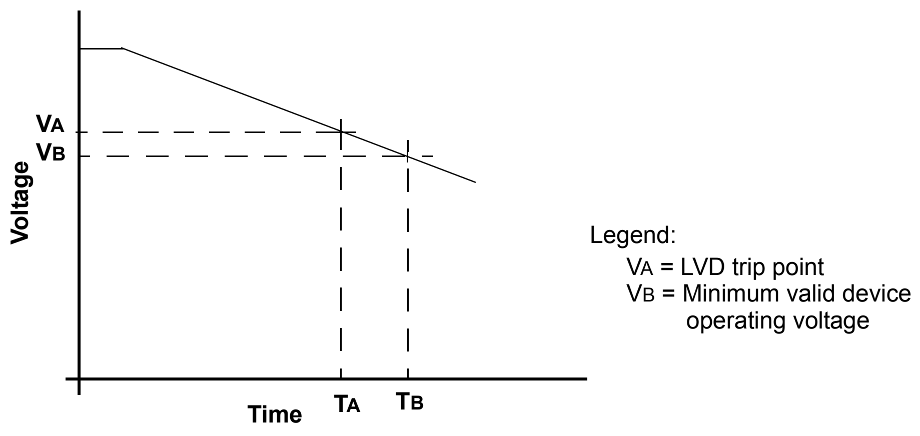
In many applications, the ability to determine if the device voltage (V_{DD}) is below a specified voltage level is a desirable feature. A window of operation for the application can be created, where the application software can do "housekeeping tasks" before the device voltage exits the valid operating range. This can be done using the Low Voltage Detect module.

This module is a software programmable circuitry, where a device voltage trip point can be specified. When the voltage of the device becomes lower then 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 19-1 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 the time, until the device voltage is no longer in valid operating range, to shutdown the system. Voltage point V_B is the minimum valid operating voltage specification. This occurs at time T_B. The difference $T_B - T_A$ is the total time for shutdown.

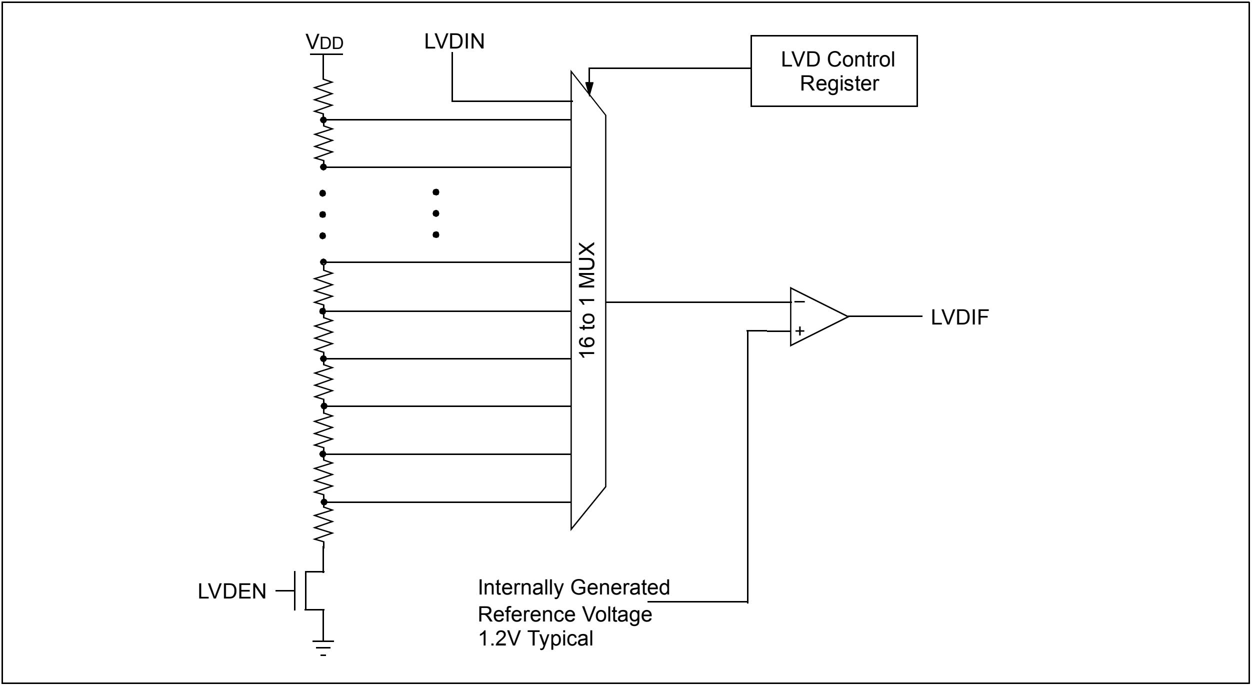
FIGURE 19-1: TYPICAL LOW VOLTAGE DETECT APPLICATION



The block diagram for the LVD module is shown in Figure 19-2. 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 than), the LVDIF bit is set.

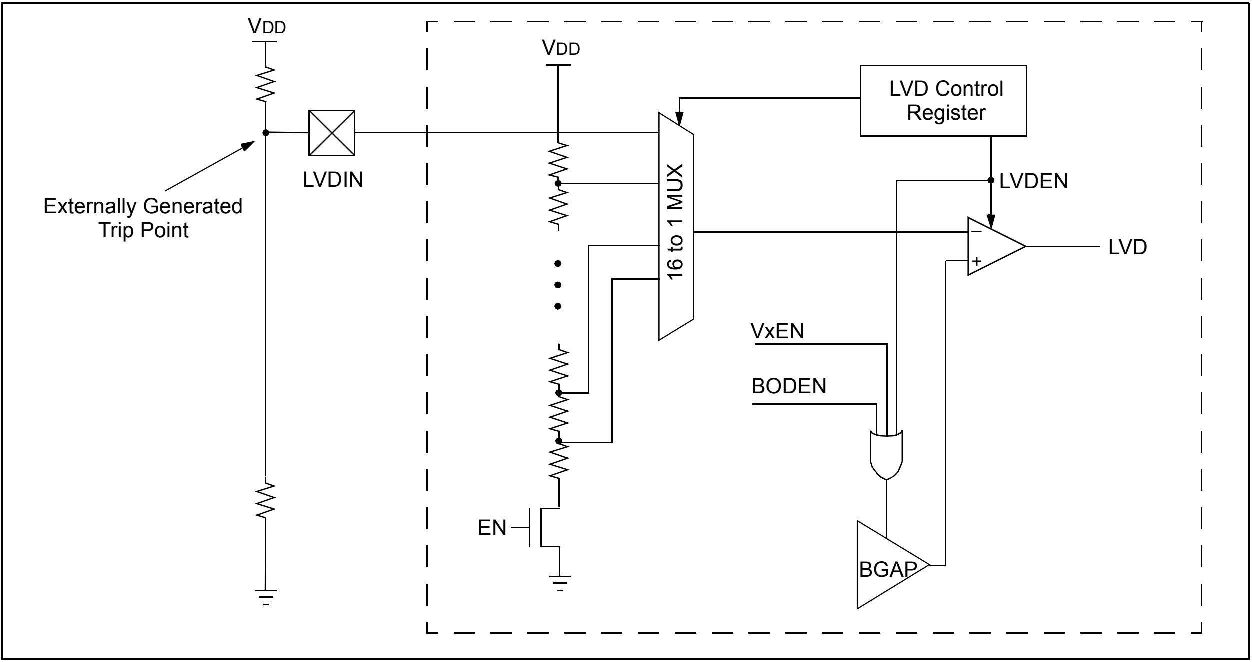
Each node in the resistor divider represents a "trip point" voltage. The "trip point" voltage is the minimum supply voltage level at which the device can operate before the LVD module asserts an interrupt. When the supply voltage is equal to the trip point, the voltage tapped off of the resistor array is equal to the 1.2V internal reference voltage generated by the 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 19-2). The trip point is selected by programming the LVDL3:LVDL0 bits (LVDCON<3:0>).

FIGURE 19-2: LOW VOLTAGE DETECT (LVD) BLOCK DIAGRAM



The LVD module has an additional feature that allows the user to supply the trip voltage to the module from an external source. This mode is enabled when bits LVDL3:LVDL0 are set to ‘`1111`’. In this state, the comparator input is multiplexed from the external input pin, LVDIN (Figure 19-3). This gives users flexibility, because it allows them to configure the Low Voltage Detect interrupt to occur at any voltage in the valid operating range.

FIGURE 19-3: LOW VOLTAGE DETECT (LVD) WITH EXTERNAL INPUT BLOCK DIAGRAM



19.1 Control Register

The Low Voltage Detect Control register controls the operation of the Low Voltage Detect circuitry.

REGISTER 19-1: 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 the 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
`1111` = External analog input is used (input comes from the LVDIN pin)
`1110` = 4.5V - 4.77V
`1101` = 4.2V - 4.45V
`1100` = 4.0V - 4.24V
`1011` = 3.8V - 4.03V
`1010` = 3.6V - 3.82V
`1001` = 3.5V - 3.71V
`1000` = 3.3V - 3.50V
`0111` = 3.0V - 3.18V
`0110` = 2.8V - 2.97V
`0101` = 2.7V - 2.86V
`0100` = 2.5V - 2.65V
`0011` = 2.4V - 2.54V
`0010` = 2.2V - 2.33V
`0001` = 2.0V - 2.12V
`0000` = Reserved

Note: LVDL3:LVDL0 modes, which result in a trip point below the valid operating voltage of the device, are not tested.

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

19.2 Operation

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.

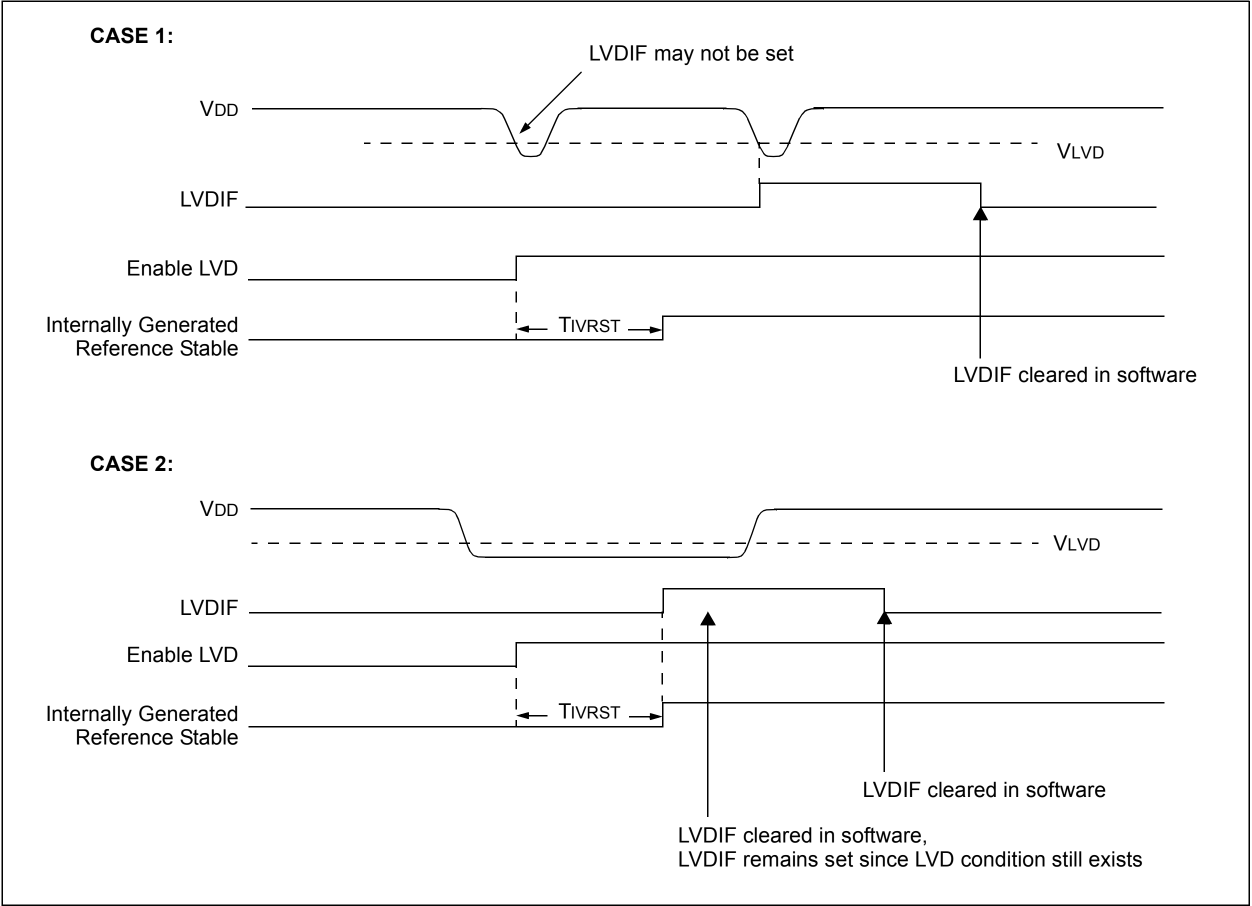
Each time that the LVD module is enabled, the circuitry requires some time to stabilize. After the circuitry has stabilized, all status flags may be cleared. The module will then indicate the proper state of the system.

The following steps are needed to set up 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 until the LVD module has stabilized (clear the LVDIF bit).
6. Enable the LVD interrupt (set the LVDIE and the GIE bits).

Figure 19-4 shows typical waveforms that the LVD module may be used to detect.

FIGURE 19-4: LOW VOLTAGE DETECT WAVEFORMS


CASE 1:
LVDIF may not be set
VDD
VLVD
LVDIF
Enable LVD
Internally Generated Reference Stable
TIVRST
LVDIF cleared in software
CASE 2:
VDD
VLVD
LVDIF
Enable LVD
Internally Generated Reference Stable
TIVRST
LVDIF cleared in software
LVDIF cleared in software,
LVDIF remains set since LVD condition still exists

19.2.1 REFERENCE VOLTAGE SET POINT

The Internal Reference Voltage of the LVD module may be used by other internal circuitry (the Programmable Brown-out Reset). If these circuits are disabled (lower current consumption), the reference voltage circuit requires a time to become stable before a low voltage condition can be reliably detected. This time is invariant of system clock speed. This start-up 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 waveform in Figure 19-4.

19.2.2 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.

19.3 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.

19.4 Effects of a RESET

A device RESET forces all registers to their RESET state. This forces the LVD module to be turned off.

NOTES:

20.0 SPECIAL FEATURES OF THE CPU

There are several features intended to maximize system reliability, minimize cost through elimination of external components, provide power saving Operating modes and offer code protection. These are:

- OSC Selection
- RESET
  - Power-on Reset (POR)
  - Power-up Timer (PWRT)
  - Oscillator Start-up Timer (OST)
  - Brown-out Reset (BOR)
- Interrupts
- Watchdog Timer (WDT)
- SLEEP
- Code Protection
- ID Locations
- In-Circuit Serial Programming

All PIC18FXX39 devices have a Watchdog Timer, which is permanently enabled via the configuration bits, or software controlled. It runs off its own RC oscillator for added reliability. There are two timers that offer necessary delays on power-up. One is the Oscillator Start-up Timer (OST), intended to keep the chip in RESET until the crystal oscillator is stable. The other is the Power-up Timer (PWRT), which provides a fixed delay on power-up only, designed to keep the part in RESET while the power supply stabilizes. With these two timers on-chip, most applications need no external RESET circuitry.

SLEEP mode is designed to offer a very low current Power-down mode. The user can wake-up from SLEEP through external RESET, Watchdog Timer Wake-up or through an interrupt. Several oscillator options are also made available to allow the part to fit the application. The RC oscillator option saves system cost, while the LP crystal option saves power. A set of configuration bits are used to select various options.

20.1 Configuration Bits

The configuration bits can be programmed (read as '0'), or left unprogrammed (read as '1'), to select various device configurations. These bits are mapped starting at program memory location 300000h.

The user will note that address 300000h is beyond the user program memory space. In fact, it belongs to the configuration memory space (300000h - 3FFFFFh), which can only be accessed using Table Reads and Table Writes.

Programming the configuration registers is done in a manner similar to programming the FLASH memory (see Section 5.5.1). The only difference is the configuration registers are written a byte at a time. The sequence of events for programming configuration registers is:

1. Load table pointer with address of configuration register being written.
2. Write a single byte using the `TBLWT` instruction.
3. Set EEPGD to point to program memory, set the CFGS bit to access configuration registers, and set WREN to enable byte writes.
4. Disable interrupts.
5. Write 55h to EECON2.
6. Write AAh to EECON2.
7. Set the WR bit. This will begin the write cycle.
8. CPU will stall for duration of write (approximately 2 ms using internal timer).
9. Execute a `NOP`.
10. Re-enable interrupts.

TABLE 20-1: CONFIGURATION BITS AND DEVICE IDS

File Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Default/ Unprogrammed Value
300001h	CONFIG1H	—	—	—[(1)]	—	—	FOSC2	FOSC1	FOSC0	`--1- -010`
300002h	CONFIG2L	—	—	—	—	BORV1	BORV0	BOREN	$\overline{\text{PWRTEN}}$	`---- 1111`
300003h	CONFIG2H	—	—	—	—	WDTPS2	WDTPS1	WDTPS0	WDTEN	`---- 1111`
300005h	CONFIG3H	—	—	—	—	—	—	—	—[(1)]	`---- ---1`
300006h	CONFIG4L	$\overline{\text{DEBUG}}$	—	—	—	—	LVP	—	STVREN	`1--- -1-1`
300008h	CONFIG5L	—	—	—	—	—[(1)]	CP2	CP1	CP0	`---- 1111`
300009h	CONFIG5H	CPD	CPB	—	—	—	—	—	—	`11-- ----`
30000Ah	CONFIG6L	—	—	—	—	—[(1)]	WRT2	WRT1	WRT0	`---- 1111`
30000Bh	CONFIG6H	WRTD	WRTB	WRTC	—	—	—	—	—	`111- ----`
30000Ch	CONFIG7L	—	—	—	—	—[(1)]	EBTR2	EBTR1	EBTR0	`---- 1111`
30000Dh	CONFIG7H	—	EBTRB	—	—	—	—	—	—	`-1-- ----`
3FFFFEh	DEVID1	DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0	**(2)**
3FFFFFh	DEVID2	DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3	`0000 0100`

Legend: `x` = unknown, `u` = unchanged, - = unimplemented. Shaded cells are unimplemented, read as ‘0’.

Note 1: Unimplemented, but reserved; maintain this bit set.

2: See Register 20-11 for DEVID1 values.

REGISTER 20-1: CONFIG1H: CONFIGURATION REGISTER 1 HIGH (BYTE ADDRESS 300001h)

U-0	U-0	U-1	U-0	U-0	R/P-0	R/P-1	R/P-0
—	—	—	—	—	FOSC2	FOSC1	FOSC0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as ‘0’

bit 5 **Unimplemented and reserved:** Maintain as ‘1’

bit 4-3 **Unimplemented:** Read as ‘0’

bit 2-0 **FOSC2:FOSC0**: Oscillator Selection bits

`111` = Reserved
`110` = HS oscillator with PLL enabled; clock frequency = (4 x FOSC)
`101` = EC oscillator w/ OSC2 configured as RA6
`100` = EC oscillator w/ OSC2 configured as divide-by-4 clock output
`011` = Reserved
`010` = HS oscillator
`001` = Reserved
`000` = Reserved

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-2: CONFIG2L: CONFIGURATION REGISTER 2 LOW (BYTE ADDRESS 300002h)

U-0	U-0	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	—	BORV1	BORV0	BOREN	$\overline{\text{PWRTEN}}$
bit 7							bit 0

bit 7-4 **Unimplemented:** Read as ‘0’

bit 3-2 **BORV1:BORV0:** Brown-out Reset Voltage bits
`11` = V_{BOR} set to 2.5V
`10` = V_{BOR} set to 2.7V
`01` = V_{BOR} set to 4.2V
`00` = V_{BOR} set to 4.5V

bit 1 **BOREN:** Brown-out Reset Enable bit
`1` = Brown-out Reset enabled
`0` = Brown-out Reset disabled

bit 0 **$\overline{\text{PWRTEN}}$:** Power-up Timer Enable bit
`1` = PWRT disabled
`0` = PWRT enabled

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-3: CONFIG2H: CONFIGURATION REGISTER 2 HIGH (BYTE ADDRESS 300003h)

U-0	U-0	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
—	—	—	—	WDTPS2	WDTPS1	WDTPS0	WDTEN
bit 7							bit 0

bit 7-4 **Unimplemented:** Read as ‘0’

bit 3-1 **WDTPS2:WDTPS0:** Watchdog Timer Postscale Select bits
`111` = 1:128
`110` = 1:64
`101` = 1:32
`100` = 1:16
`011` = 1:8
`010` = 1:4
`001` = 1:2
`000` = 1:1

bit 0 **WDTEN:** Watchdog Timer Enable bit
`1` = WDT enabled
`0` = WDT disabled (control is placed on the SWDTEN bit)

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-4: CONFIG4L: CONFIGURATION REGISTER 4 LOW (BYTE ADDRESS 300006h)

R/P-1	U-0	U-0	U-0	U-0	R/P-1	U-0	R/P-1
$\overline{\text{DEBUG}}$	—	—	—	—	LVP	—	STVREN
bit 7							bit 0

bit 7	**$\overline{\text{DEBUG}}$:** Background Debugger Enable bit `1` = Background Debugger disabled. RB6 and RB7 configured as general purpose I/O pins. `0` = Background Debugger enabled. RB6 and RB7 are dedicated to In-Circuit Debug.
bit 6-3	**Unimplemented:** Read as ‘0’
bit 2	**LVP:** Low Voltage ICSP Enable bit `1` = Low Voltage ICSP enabled `0` = Low Voltage ICSP disabled
bit 1	**Unimplemented:** Read as ‘0’
bit 0	**STVREN:** Stack Full/Underflow Reset Enable bit `1` = Stack Full/Underflow will cause RESET `0` = Stack Full/Underflow will not cause RESET

Legend:		
R = Readable bit	C = Clearable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-5: CONFIG5L: CONFIGURATION REGISTER 5 LOW (BYTE ADDRESS 300008h)

U-0	U-0	U-0	U-0	U-1	R/C-1	R/C-1	R/C-1
—	—	—	—	—	CP2[1]	CP1	CP0
bit 7							bit 0

bit 7-4 **Unimplemented:** Read as ‘0’

bit 3 **Unimplemented and reserved:** Maintain as ‘1’

bit 2 **CP2:** Code Protection bit[1]
1 = Block 2 (004000-005FFFh) not code protected
0 = Block 2 (004000-005FFFh) code protected

bit 1 **CP1:** Code Protection bit
1 = Block 1 (002000-003FFFh) not code protected
0 = Block 1 (002000-003FFFh) code protected

bit 0 **CP0:** Code Protection bit
1 = Block 0 (000200-001FFFh) not code protected
0 = Block 0 (000200-001FFFh) code protected

Note 1: Unimplemented in PIC18FX439 devices; maintain this bit set.

Legend:		
R = Readable bit	C = Clearable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-6: CONFIG5H: CONFIGURATION REGISTER 5 HIGH (BYTE ADDRESS 300009h)

R/C-1	R/C-1	U-0	U-0	U-0	U-0	U-0	U-0
CPD	CPB	—	—	—	—	—	—
bit 7							bit 0

bit 7 **CPD:** Data EEPROM Code Protection bit
1 = Data EEPROM not code protected
0 = Data EEPROM code protected

bit 6 **CPB:** Boot Block Code Protection bit
1 = Boot block (000000-0001FFh) not code protected
0 = Boot block (000000-0001FFh) code protected

bit 5-0 **Unimplemented:** Read as ‘0’

Legend:		
R = Readable bit	C = Clearable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-7: CONFIG6L: CONFIGURATION REGISTER 6 LOW (BYTE ADDRESS 30000Ah)

U-0	U-0	U-0	U-0	U-1	R/C-1	R/C-1	R/C-1
—	—	—	—	—	WRT2[(1)]	WRT1	WRT0
bit 7							bit 0

bit 7-4 **Unimplemented:** Read as ‘0’

bit 3 **Unimplemented and reserved:** Maintain as ‘1’

bit 2 **WRT2:** Write Protection bit[(1)]
1 = Block 2 (004000-005FFFh) not write protected
0 = Block 2 (004000-005FFFh) write protected

bit 1 **WRT1:** Write Protection bit
1 = Block 1 (002000-003FFFh) not write protected
0 = Block 1 (002000-003FFFh) write protected

bit 0 **WRT0:** Write Protection bit
1 = Block 0 (000200h-001FFFh) not write protected
0 = Block 0 (000200h-001FFFh) write protected

Note 1: Unimplemented in PIC18FX439 devices; maintain this bit set.

Legend:		
R = Readable bit	C = Clearable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-8: CONFIG6H: CONFIGURATION REGISTER 6 HIGH (BYTE ADDRESS 30000Bh)

R/C-1	R/C-1	C-1	U-0	U-0	U-0	U-0	U-0
WRTD	WRTB	WRTC	—	—	—	—	—
bit 7							bit 0

bit 7 **WRTD:** Data EEPROM Write Protection bit
1 = Data EEPROM not write protected
0 = Data EEPROM write protected

bit 6 **WRTB:** Boot Block Write Protection bit
1 = Boot block (000000-0001FFh) not write protected
0 = Boot block (000000-0001FFh) write protected

bit 5 **WRTC:** Configuration Register Write Protection bit
1 = Configuration registers (300000-3000FFh) not write protected
0 = Configuration registers (300000-3000FFh) write protected

Note: This bit is read only, and cannot be changed in User mode.

bit 4-0 **Unimplemented:** Read as ‘0’

Legend:		
R = Readable bit	C = Clearable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-9: CONFIG7L: CONFIGURATION REGISTER 7 LOW (BYTE ADDRESS 30000Ch)

U-0	U-0	U-0	U-0	U-1	R/C-1	R/C-1	R/C-1
—	—	—	—	—	EBTR2(1)	EBTR1	EBTR0
bit 7							bit 0

bit 7-4	**Unimplemented:** Read as '0'
bit 3	**Unimplemented and reserved:** Maintain as '1'
bit 2	**EBTR2:** Table Read Protection bit(1) `1` = Block 2 (004000-005FFFh) not protected from Table Reads executed in other blocks `0` = Block 2 (004000-005FFFh) protected from Table Reads executed in other blocks
bit 1	**EBTR1:** Table Read Protection bit `1` = Block 1 (002000-003FFFh) not protected from Table Reads executed in other blocks `0` = Block 1 (002000-003FFFh) protected from Table Reads executed in other blocks
bit 0	**EBTR0:** Table Read Protection bit `1` = Block 0 (000200h-001FFFh) not protected from Table Reads executed in other blocks `0` = Block 0 (000200h-001FFFh) protected from Table Reads executed in other blocks

Note 1: Unimplemented in PIC18FX439 devices; maintain this bit set.

Legend:		
R = Readable bit	C = Clearable bit	U = Unimplemented bit, read as '0'
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-10: CONFIG7H: CONFIGURATION REGISTER 7 HIGH (BYTE ADDRESS 30000Dh)

U-0	R/C-1	U-0	U-0	U-0	U-0	U-0	U-0
—	EBTRB	—	—	—	—	—	—
bit 7							bit 0

bit 7	**Unimplemented:** Read as '0'
bit 6	**EBTRB:** Boot Block Table Read Protection bit `1` = Boot block (000000-0001FFh) not protected from Table Reads executed in other blocks `0` = Boot block (000000-0001FFh) protected from Table Reads executed in other blocks
bit 5-0	**Unimplemented:** Read as '0'

Legend:		
R = Readable bit	C = Clearable bit	U = Unimplemented bit, read as '0'
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-11: DEVID1: DEVICE ID REGISTER 1 FOR PIC18FXX39 (BYTE ADDRESS 3FFFFEh)

R	R	R	R	R	R	R	R
DEV2	DEV1	DEV0	REV4	REV3	REV2	REV1	REV0
bit 7							bit 0

bit 7-5 **DEV2:DEV0:** Device ID bits

`000` = PIC18F2539
`001` = PIC18F4539
`100` = PIC18F2439
`101` = PIC18F4439

bit 4-0 **REV4:REV0:** Revision ID bits

These bits are used to indicate the device revision.

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

REGISTER 20-12: DEVID2: DEVICE ID REGISTER 2 FOR PIC18FXX39 (BYTE ADDRESS 3FFFFFh)

R	R	R	R	R	R	R	R
DEV10	DEV9	DEV8	DEV7	DEV6	DEV5	DEV4	DEV3
bit 7							bit 0

bit 7-0 **DEV10:DEV3:** Device ID bits

These bits are used with the DEV2:DEV0 bits in the Device ID Register 1 to identify the part number.

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as ‘0’
- n = Value when device is unprogrammed		u = Unchanged from programmed state

20.2 Watchdog Timer (WDT)

The Watchdog Timer is a free running on-chip RC oscillator, which does not require any external components. This RC oscillator is separate from the RC oscillator of the OSC1/CLKI pin. That means that the WDT will run, even if the clock on the OSC1/CLKI and OSC2/CLKO/RA6 pins of the device has been stopped, for example, by execution of a `SLEEP` instruction.

During normal operation, a WDT time-out generates a device RESET (Watchdog Timer Reset). If the device is in SLEEP mode, a WDT time-out causes the device to wake-up and continue with normal operation (Watchdog Timer Wake-up). The $\overline{\text{TO}}$ bit in the RCON register will be cleared upon a WDT time-out.

The Watchdog Timer 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.

The WDT time-out period values may be found in the Electrical Specifications (Section 23.0) under parameter D031. Values for the WDT postscaler may be assigned using the configuration bits.

Note 1: The `CLRWDT` and `SLEEP` instructions clear the WDT and the postscaler, if assigned to the WDT and prevent it from timing out and generating a device RESET condition.

2: When a `CLRWDT` instruction is executed and the postscaler is assigned to the WDT, the postscaler count will be cleared, but the postscaler assignment is not changed.

20.2.1 CONTROL REGISTER

Register 20-13 shows the WDTCON register. This is a readable and writable register, which contains a control bit that allows software to override the WDT enable configuration bit, only when the configuration bit has disabled the WDT.

REGISTER 20-13: 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'

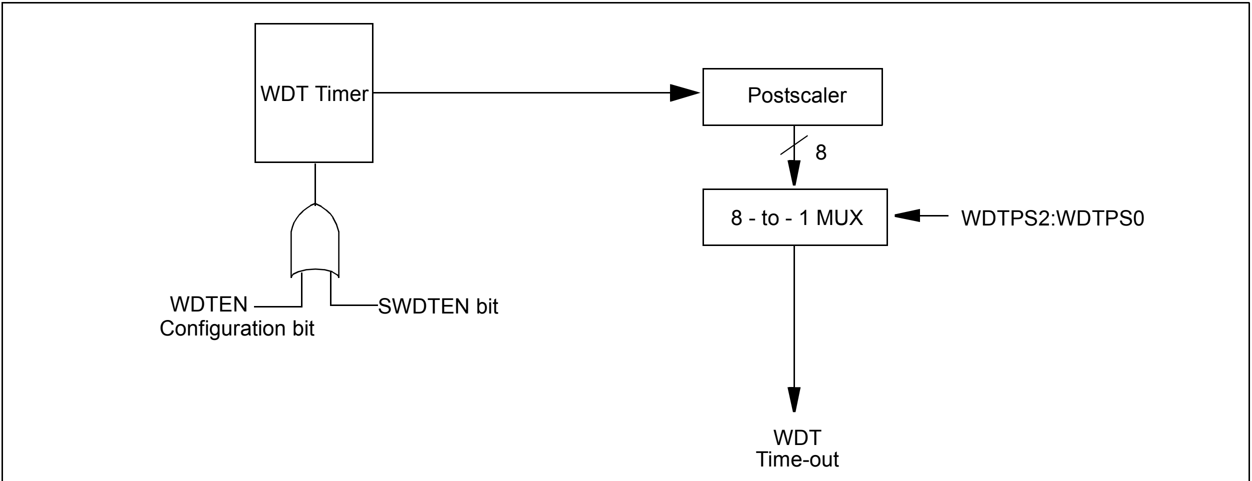
bit 0 **SWDTEN:** Software Controlled Watchdog Timer Enable bit
`1` = Watchdog Timer is on
`0` = Watchdog Timer is turned off if the WDTEN configuration bit in the configuration register = 0

Legend:	
R = Readable bit	W = Writable bit
U = Unimplemented bit, read as '0'	- n = Value at POR

20.2.2 WDT POSTSCALER

The WDT has a postscaler that can extend the WDT Reset period. The postscaler is selected at the time of the device programming, by the value written to the CONFIG2H configuration register.

FIGURE 20-1: WATCHDOG TIMER BLOCK DIAGRAM



Note: WDPS2:WDPS0 are bits in register CONFIG2H.

TABLE 20-2: SUMMARY OF WATCHDOG TIMER REGISTERS

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CONFIG2H	—	—	—	—	WDTPS2	WDTPS2	WDTPS0	WDTEN
RCON	IPEN	—	—	$\overline{\text{RI}}$	$\overline{\text{TO}}$	$\overline{\text{PD}}$	$\overline{\text{POR}}$	$\overline{\text{BOR}}$
WDTCON	—	—	—	—	—	—	—	SWDTEN

Legend: Shaded cells are not used by the Watchdog Timer.

20.3 Power-down Mode (SLEEP)

Power-down mode is entered by executing a `SLEEP` instruction.

If enabled, the Watchdog Timer will be cleared, but keeps running, the $\overline{\text{PD}}$ bit (RCON<3>) is cleared, the $\overline{\text{TO}}$ (RCON<4>) 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, place all I/O pins at either V_{DD} or V_{SS}, ensure no external circuitry is drawing current from the I/O pin, power-down the A/D and disable external clocks. Pull all I/O pins that are hi-impedance inputs, high or low externally, to avoid switching currents caused by floating inputs. The T0CKI input should also be at V_{DD} or V_{SS} for lowest current consumption. The contribution from on-chip pull-ups on PORTB should be considered.

The $\overline{\text{MCLR}}$ pin must be at a logic high level (V_{IHMC}).

20.3.1 WAKE-UP FROM SLEEP

The device can wake-up from SLEEP through one of the following events:

1. External RESET input on $\overline{\text{MCLR}}$ pin.
2. Watchdog Timer Wake-up (if WDT was enabled).
3. Interrupt from INT pin, RB port change or a Peripheral Interrupt.

The following peripheral interrupts can wake the device from SLEEP:

1. PSP read or write.
2. TMR1 interrupt. Timer1 must be operating as an asynchronous counter.
3. TMR3 interrupt. Timer3 must be operating as an asynchronous counter.
4. CCP Capture mode interrupt.
5. Special event trigger (Timer1 in Asynchronous mode using an external clock).
6. MSSP (START/STOP) bit detect interrupt.
7. MSSP transmit or receive in Slave mode (SPI/I^{2}C).
8. USART RX or TX (Synchronous Slave mode).
9. A/D conversion (when A/D clock source is RC).
10. EEPROM write operation complete.
11. LVD interrupt.

Other peripherals cannot generate interrupts, since during SLEEP, no on-chip clocks are present.

External $\overline{\text{MCLR}}$ Reset will cause a device RESET. All other events are considered a continuation of program execution and will cause a "wake-up". The $\overline{\text{TO}}$ and $\overline{\text{PD}}$ bits in the RCON register can be used to determine the cause of the device RESET. The $\overline{\text{PD}}$ bit, which is set on power-up, is cleared when SLEEP is invoked. The $\overline{\text{TO}}$ bit is cleared, if a WDT time-out occurred (and caused 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.

20.3.2 WAKE-UP USING INTERRUPTS

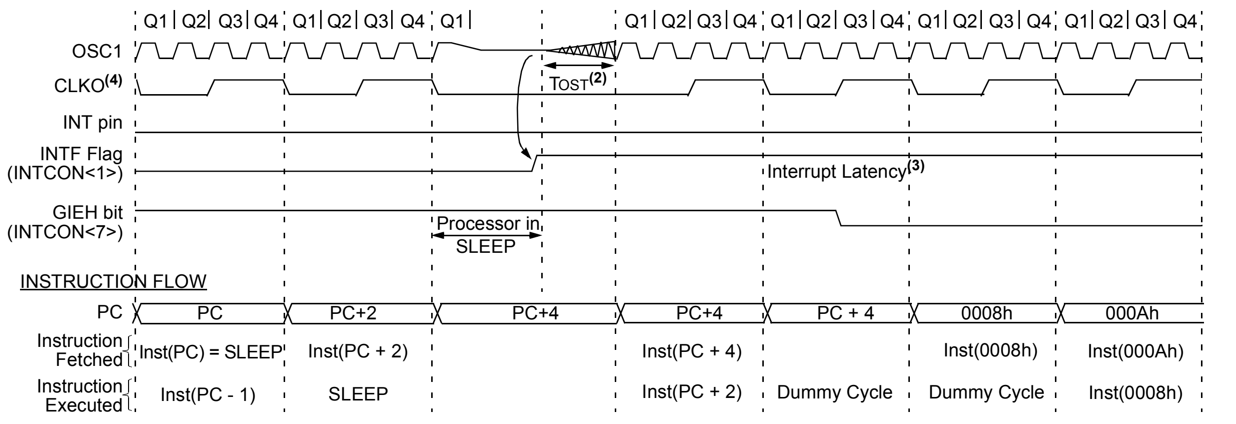
When global interrupts are disabled (GIE cleared) and any interrupt source has both its interrupt enable bit and interrupt flag bit set, one of the following will occur:

- 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 $\overline{\text{TO}}$ bit will not be set and $\overline{\text{PD}}$ bits 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 $\overline{\text{TO}}$ bit will be set and the $\overline{\text{PD}}$ 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 $\overline{\text{PD}}$ bit. If the $\overline{\text{PD}}$ bit is set, the `SLEEP` instruction was executed as a `NOP`.

To ensure that the WDT is cleared, a `CLRWDT` instruction should be executed before a `SLEEP` instruction.

FIGURE 20-2: WAKE-UP FROM SLEEP THROUGH INTERRUPT(1,2)



Note 1: XT, HS or LP Oscillator mode assumed.
2: GIE = 1 assumed. In this case, after wake-up, the processor jumps to the interrupt routine. If GIE = 0, execution will continue in-line.
3: TOST = 1024 TOSC (drawing not to scale). This delay will not occur for RC and EC Osc modes.
4: CLKO is not available in these Osc modes, but shown here for timing reference.

20.4 Program Verification and Code Protection

The overall structure of the code protection on the PIC18 FLASH devices differs significantly from other PICmicro devices. The user program memory is divided on binary boundaries into individual blocks, each of which has three separate code protection bits associated with it:

- Code Protect bit (CPn)
- Write Protect bit (WRTn)
- External Block Table Read bit (EBTRn)

The code protection bits are located in Configuration Registers 5L through 7H. Their locations within the registers are summarized in Table 20-3.

In the PIC18FXX39 family, program memory is divided into segments of 8 Kbytes. The first block in turn divided into a boot block of 512 bytes and a separately protected remainder (Block 0) of 7.5 Kbytes. This means for PIC18FXX39 devices, that there may be up to five blocks, depending on the program memory size. The organization of the blocks and their associated code protection bits are shown in Figure 20-3.

For PIC18FX439 devices, program memory is divided into three blocks: a boot block, Block 0 (7.5 Kbytes) and Block 1 (8 Kbytes). Block 1 is further divided in half; the upper portion above 3000h is reserved, and unavailable to user applications. The entire block can be protected as a whole by bits CP1, WRT1 and EBTR1. By default, Block 1 is not code protected.

For PIC18FX539 devices, program memory is divided into five blocks: the boot block, Block 0 (7.5 Kbytes), and Blocks 1 through 3 (8 Kbytes). Code protection is implemented for the boot block and Blocks 0 through 2. There is no provision for code protection for Block 3.

Note: The reserved segments of the program memory space are used by the Motor Control kernel. For the kernel to function properly, this area must not be write protected. If users are developing applications that require code protection for PIC18FX439 devices, they should restrict program code (or at least those sections requiring protection) to below the 1FFFh memory boundary.

FIGURE 20-3: CODE PROTECTED PROGRAM MEMORY FOR PIC18FXX39

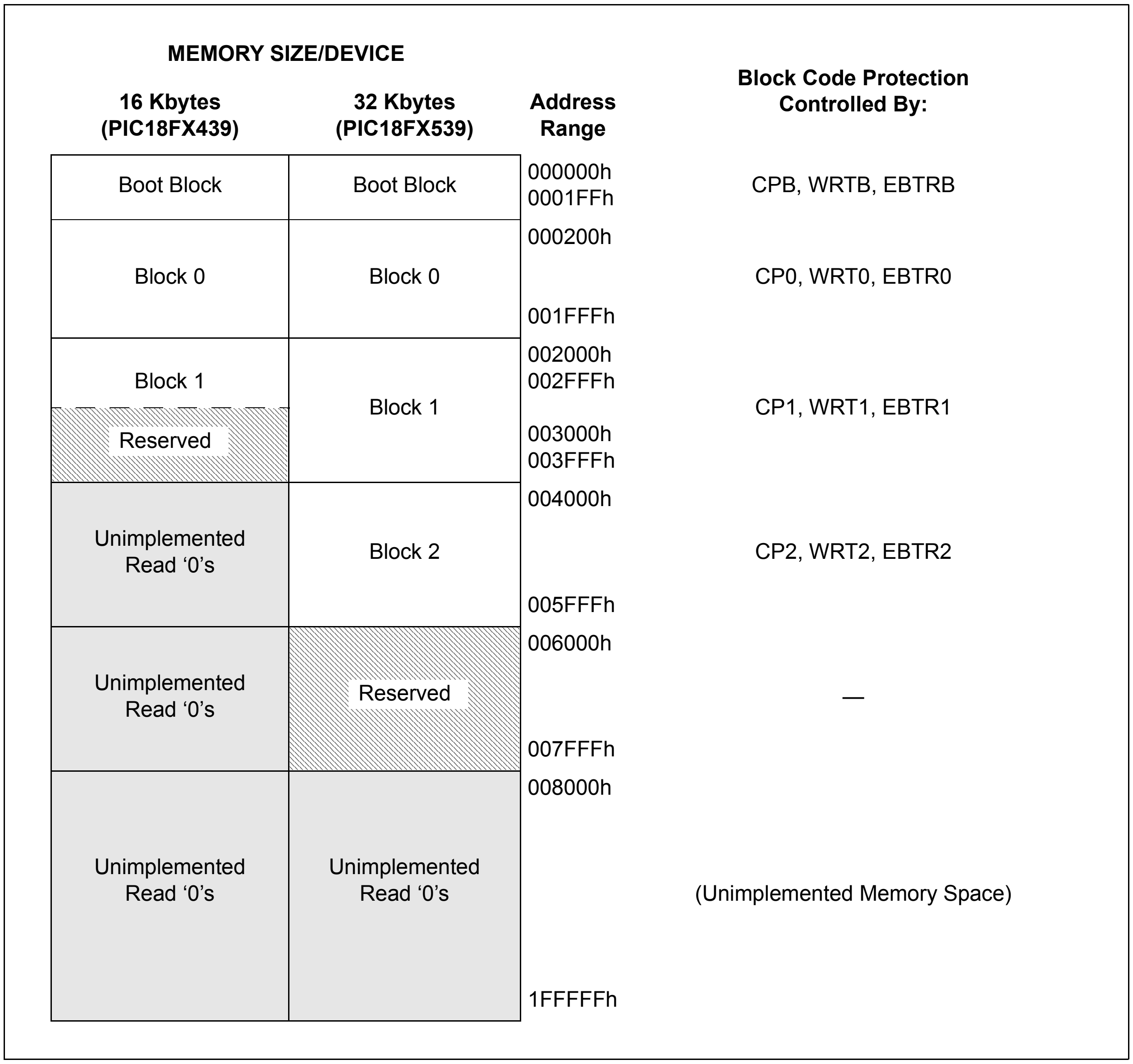


TABLE 20-3: SUMMARY OF CODE PROTECTION REGISTERS

File Name		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
300008h	CONFIG5L	—	—	—	—	—[1]	CP2	CP1	CP0
300009h	CONFIG5H	CPD	CPB	—	—	—	—	—	—
30000Ah	CONFIG6L	—	—	—	—	—[1]	WRT2	WRT1	WRT0
30000Bh	CONFIG6H	WRTD	WRTB	WRTC	—	—	—	—	—
30000Ch	CONFIG7L	—	—	—	—	—[1]	EBTR2	EBTR1	EBTR0
30000Dh	CONFIG7H	—	EBTRB	—	—	—	—	—	—

Legend: Shaded cells are unimplemented.
Note 1: Unimplemented, but reserved; maintain this bit set.

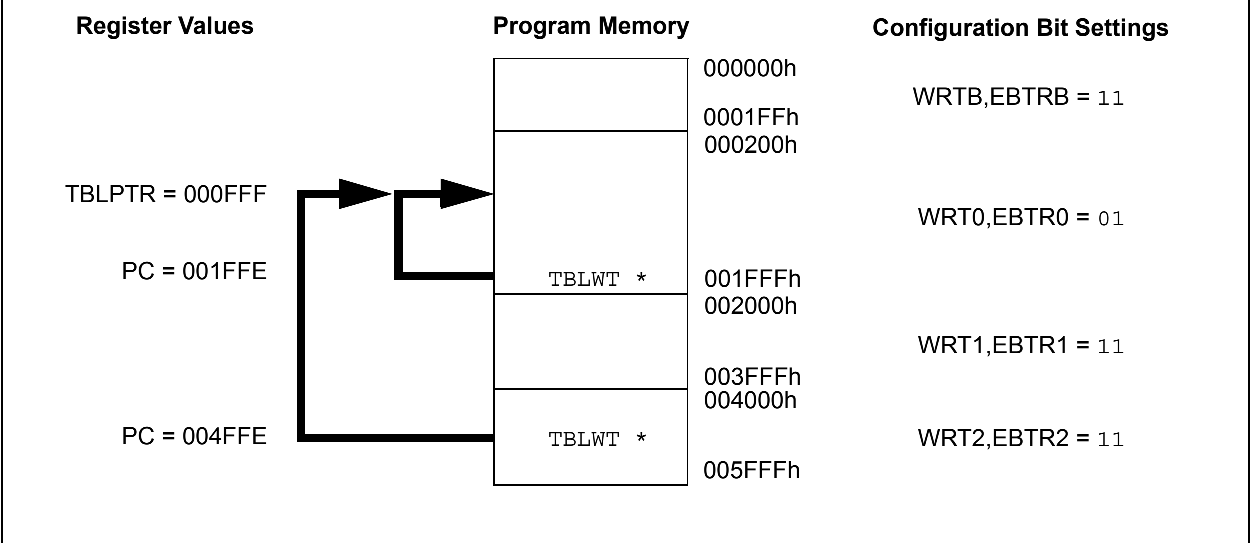
20.4.1 PROGRAM MEMORY CODE PROTECTION

The user memory may be read to, or written from, any location using the Table Read and Table Write instructions. The device ID may be read with Table Reads. The configuration registers may be read and written with the Table Read and Table Write instructions.

In User mode, the CPn bits have no direct effect. CPn bits inhibit external reads and writes. A block of user memory may be protected from Table Writes if the WRTn configuration bit is ‘0’. The EBTRn bits control Table Reads. For a block of user memory with the EBTRn bit set to ‘0’, a Table Read instruction that executes from within that block is allowed to read. A Table Read instruction that executes from a location outside of that block is not allowed to read, and will result in reading ‘0’s. Figures 20-4 through 20-6 illustrate Table Write and Table Read protection.

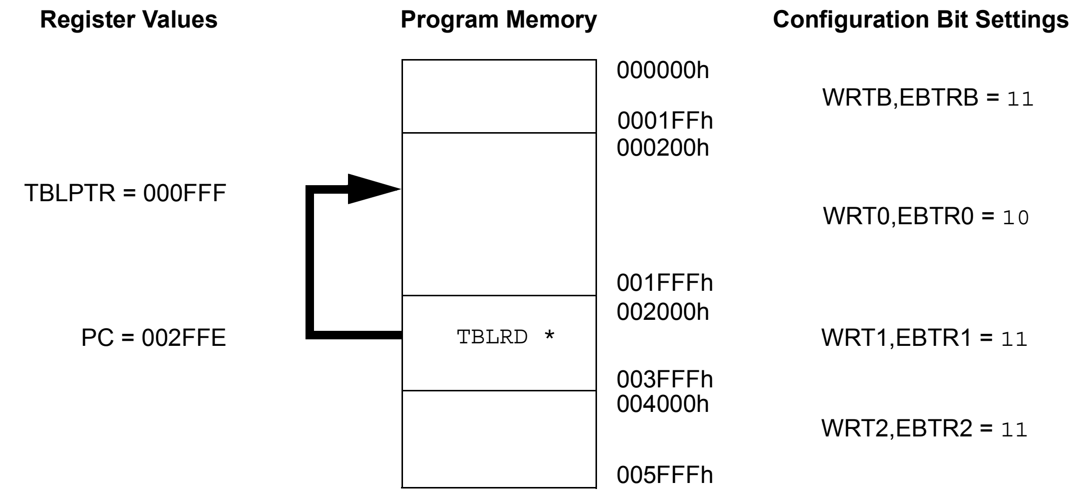
Note:	Code protection bits may only be written to a ‘0’ from a ‘1’ state. It is not possible to write a ‘1’ to a bit in the ‘0’ state. Code protection bits are only set to ‘1’ by a block erase function. The block erase function can only be initiated via ICSP or an external programmer.

FIGURE 20-4: TABLE WRITE (WRTn) DISALLOWED



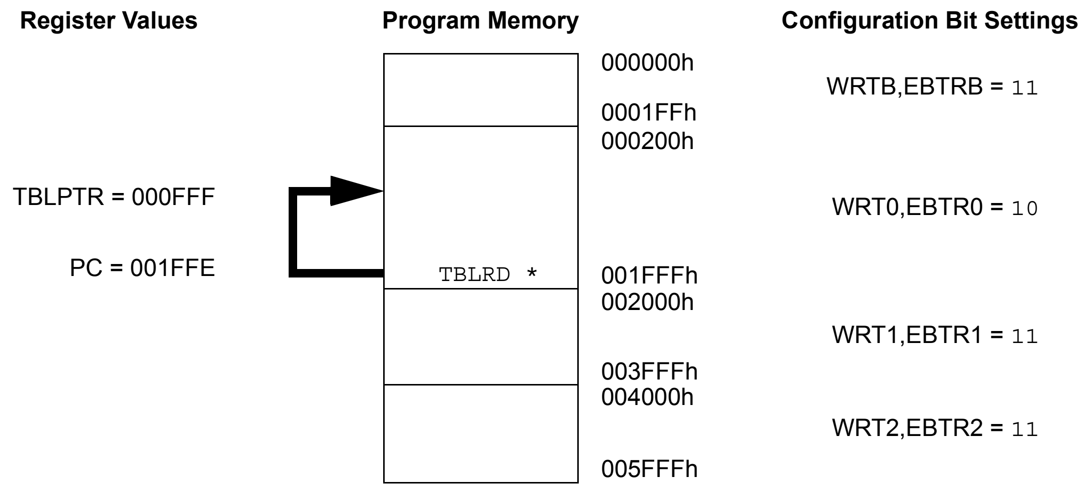
Results: All Table Writes disabled to Blockn whenever WRTn = 0.

FIGURE 20-5: EXTERNAL BLOCK TABLE READ (EBTRn) DISALLOWED



Results: All Table Reads from external blocks to Blockn are disabled whenever EBTRn = 0.
TABLAT register returns a value of '0'.

FIGURE 20-6: EXTERNAL BLOCK TABLE READ (EBTRn) ALLOWED



Results: Table Reads permitted within Blockn, even when EBTRBn = 0.
TABLAT register returns the value of the data at the location TBLPTR.

20.4.2 DATA EEPROM CODE PROTECTION

The entire Data EEPROM is protected from external reads and writes by two bits: CPD and WRTD. CPD inhibits external reads and writes of Data EEPROM. WRTD inhibits external writes to Data EEPROM. The CPU can continue to read and write Data EEPROM, regardless of the protection bit settings.

20.4.3 CONFIGURATION REGISTER PROTECTION

The configuration registers can be write protected. The WRTC bit controls protection of the configuration registers. In User mode, the WRTC bit is readable only. WRTC can only be written via ICSP or an external programmer.

20.5 ID Locations

Eight memory locations (200000h - 200007h) 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` and `TBLWT` instructions, or during program/verify. The ID locations can be read when the device is code protected.

The sequence for programming the ID locations is similar to programming the FLASH memory (see Section 5.5.1).

20.6 In-Circuit Serial Programming

PIC18FXXX microcontrollers can be serially programmed 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. This allows customers to manufacture boards with unprogrammed devices, and then program the microcontroller just before shipping the product. This also allows the most recent firmware, or a custom firmware to be programmed.

20.7 In-Circuit Debugger

When the DEBUG bit in configuration register CONFIG4L is programmed to a '0', the In-Circuit Debugger functionality is enabled. This function allows simple debugging functions when used with MPLAB® IDE. When the microcontroller has this feature enabled, some of the resources are not available for general use. Table 20-4 shows which features are consumed by the background debugger.

TABLE 20-4: DEBUGGER RESOURCES

I/O pins	RB6, RB7
Stack	2 levels
Program Memory	512 bytes
Data Memory	10 bytes

To use the In-Circuit Debugger function of the microcontroller, the design must implement In-Circuit Serial Programming connections to $\overline{\text{MCLR}}$/VPP, VDD, GND, RB7 and RB6. This will interface to the In-Circuit Debugger module available from Microchip, or one of the third party development tool companies.

20.8 Low Voltage ICSP Programming

The LVP bit configuration register CONFIG4L enables low voltage ICSP programming. This mode allows the microcontroller to be programmed via ICSP using a VDD source in the operating voltage range. This only means that VPP does not have to be brought to VIHH, but can instead be left at the normal operating voltage. In this mode, the RB5/PGM pin is dedicated to the programming function and ceases to be a general purpose I/O pin. During programming, VDD is applied to the $\overline{\text{MCLR}}$/VPP pin. To enter Programming mode, VDD must be applied to the RB5/PGM, provided the LVP bit is set. The LVP bit defaults to a '`1`' from the factory.

Note 1: The High Voltage Programming mode is always available, regardless of the state of the LVP bit, by applying VIHH to the $\overline{\text{MCLR}}$ pin.

2: While in low voltage ICSP mode, the RB5 pin can no longer be used as a general purpose I/O pin, and should be held low during normal operation to protect against inadvertent ICSP mode entry.

3: When using low voltage ICSP programming (LVP), the pull-up on RB5 becomes disabled. If TRISB bit 5 is cleared, thereby setting RB5 as an output, LATB bit 5 must also be cleared for proper operation.

If Low Voltage Programming mode is not used, the LVP bit can be programmed to a '0' and RB5/PGM becomes a digital I/O pin. However, the LVP bit may only be programmed when programming is entered with VIHH on $\overline{\text{MCLR}}$/VPP.

It should be noted that once the LVP bit is programmed to '0', only the High Voltage Programming mode is available and only High Voltage Programming mode can be used to program the device.

When using low voltage ICSP, the part must be supplied 4.5V to 5.5V, if a bulk erase will be executed. This includes reprogramming of the code protect bits from an on-state to an off-state. For all other cases of low voltage ICSP, the part may be programmed at the normal operating voltage. This means unique user IDs, or user code can be reprogrammed or added.

21.0 INSTRUCTION SET SUMMARY

The PIC18FXXX 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 there are three instructions that require two program memory locations.

Each single word instruction is a 16-bit word divided into an OPCODE, which 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

The PIC18FXXX instruction set summary in Table 21-2 lists **byte-oriented**, **bit-oriented**, **literal** and **control** operations. Table 21-1 shows the opcode field descriptions.

Most **byte-oriented** instructions have three operands:

1. The file register (specified by 'f')
2. The destination of the result (specified by 'd')
3. The accessed memory (specified by 'a')

The file register designator 'f' specifies which file register is to be used by the instruction.

The destination designator 'd' 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 'f')
2. The bit in the file register (specified by 'b')
3. The accessed memory (specified by 'a')

The bit field designator 'b' selects the number of the bit affected by the operation, while the file register designator 'f' represents the number of the file in which the bit is located.

The **literal** instructions may use some of the following operands:

- A literal value to be loaded into a file register (specified by 'k')
- The desired FSR register to load the literal value into (specified by 'f')
- No operand required (specified by '—')

The **control** instructions may use some of the following operands:

- A program memory address (specified by 'n')
- The mode of the Call or Return instructions (specified by 's')
- The mode of the Table Read and Table Write instructions (specified by 'm')
- No operand required (specified by '—')

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 MSbs 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 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.

Figure 21-1 shows the general formats that the instructions can have.

All examples use the format '`nnh`' to represent a hexadecimal number, where '`h`' signifies a hexadecimal digit.

The Instruction Set Summary, shown in Table 21-2, lists the instructions recognized by the Microchip Assembler (MPASM™).

Section 21.1 provides a description of each instruction.

TABLE 21-1: OPCODE FIELD DESCRIPTIONS

Field	Description
a	RAM access bit a = 0: RAM location in Access RAM (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 (Fast mode)
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.
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.
$\overline{\text{TO}}$	Time-out bit.
$\overline{\text{PD}}$	Power-down bit.
C, DC, Z, OV, N	ALU status bits Carry, Digit Carry, Zero, Overflow, Negative.
[ ]	Optional.
( )	Contents.
$\rightarrow$	Assigned to.
< >	Register bit field.
$\in$	In the set of.
italics	User defined term (font is courier).

FIGURE 21-1: GENERAL FORMAT FOR INSTRUCTIONS

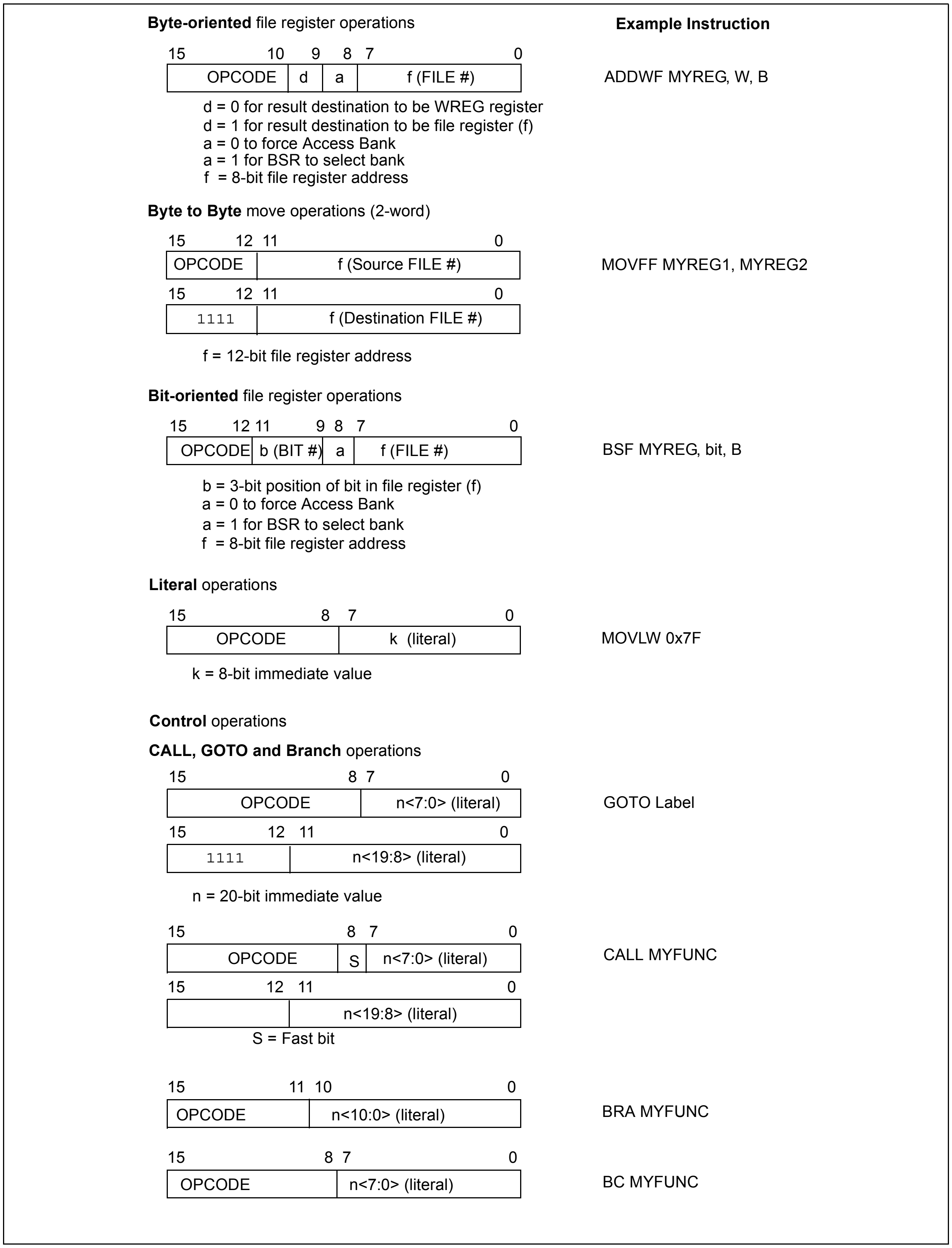

Byte-oriented file register operations
Example Instruction
15 10 9 8 7 0
OPCODE | d | a | f (FILE #)
ADDWF MYREG, W, B
d = 0 for result destination to be WREG register
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address
Byte to Byte move operations (2-word)
15 12 11 0
OPCODE | f (Source FILE #)
MOVFF MYREG1, MYREG2
15 12 11 0
1111 | f (Destination FILE #)
f = 12-bit file register address
Bit-oriented file register operations
15 12 11 9 8 7 0
OPCODE | b (BIT #) | a | f (FILE #)
BSF MYREG, bit, B
b = 3-bit position of bit in file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address
Literal operations
15 8 7 0
OPCODE | k (literal)
MOVLW 0x7F
k = 8-bit immediate value
Control operations
CALL, GOTO and Branch operations
15 8 7 0
OPCODE | n<7:0> (literal)
GOTO Label
15 12 11 0
1111 | n<19:8> (literal)
n = 20-bit immediate value
15 8 7 0
OPCODE | S | n<7:0> (literal)
CALL MYFUNC
15 12 11 0
n<19:8> (literal)
S = Fast bit
15 11 10 0
OPCODE | n<10:0> (literal)
BRA MYFUNC
15 8 7 0
OPCODE | n<7:0> (literal)
BC MYFUNC

TABLE 21-2: PIC18FXXX INSTRUCTION SET

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	01da0	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010	0da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	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, N	1, 2
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1
MOVFF	f_s, f_d	Move f_s (source) to 1st word	2	1100	ffff	ffff	ffff	None	
		f_d (destination) 2nd word		1111	ffff	ffff	ffff		
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, Z, N	
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	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	100a	ffff	ffff	None	
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	
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	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 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 memory, the write will continue until terminated.

TABLE 21-2: PIC18FXXX INSTRUCTION SET (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	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	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 memory, the write will continue until terminated.

TABLE 21-2: PIC18FXXX INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
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) 2nd word	2	`1110`	`1110`	`00ff`	`kkkk`	None	
		to FSRx 1st word		`1111`	`0000`	`kkkk`	`kkkk`		
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		`0000`	`0000`	`0000`	`1001`	None	
TBLRD*-		Table Read with post-decrement		`0000`	`0000`	`0000`	`1010`	None	
TBLRD+*		Table Read with pre-increment		`0000`	`0000`	`0000`	`1011`	None	
TBLWT*		Table Write	2 (5)	`0000`	`0000`	`0000`	`1100`	None	
TBLWT*+		Table Write with post-increment		`0000`	`0000`	`0000`	`1101`	None	
TBLWT*-		Table Write with post-decrement		`0000`	`0000`	`0000`	`1110`	None	
TBLWT+*		Table Write with pre-increment		`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 memory, the write will continue until terminated.

21.1 Instruction Set

ADDLW ADD literal to W

Syntax:	[*label*] ADDLW k
Operands:	$0 \le k \le 255$
Operation:	$(W) + k \rightarrow W$
Status Affected:	N, OV, C, DC, Z
Encoding:	`0000` `1111` `kkkk` `kkkk`
Description:	The contents of W are added to the 8-bit literal 'k' and the result is placed in W.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

Example: `ADDLW 0x15`

Before Instruction

W = 0x10

After Instruction

W = 0x25

ADDWF ADD W to f

Syntax:	[*label*] ADDWF f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(W) + (f) \rightarrow$ dest
Status Affected:	N, OV, C, DC, Z
Encoding:	`0010` `01da` `ffff` `ffff`
Description:	Add W to register 'f'. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If ‘a’ is 0, the Access Bank will be selected. If ‘a’ is 1, the BSR is used.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `ADDWF REG, 0, 0`

Before Instruction

W = 0x17
REG = 0xC2

After Instruction

W = 0xD9
REG = 0xC2

ADDWFC **ADD W and Carry bit to f**

Syntax:	[*label*] ADDWFC f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(W) + (f) + (C) $\rightarrow$ dest
Status Affected:	N,OV, C, DC, Z
Encoding:	`0010` `00da` `ffff` `ffff`
Description:	Add W, the Carry Flag and data memory location 'f'. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed in data memory location 'f'. If ‘a’ is 0, the Access Bank will be selected. If ‘a’ is 1, the BSR will not be overridden.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `ADDWFC REG, 0, 1`

Before Instruction

Carry bit	=	1
REG	=	0x02
W	=	0x4D

After Instruction

Carry bit	=	0
REG	=	0x02
W	=	0x50

ANDLW **AND literal with W**

Syntax:	[*label*] ANDLW k
Operands:	$0 \le k \le 255$
Operation:	(W) .AND. k $\rightarrow$ W
Status Affected:	N,Z
Encoding:	`0000` `1011` `kkkk` `kkkk`
Description:	The contents of W are ANDed with the 8-bit literal 'k'. The result is placed in W.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

Example: `ANDLW 0x5F`

Before Instruction

W	=	0xA3

After Instruction

W	=	0x03

ANDWF AND W with f

Syntax:	[*label*] ANDWF f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(W) .AND. (f) → dest
Status Affected:	N,Z

Encoding:

`0001`	`01da`	`ffff`	`ffff`

Description: The contents of W are AND'ed with register 'f'. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the BSR will not be overridden (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `ANDWF REG, 0, 0`

Before Instruction

W	=	0x17
REG	=	0xC2

After Instruction

W	=	0x02
REG	=	0xC2

BC Branch if Carry

Syntax:	[*label*] BC n
Operands:	-128 ≤ n ≤ 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' 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 Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BC 5`

Before Instruction

PC	=	address `(HERE)`

After Instruction

If Carry	=	1;
PC	=	address `(HERE+12)`
If Carry	=	0;
PC	=	address `(HERE+2)`

BCF Bit Clear f

Syntax:	[*label*] BCF f,b[,a]
Operands:	0 ≤ f ≤ 255 0 ≤ b ≤ 7 a ∈ [0,1]
Operation:	0 → f<b>
Status Affected:	None
Encoding:	`1001` `bbba` `ffff` `ffff`
Description:	Bit 'b' in register 'f' is cleared. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `BCF FLAG_REG, 7, 0`

Before Instruction
FLAG_REG = 0xC7
After Instruction
FLAG_REG = 0x47

BN Branch if Negative

Syntax:	[*label*] BN n
Operands:	-128 ≤ n ≤ 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' 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 Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BN Jump`

Before Instruction
PC = address (`HERE`)
After Instruction
If Negative = 1;
PC = address (`Jump`)
If Negative = 0;
PC = address (`HERE+2`)

BNC Branch if Not Carry

Syntax:	[*label*] BNC n
Operands:	-128 ≤ n ≤ 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’ 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 Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE      BNC  Jump`

Before Instruction

PC	=	address `(HERE)`

After Instruction

If Carry	=	0;
PC	=	address `(Jump)`
If Carry	=	1;
PC	=	address `(HERE+2)`

BNN Branch if Not Negative

Syntax:	[*label*] BNN n
Operands:	-128 ≤ n ≤ 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’ 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 Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE      BNN  Jump`

Before Instruction

PC	=	address `(HERE)`

After Instruction

If Negative	=	0;
PC	=	address `(Jump)`
If Negative	=	1;
PC	=	address `(HERE+2)`

BNOV — Branch if Not Overflow

Syntax:	[*label*] BNOV n
Operands:	-128 ≤ n ≤ 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’ 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 Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE      BNOV  Jump`

Before Instruction

PC	=	address `(HERE)`

After Instruction

If Overflow	=	0;
PC	=	address `(Jump)`
If Overflow	=	1;
PC	=	address `(HERE+2)`

BNZ — Branch if Not Zero

Syntax:	[*label*] BNZ n
Operands:	-128 ≤ n ≤ 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’ 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 Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE      BNZ   Jump`

Before Instruction

PC	=	address `(HERE)`

After Instruction

If Zero	=	0;
PC	=	address `(Jump)`
If Zero	=	1;
PC	=	address `(HERE+2)`

BRA **Unconditional Branch**

Syntax:	[*label*] BRA n
Operands:	$-1024 \le n \le 1023$
Operation:	(PC) + 2 + 2n $\rightarrow$ PC
Status Affected:	None
Encoding:	`1101` `0nnn` `nnnn` `nnnn`
Description:	Add the 2's complement number '2n' 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'	Process Data	Write to PC
No operation	No operation	No operation	No operation

<u>Example</u>: `HERE BRA Jump`

Before Instruction
PC = address `(HERE)`
After Instruction
PC = address `(Jump)`

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. If 'a' is 0 Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

<u>Example</u>: `BSF FLAG_REG, 7, 1`

Before Instruction
FLAG_REG = 0x0A
After Instruction
FLAG_REG = 0x8A

BTFSC **Bit Test File, Skip if Clear**

Syntax:	[*label*] BTFSC f,b[,a]
Operands:	0 ≤ f ≤ 255 0 ≤ b ≤ 7 a ∈ [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 two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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     BTFSC   FLAG, 1, 0
FALSE    :
TRUE     :
```

Before Instruction

PC	=	address `(HERE)`

After Instruction

If FLAG<1>	=	0;
PC	=	address `(TRUE)`
If FLAG<1>	=	1;
PC	=	address `(FALSE)`

BTFSS **Bit Test File, Skip if Set**

Syntax:	[*label*] BTFSS f,b[,a]
Operands:	0 ≤ f ≤ 255 0 ≤ b ≤ 7 a ∈ [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 two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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     BTFSS   FLAG, 1, 0
FALSE    :
TRUE     :
```

Before Instruction

PC	=	address `(HERE)`

After Instruction

If FLAG<1>	=	0;
PC	=	address `(FALSE)`
If FLAG<1>	=	1;
PC	=	address `(TRUE)`

BTG — Bit Toggle f

Syntax:	[*label*] BTG f,b[,a]
Operands:	$0 \leq f \leq 255$ $0 \leq b \leq 7$ $a \in [0,1]$
Operation:	$(\overline{f<b>}) \rightarrow f<b>$
Status Affected:	None
Encoding:	`0111` `bbba` `ffff` `ffff`
Description:	Bit 'b' in data memory location 'f' is inverted. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `BTG PORTC, 4, 0`

Before Instruction:
PORTC = `0111 0101` [0x75]

After Instruction:
PORTC = `0110 0101` [0x65]

BOV — Branch if Overflow

Syntax:	[*label*] BOV n
Operands:	$-128 \leq n \leq 127$
Operation:	if overflow bit is ‘1’ $(PC) + 2 + 2n \rightarrow PC$
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’ 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 Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BOV Jump`

Before Instruction
PC = address `(HERE)`

After Instruction
If Overflow = 1;
PC = address `(Jump)`
If Overflow = 0;
PC = address `(HERE+2)`

BZ **Branch if Zero**

Syntax:	[*label*] BZ n
Operands:	$-128 \leq n \leq 127$
Operation:	if Zero bit is '1' (PC) + 2 + 2n $\rightarrow$ 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' 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 Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	Write to PC
No operation	No operation	No operation	No operation

If No Jump:

Q1	Q2	Q3	Q4
Decode	Read literal 'n'	Process Data	No operation

Example: `HERE BZ Jump`

Before Instruction
PC = address `(HERE)`

After Instruction
If Zero = 1;
PC = address `(Jump)`
If Zero = 0;
PC = address `(HERE+2)`

CALL **Subroutine Call**

Syntax:	[*label*] CALL k [,s]
Operands:	$0 \leq k \leq 1048575$ $s \in [0,1]$
Operation:	(PC) + 4 $\rightarrow$ TOS, k $\rightarrow$ PC<20:1>, if s = 1 (W) $\rightarrow$ WS, (STATUS) $\rightarrow$ STATUSS, (BSR) $\rightarrow$ 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 2 Mbyte memory range. First, return address (PC+ 4) is pushed onto the return stack. If 's' = 1, the W, STATUS and BSR registers are also pushed into their respective shadow registers, WS, STATUSS and BSRS. If 's' = 0, no update occurs (default). 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:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	Push PC to stack	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example: `HERE CALL THERE,1`

Before Instruction
PC = address `(HERE)`

After Instruction
PC = address `(THERE)`
TOS = address `(HERE + 4)`
WS = W
BSRS = BSR
STATUSS= STATUS

CLRF	**Clear f**
Syntax:	[*label*] CLRF f [,a]
Operands:	0 ≤ f ≤ 255 a ∈ [0,1]
Operation:	000h → f 1 → Z
Status Affected:	Z
Encoding:	`0110` | `101a` | `ffff` | `ffff`
Description:	Clears the contents of the specified register. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `CLRF FLAG_REG,1`

Before Instruction

FLAG_REG = 0x5A

After Instruction

FLAG_REG = 0x00

CLRWDT	**Clear Watchdog Timer**
Syntax:	[*label*] CLRWDT
Operands:	None
Operation:	000h → WDT, 000h → WDT postscaler, 1 → $\overline{TO}$, 1 → $\overline{PD}$
Status Affected:	$\overline{TO}$, $\overline{PD}$
Encoding:	`0000` | `0000` | `0000` | `0100`
Description:	`CLRWDT` instruction resets the Watchdog Timer. It also resets the postscaler 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	No operation

Example: `CLRWDT`

Before Instruction

WDT Counter = ?

After Instruction

WDT Counter = 0x00
WDT Postscaler = 0
$\overline{TO}$ = 1
$\overline{PD}$ = 1

COMF Complement f

Syntax:	[*label*] COMF f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(\overline{f}) \rightarrow$ dest
Status Affected:	N, Z
Encoding:	`0001` `11da` `ffff` `ffff`
Description:	The contents of register 'f' are complemented. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `COMF REG, 0, 0`

Before Instruction
REG = 0x13
After Instruction
REG = 0x13
W = 0xEC

CPFSEQ Compare f with W, skip if f = W

Syntax:	[*label*] CPFSEQ f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	(f) – (W), skip if (f) = (W) (unsigned comparison)
Status Affected:	None
Encoding:	`0110` `001a` `ffff` `ffff`
Description:	Compares the contents of data memory location 'f' to the contents of W by performing an unsigned subtraction. If 'f' = W, then the fetched instruction is discarded and a `NOP` is executed instead, making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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      CPFSEQ REG, 0
NEQUAL    :
EQUAL     :
```

Before Instruction
PC Address = `HERE`
W = ?
REG = ?
After Instruction
If REG = W;
PC = Address `(EQUAL)`
If REG ≠ W;
PC = Address `(NEQUAL)`

CPFSGT	Compare f with W, skip if f > W
Syntax:	[*label*] CPFSGT f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	(f) – (W), skip if (f) > (W) (unsigned comparison)
Status Affected:	None
Encoding:	`0110` `010a` `ffff` `ffff`
Description:	Compares the contents of data memory location 'f' to the contents of the W by performing an unsigned subtraction. If the contents of 'f' are greater than the contents of WREG, then the fetched instruction is discarded and a `NOP` is executed instead, making this a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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      CPFSGT REG, 0
NGREATER  :
GREATER   :
```

Before Instruction

PC	=	Address `(HERE)`
W	=	?

After Instruction

If REG	>	W;
PC	=	Address `(GREATER)`
If REG	≤	W;
PC	=	Address `(NGREATER)`

CPFSLT	Compare f with W, skip if f < W
Syntax:	[*label*] CPFSLT f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	(f) – (W), skip if (f) < (W) (unsigned comparison)
Status Affected:	None
Encoding:	`0110` `000a` `ffff` `ffff`
Description:	Compares the contents of data memory location 'f' to the contents of W by performing an unsigned subtraction. If the contents of 'f' are less than the contents of W, then the fetched instruction is discarded and a `NOP` is executed instead, making this a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected. If ‘a’ is 1, the BSR will not be overridden (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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    CPFSLT REG, 1
NLESS   :
LESS    :
```

Before Instruction

PC	=	Address `(HERE)`
W	=	?

After Instruction

If REG	<	W;
PC	=	Address `(LESS)`
If REG	≥	W;
PC	=	Address `(NLESS)`

DAW **Decimal Adjust W Register**

Syntax:	[*label*] DAW
Operands:	None
Operation:	If [W<3:0> >9] or [DC = 1] then (W<3:0>) + 6 → W<3:0>; else (W<3:0>) → W<3:0>; If [W<7:4> >9] or [C = 1] then (W<7:4>) + 6 → W<7:4>; else (W<7:4>) → W<7:4>;
Status Affected:	C
Encoding:	`0000` `0000` `0000` `0111`
Description:	`DAW` adjusts the eight-bit value in W, 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 W	Process Data	Write W

Example1: `DAW`

Before Instruction

```
W   =  0xA5
C   =  0
DC  =  0
```

After Instruction

```
W   =  0x05
C   =  1
DC  =  0
```

Example 2:

Before Instruction

```
W   =  0xCE
C   =  0
DC  =  0
```

After Instruction

```
W   =  0x34
C   =  1
DC  =  0
```

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 → dest
Status Affected:	C, DC, N, OV, Z
Encoding:	`0000` `01da` `ffff` `ffff`
Description:	Decrement register 'f'. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `DECF    CNT,   1, 0`

Before Instruction

```
CNT  =  0x01
Z    =  0
```

After Instruction

```
CNT  =  0x00
Z    =  1
```

DECFSZ **Decrement f, skip if 0**

Syntax:	[*label*] DECFSZ f [,d [,a]]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(f) – 1 → dest, skip if result = 0
Status Affected:	None
Encoding:	`0010` `11da` `ffff` `ffff`
Description:	The contents of register 'f' are decremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If the result is 0, the next instruction, which is already fetched, is discarded, and a `NOP` is executed instead, making it a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

If skip:

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
            GOTO      LOOP
CONTINUE
```

Before Instruction
```
PC      =   Address (HERE)
```
After Instruction
```
CNT     =   CNT - 1
If CNT  =   0;
    PC  =   Address (CONTINUE)
If CNT  ≠   0;
    PC  =   Address (HERE+2)
```

DCFSNZ **Decrement f, skip if not 0**

Syntax:	[*label*] DCFSNZ f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(f) – 1 → dest, skip if result ≠ 0
Status Affected:	None
Encoding:	`0100` `11da` `ffff` `ffff`
Description:	The contents of register 'f' are decremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If the result is not 0, the next instruction, which is already fetched, is discarded, and a `NOP` is executed instead, making it a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

If skip:

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  TEMP, 1, 0
ZERO    :
NZERO   :
```

Before Instruction
```
TEMP        =   ?
```
After Instruction
```
TEMP        =   TEMP - 1,
If TEMP     =   0;
    PC      =   Address (ZERO)
If TEMP     ≠   0;
    PC      =   Address (NZERO)
```

GOTO **Unconditional Branch**

Syntax:	[*label*] GOTO k
Operands:	$0 \le k \le 1048575$
Operation:	$k \rightarrow$ PC<20:1>
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 entire 2 Mbyte memory range. The 20-bit value 'k' is loaded into PC<20:1>. `GOTO` is always a two-cycle instruction.

Words: 2

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'<7:0>,	No operation	Read literal 'k'<19:8>, Write to PC
No operation	No operation	No operation	No operation

Example: `GOTO THERE`

After Instruction

```
PC  =   Address (THERE)
```

INCF **Increment f**

Syntax:	[*label*] INCF f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(f) + 1 \rightarrow$ dest
Status Affected:	C, DC, N, OV, Z

Encoding:

`0010`	`10da`	`ffff`	`ffff`

Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `INCF     CNT, 1, 0`

Before Instruction

```
CNT   =   0xFF
Z     =   0
C     =   ?
DC    =   ?
```

After Instruction

```
CNT   =   0x00
Z     =   1
C     =   1
DC    =   1
```

INCFSZ Increment f, skip if 0

Syntax:	[*label*] INCFSZ f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(f) + 1 → dest, skip if result = 0
Status Affected:	None
Encoding:	`0011 11da ffff ffff`
Description:	The contents of register 'f' are incremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f'. (default) If the result is 0, the next instruction, which is already fetched, is discarded, and a `NOP` is executed instead, making it a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

If skip:

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     INCFSZ   CNT, 1, 0
NZERO    :
ZERO     :
```

Before Instruction

PC	=	Address `(HERE)`

After Instruction

CNT	=	CNT + 1
If CNT	=	0;
PC	=	Address `(ZERO)`
If CNT	≠	0;
PC	=	Address `(NZERO)`

INFSNZ Increment f, skip if not 0

Syntax:	[*label*] INFSNZ f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(f) + 1 → dest, skip if result ≠ 0
Status Affected:	None
Encoding:	`0100 10da ffff ffff`
Description:	The contents of register 'f' are incremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If the result is not 0, the next instruction, which is already fetched, is discarded, and a `NOP` is executed instead, making it a two-cycle instruction. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

If skip:

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     INFSNZ  REG, 1, 0
ZERO
NZERO
```

Before Instruction

PC	=	Address `(HERE)`

After Instruction

REG	=	REG + 1
If REG	≠	0;
PC	=	Address `(NZERO)`
If REG	=	0;
PC	=	Address `(ZERO)`

IORLW **Inclusive OR literal with W**

Syntax:	[*label*] IORLW k
Operands:	$0 \le k \le 255$
Operation:	(W) .OR. k $\rightarrow$ W
Status Affected:	N, Z
Encoding:	`0000` `1001` `kkkk` `kkkk`
Description:	The contents of W are OR'ed with the eight-bit literal 'k'. The result is placed in W.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

Example: `IORLW 0x35`

Before Instruction

W	=	0x9A

After Instruction

W	=	0xBF

IORWF **Inclusive OR W with f**

Syntax:	[*label*] IORWF f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(W) .OR. (f) $\rightarrow$ dest
Status Affected:	N, Z
Encoding:	`0001` `00da` `ffff` `ffff`
Description:	Inclusive OR W with register 'f'. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, over-riding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `IORWF RESULT, 0, 1`

Before Instruction

RESULT	=	0x13
W	=	0x91

After Instruction

RESULT	=	0x13
W	=	0x93

LFSR **Load FSR**

Syntax:	[*label*] LFSR f,k
Operands:	0 ≤ f ≤ 2 0 ≤ k ≤ 4095
Operation:	k → FSRf
Status Affected:	None

Encoding:

`1110`	`1110`	`00ff`	k_{11}`kkk`
`1111`	`0000`	k_7`kkk`	`kkkk`

Description:	The 12-bit literal 'k' is loaded into the file select register pointed to by 'f'.
Words:	2
Cycles:	2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k' MSB	Process Data	Write literal 'k' MSB to FSRfH
Decode	Read literal 'k' LSB	Process Data	Write literal 'k' to FSRfL

Example: `LFSR 2, 0x3AB`

After Instruction

FSR2H	=	0x03
FSR2L	=	0xAB

MOVF **Move f**

Syntax:	[*label*] MOVF f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	f → dest
Status Affected:	N, Z

Encoding:

`0101`	`00da`	`ffff`	`ffff`

Description:	The contents of register 'f' are moved to a destination dependent upon the status of 'd'. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). Location 'f' can be anywhere in the 256 byte bank. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write W

Example: `MOVF REG, 0, 0`

Before Instruction

REG	=	0x22
W	=	0xFF

After Instruction

REG	=	0x22
W	=	0x22

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`$_s$
2nd word (destin.)	`1111`	`ffff`	`ffff`	`ffff`$_d$

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.
Either source or destination can be W (a useful special situation).
`MOVFF` is particularly useful for transferring a data memory location to a peripheral register (such as the transmit buffer or an I/O port).

The `MOVFF` instruction cannot use the PCL, TOSU, TOSH or TOSL as the destination register.

Note: The `MOVFF` instruction should not be used to modify interrupt settings while any interrupt is enabled. See Section 8.0 for more information.

Words:	2
Cycles:	2 (3)

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f' (src)	Process Data	No operation
Decode	No operation No dummy read	No operation	Write register 'f' (dest)

Example: `MOVFF REG1, REG2`

Before Instruction

REG1	=	0x33
REG2	=	0x11

After Instruction

REG1	=	0x33,
REG2	=	0x33

MOVLB Move literal to low nibble in BSR

Syntax:	[*label*] MOVLB k
Operands:	$0 \le k \le 255$
Operation:	$k \rightarrow$ BSR
Status Affected:	None
Encoding:	`0000` `0001` `kkkk` `kkkk`
Description:	The 8-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: `MOVLB 5`

Before Instruction

BSR register	=	0x02

After Instruction

BSR register	=	0x05

MOVLW **Move literal to W**

Syntax:	[*label*] MOVLW k
Operands:	$0 \leq k \leq 255$
Operation:	$k \rightarrow W$
Status Affected:	None

Encoding:

`0000`	`1110`	`kkkk`	`kkkk`

Description:	The eight-bit literal 'k' is loaded into W.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

Example: `MOVLW 0x5A`

After Instruction

W	=	0x5A

MOVWF **Move W to f**

Syntax:	[*label*] MOVWF f [,a]
Operands:	$0 \leq f \leq 255$ $a \in [0,1]$
Operation:	$(W) \rightarrow f$
Status Affected:	None

Encoding:

`0110`	`111a`	`ffff`	`ffff`

Description:	Move data from W to register 'f'. Location 'f' can be anywhere in the 256 byte bank. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `MOVWF REG, 0`

Before Instruction

W	=	0x4F
REG	=	0xFF

After Instruction

W	=	0x4F
REG	=	0x4F

MULLW **Multiply Literal with W**

Syntax:	[*label*] MULLW k
Operands:	$0 \le k \le 255$
Operation:	(W) x k $\rightarrow$ PRODH:PRODL
Status Affected:	None
Encoding:	`0000` `1101` `kkkk` `kkkk`
Description:	An unsigned multiplication is carried out between the contents of W and the 8-bit literal 'k'. The 16-bit result is placed in PRODH:PRODL register pair. PRODH contains the high byte. W is unchanged. None of the status flags are affected. Note that neither 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: `MULLW 0xC4`

Before Instruction

W	=	0xE2
PRODH	=	?
PRODL	=	?

After Instruction

W	=	0xE2
PRODH	=	0xAD
PRODL	=	0x08

MULWF **Multiply W with f**

Syntax:	[*label*] MULWF f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	(W) x (f) $\rightarrow$ PRODH:PRODL
Status Affected:	None
Encoding:	`0000` `001a` `ffff` `ffff`
Description:	An unsigned multiplication is carried out between the contents of W and the register file location 'f'. The 16-bit result is stored in the PRODH:PRODL register pair. PRODH contains the high byte. Both W and 'f' are unchanged. None of the status flags are affected. Note that neither overflow nor carry is possible in this operation. A zero result is possible but not detected. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write registers PRODH: PRODL

Example: `MULWF REG, 1`

Before Instruction

W	=	0xC4
REG	=	0xB5
PRODH	=	?
PRODL	=	?

After Instruction

W	=	0xC4
REG	=	0xB5
PRODH	=	0x8A
PRODL	=	0x94

NEGF **Negate f**

Syntax:	[*label*] NEGF f [,a]
Operands:	$0 \leq f \leq 255$ $a \in [0,1]$
Operation:	$(\overline{f}) + 1 \rightarrow f$
Status Affected:	N, OV, C, DC, Z

Encoding:

`0110`	`110a`	`ffff`	`ffff`

Description:	Location ‘f’ is negated using two’s complement. The result is placed in the data memory location 'f'. If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `NEGF REG, 1`

Before Instruction
REG = `0011 1010` [0x3A]
After Instruction
REG = `1100 0110` [0xC6]

NOP **No Operation**

Syntax:	[*label*] NOP
Operands:	None
Operation:	No operation
Status Affected:	None

Encoding:

`0000`
`1111`
	`0000`
	`xxxx`
		`0000`
		`xxxx`
			`0000`
			`xxxx`

Description:	No operation.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation

Example:

None.

POP — Pop Top of Return Stack

Syntax:	[*label*] POP
Operands:	None
Operation:	(TOS) $\rightarrow$ bit bucket
Status Affected:	None

Encoding:

0000	0000	0000	0110

Description:	The 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 properly manage the return stack to incorporate a software stack.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	POP TOS value	No operation

Example:
```
POP
GOTO    NEW
```

Before Instruction

TOS	=	0031A2h
Stack (1 level down)	=	014332h

After Instruction

TOS	=	014332h
PC	=	NEW

PUSH — Push Top of Return Stack

Syntax:	[*label*] PUSH
Operands:	None
Operation:	(PC+2) $\rightarrow$ TOS
Status Affected:	None

Encoding:

0000	0000	0000	0101

Description:	The PC+2 is pushed onto the top of the return stack. The previous TOS value is pushed down on the stack. This instruction allows to implement a software stack by modifying TOS, and then push it onto the return stack.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	PUSH PC+2 onto return stack	No operation	No operation

Example: `PUSH`

Before Instruction

TOS	=	00345Ah
PC	=	000124h

After Instruction

PC	=	000126h
TOS	=	000126h
Stack (1 level down)	=	00345Ah

RCALL **Relative Call**

Syntax:	[*label*] RCALL n
Operands:	$-1024 \leq n \leq 1023$
Operation:	$(PC) + 2 \rightarrow TOS$, $(PC) + 2 + 2n \rightarrow PC$
Status Affected:	None
Encoding:	`1101` | `1nnn` | `nnnn` | `nnnn`
Description:	Subroutine call with a jump up to 1K from the current location. First, return address (PC+2) is pushed onto the stack. Then, add the 2's complement number '2n' 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: `HERE RCALL Jump`

Before Instruction

PC = Address `(HERE)`

After Instruction

PC = Address `(Jump)`
TOS = Address `(HERE+2)`

RESET **Reset**

Syntax:	[*label*] RESET
Operands:	None
Operation:	Reset all registers and flags that are affected by a $\overline{\text{MCLR}}$ Reset.
Status Affected:	All
Encoding:	`0000` | `0000` | `1111` | `1111`
Description:	This instruction provides a way to execute a $\overline{\text{MCLR}}$ Reset in software.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Start reset	No operation	No operation

Example: `RESET`

After Instruction

Registers = Reset Value
Flags* = Reset Value

RETFIE	**Return from Interrupt**
Syntax:	[*label*] RETFIE [s]
Operands:	s ∈ [0,1]
Operation:	(TOS) → PC, 1 → GIE/GIEH or PEIE/GIEL, if s = 1 (WS) → W, (STATUSS) → STATUS, (BSRS) → BSR, 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 bit. If 's' = 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, W, STATUS and BSR. If 's' = 0, no update of these registers occurs (default).
Words:	1
Cycles:	2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	pop PC from stack Set GIEH or GIEL
No operation	No operation	No operation	No operation

Example: `RETFIE  1`

After Interrupt

PC	=	TOS
W	=	WS
BSR	=	BSRS
STATUS	=	STATUSS
GIE/GIEH, PEIE/GIEL	=	1

RETLW	**Return Literal to W**
Syntax:	[*label*] RETLW k
Operands:	0 ≤ k ≤ 255
Operation:	k → W, (TOS) → PC, PCLATU, PCLATH are unchanged
Status Affected:	None
Encoding:	`0000 1100 kkkk kkkk`
Description:	W is loaded with the eight-bit literal 'k'. The program counter is loaded from the top of the stack (the return address). The high address latch (PCLATH) remains unchanged.
Words:	1
Cycles:	2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	pop PC from stack, Write to W
No operation	No operation	No operation	No operation

Example:

```
    CALL TABLE ; W contains table
               ; offset value
               ; W now has
               ; table value
   :
TABLE
    ADDWF PCL  ; W = offset
    RETLW k0   ; Begin table
    RETLW k1   ;
   :
   :
    RETLW kn   ; End of table
```

Before Instruction

W	=	0x07

After Instruction

W	=	value of kn

RETURN **Return from Subroutine**

Syntax:	[*label*] RETURN [s]
Operands:	s ∈ [0,1]
Operation:	(TOS) → PC, if s = 1 (WS) → W, (STATUSS) → STATUS, (BSRS) → BSR, PCLATU, PCLATH are unchanged
Status Affected:	None
Encoding:	`0000` `0000` `0001` `001s`
Description:	Return from subroutine. The stack is popped and the top of the stack (TOS) is loaded into the program counter. If ‘s’= 1, the contents of the shadow registers WS, STATUSS and BSRS are loaded into their corresponding registers, W, STATUS and BSR. If ‘s’ = 0, no update of these registers occurs (default).
Words:	1
Cycles:	2

Q Cycle Activity:

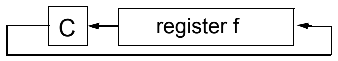
Q1	Q2	Q3	Q4
Decode	No operation	Process Data	pop PC from stack
No operation	No operation	No operation	No operation

Example: `RETURN`

After Interrupt
PC = TOS

RLCF **Rotate Left f through Carry**

Syntax:	[*label*] RLCF f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(f<n>) → dest<n+1>, (f<7>) → C, (C) → dest<0>
Status Affected:	C, N, Z
Encoding:	`0011` `01da` `ffff` `ffff`
Description:	The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is stored back in register 'f' (default). If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ = 1, then the bank will be selected as per the BSR value (default).



Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `RLCF REG, 0, 0`

Before Instruction

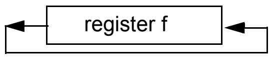
REG	=	`1110 0110`
C	=	0

After Instruction

REG	=	`1110 0110`
W	=	`1100 1100`
C	=	1

RLNCF **Rotate Left f (no carry)**

Syntax:	[*label*] RLNCF f [,d [,a]
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(f<n>) \rightarrow dest<n+1>$, $(f<7>) \rightarrow dest<0>$
Status Affected:	N, Z
Encoding:	`0100` | `01da` | `ffff` | `ffff`
Description:	The contents of register 'f' are rotated one bit to the left. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).



Words:	1
Cycles:	1

Q Cycle Activity:

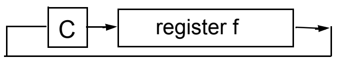
Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

<u>Example</u>: `RLNCF REG, 1, 0`

Before Instruction
REG = `1010 1011`
After Instruction
REG = `0101 0111`

RRCF **Rotate Right f through Carry**

Syntax:	[*label*] RRCF f [,d [,a]
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(f<n>) \rightarrow dest<n-1>$, $(f<0>) \rightarrow C$, $(C) \rightarrow dest<7>$
Status Affected:	C, N, Z
Encoding:	`0011` | `00da` | `ffff` | `ffff`
Description:	The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).



Words:	1
Cycles:	1

Q Cycle Activity:

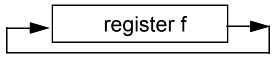
Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

<u>Example</u>: `RRCF REG, 0, 0`

Before Instruction
REG = `1110 0110`
C = 0
After Instruction
REG = `1110 0110`
W = `0111 0011`
C = 0

RRNCF **Rotate Right f (no carry)**

Syntax:	[*label*] RRNCF f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(f<n>) $\rightarrow$ dest<n-1>, (f<0>) $\rightarrow$ dest<7>
Status Affected:	N, Z
Encoding:	`0100` `00da` `ffff` `ffff`
Description:	The contents of register 'f' are rotated one bit to the right. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).



Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example 1: `RRNCF REG, 1, 0`

Before Instruction
REG = `1101 0111`

After Instruction
REG = `1110 1011`

Example 2: `RRNCF REG, 0, 0`

Before Instruction
W = ?
REG = `1101 0111`

After Instruction
W = `1110 1011`
REG = `1101 0111`

SETF **Set f**

Syntax:	[*label*] SETF f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	FFh $\rightarrow$ f
Status Affected:	None
Encoding:	`0110` `100a` `ffff` `ffff`
Description:	The contents of the specified register are set to FFh. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write register 'f'

Example: `SETF REG,1`

Before Instruction
REG = 0x5A

After Instruction
REG = 0xFF

SLEEP **Enter SLEEP mode**

Syntax:	[*label*] SLEEP
Operands:	None
Operation:	00h → WDT, 0 → WDT postscaler, 1 → $\overline{\text{TO}}$, 0 → $\overline{\text{PD}}$
Status Affected:	$\overline{\text{TO}}$, $\overline{\text{PD}}$
Encoding:	`0000` `0000` `0000` `0011`
Description:	The power-down status bit ($\overline{\text{PD}}$) is cleared. The time-out status bit ($\overline{\text{TO}}$) is set. Watchdog Timer and its postscaler 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	Process Data	Go to sleep

Example: `SLEEP`

Before Instruction

$\overline{\text{TO}}$	=	?
$\overline{\text{PD}}$	=	?

After Instruction

$\overline{\text{TO}}$	=	1 †
$\overline{\text{PD}}$	=	0

† If WDT causes wake-up, this bit is cleared.

SUBFWB **Subtract f from W with borrow**

Syntax:	[*label*] SUBFWB f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(W) – (f) – ($\overline{\text{C}}$) → dest
Status Affected:	N, OV, C, DC, Z
Encoding:	`0101` `01da` `ffff` `ffff`
Description:	Subtract register 'f' and carry flag (borrow) from W (2’s complement method). If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored in register 'f' (default). If ‘a’ is 0, the Access Bank will be selected, overriding the BSR value. If ‘a’ is 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example 1: `SUBFWB REG, 1, 0`

Before Instruction

REG	=	3	
W	=	2	
C	=	1	

After Instruction

REG	=	FF	
W	=	2	
C	=	0	
Z	=	0	
N	=	1	; result is negative

Example 2: `SUBFWB REG, 0, 0`

Before Instruction

REG	=	2	
W	=	5	
C	=	1	

After Instruction

REG	=	2	
W	=	3	
C	=	1	
Z	=	0	
N	=	0	; result is positive

Example 3: `SUBFWB REG, 1, 0`

Before Instruction

REG	=	1	
W	=	2	
C	=	0	

After Instruction

REG	=	0	
W	=	2	
C	=	1	
Z	=	1	; result is zero
N	=	0	

SUBLW **Subtract W from literal**

Syntax:	[*label*] SUBLW k
Operands:	0 ≤ k ≤ 255
Operation:	k – (W) → W
Status Affected:	N, OV, C, DC, Z
Encoding:	`0000` `1000` `kkkk` `kkkk`
Description:	W is subtracted from the eight-bit literal 'k'. The result is placed in W.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

Example 1: `SUBLW 0x02`

Before Instruction

```
W   =  1
C   =  ?
```

After Instruction

```
W   =  1
C   =  1    ; result is positive
Z   =  0
N   =  0
```

Example 2: `SUBLW 0x02`

Before Instruction

```
W   =  2
C   =  ?
```

After Instruction

```
W   =  0
C   =  1    ; result is zero
Z   =  1
N   =  0
```

Example 3: `SUBLW 0x02`

Before Instruction

```
W   =  3
C   =  ?
```

After Instruction

```
W   =  FF   ; (2's complement)
C   =  0   ; result is negative
Z   =  0
N   =  1
```

SUBWF **Subtract W from f**

Syntax:	[*label*] SUBWF f [,d [,a]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(f) – (W) → dest
Status Affected:	N, OV, C, DC, Z
Encoding:	`0101` `11da` `ffff` `ffff`
Description:	Subtract W from register 'f' (2's complement method). If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example 1: `SUBWF REG, 1, 0`

Before Instruction

```
REG  =  3
W    =  2
C    =  ?
```

After Instruction

```
REG  =  1
W    =  2
C    =  1    ; result is positive
Z    =  0
N    =  0
```

Example 2: `SUBWF REG, 0, 0`

Before Instruction

```
REG  =  2
W    =  2
C    =  ?
```

After Instruction

```
REG  =  2
W    =  0
C    =  1    ; result is zero
Z    =  1
N    =  0
```

Example 3: `SUBWF REG, 1, 0`

Before Instruction

```
REG  =  1
W    =  2
C    =  ?
```

After Instruction

```
REG  =  FFh   ;(2's complement)
W    =  2
C    =  0    ; result is negative
Z    =  0
N    =  1
```

SUBWFB Subtract W from f with Borrow

Syntax:	[*label*] SUBWFB f [,d [,a]
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(f) - (W) - (\overline{C}) \rightarrow$ dest
Status Affected:	N, OV, C, DC, Z

Encoding:

`0101`	`10da`	`ffff`	`ffff`

Description: Subtract W and the carry flag (borrow) from register 'f' (2's complement method). If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example 1: `SUBWFB REG, 1, 0`

```
Before Instruction
    REG   =   0x19    (0001 1001)
    W     =   0x0D    (0000 1101)
    C     =   1
After Instruction
    REG   =   0x0C    (0000 1011)
    W     =   0x0D    (0000 1101)
    C     =   1
    Z     =   0
    N     =   0       ; result is positive
```

Example 2: `SUBWFB REG, 0, 0`

```
Before Instruction
    REG   =   0x1B    (0001 1011)
    W     =   0x1A    (0001 1010)
    C     =   0
After Instruction
    REG   =   0x1B    (0001 1011)
    W     =   0x00
    C     =   1
    Z     =   1       ; result is zero
    N     =   0
```

Example 3: `SUBWFB REG, 1, 0`

```
Before Instruction
    REG   =   0x03    (0000 0011)
    W     =   0x0E    (0000 1101)
    C     =   1
After Instruction
    REG   =   0xF5    (1111 0100)
                      ; [2's comp]
    W     =   0x0E    (0000 1101)
    C     =   0
    Z     =   0
    N     =   1       ; result is negative
```

SWAPF Swap f

Syntax:	[*label*] SWAPF f [,d [,a]
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(f<3:0>) $\rightarrow$ dest<7:4>, (f<7:4>) $\rightarrow$ dest<3:0>
Status Affected:	None

Encoding:

`0011`	`10da`	`ffff`	`ffff`

Description: The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed in register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `SWAPF REG, 1, 0`

```
Before Instruction
    REG   =   0x53
After Instruction
    REG   =   0x35
```

TBLRD Table Read

Syntax:	[*label*] TBLRD (*; *+; *-; +*)
Operands:	None

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	10nn nn=0 * =1 *+ =2 *- =3 +*

Description: This instruction is used to read the contents of Program Memory (P.M.). 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.

TBLPTR[0] = 0:	Least Significant Byte of Program Memory Word
TBLPTR[0] = 1:	Most Significant Byte of Program Memory Word

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:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation
No operation	No operation (Read Program Memory)	No operation	No operation (Write TABLAT)

TBLRD Table Read (cont'd)

Example1: `TBLRD *+ ;`

Before Instruction

TABLAT	=	0x55
TBLPTR	=	0x00A356
MEMORY(0x00A356)	=	0x34

After Instruction

TABLAT	=	0x34
TBLPTR	=	0x00A357

Example2: `TBLRD +* ;`

Before Instruction

TABLAT	=	0xAA
TBLPTR	=	0x01A357
MEMORY(0x01A357)	=	0x12
MEMORY(0x01A358)	=	0x34

After Instruction

TABLAT	=	0x34
TBLPTR	=	0x01A358

TBLWT Table Write

Syntax:	[*label*] TBLWT (*; *+; *-; +*)
Operands:	None

Operation:	if TBLWT*,
	(TABLAT) → Holding Register;
	TBLPTR - No Change;
	if TBLWT*+,
	(TABLAT) → Holding Register;
	(TBLPTR) +1 → TBLPTR;
	if TBLWT*-,
	(TABLAT) → Holding Register;
	(TBLPTR) -1 → TBLPTR;
	if TBLWT+*,
	(TBLPTR) +1 → TBLPTR;
	(TABLAT) → Holding Register;

Status Affected:	None

Encoding:

0000	0000	0000	11nn nn=0 * =1 *+ =2 *- =3 +*

Description: This instruction uses the 3 LSbs of the TBLPTR to determine which of the 8 holding registers the TABLAT data is written to. The 8 holding registers are used to program the contents of Program Memory (P.M.). See Section 5.0 for information on writing to FLASH memory.
The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2 MBtye 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 `TBLWT` instruction can modify the value of TBLPTR as follows:

- no change
- post-increment
- post-decrement
- pre-increment

Words: 1

Cycles: 2

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	No operation	No operation	No operation
No operation	No operation (Read TABLAT)	No operation	No operation (Write to Holding Register or Memory)

TBLWT Table Write (Continued)

Example1: `TBLWT *+;`

Before Instruction

TABLAT	=	0x55
TBLPTR	=	0x00A356
HOLDING REGISTER (0x00A356)	=	0xFF

After Instructions (table write completion)

TABLAT	=	0x55
TBLPTR	=	0x00A357
HOLDING REGISTER (0x00A356)	=	0x55

Example 2: `TBLWT +*;`

Before Instruction

TABLAT	=	0x34
TBLPTR	=	0x01389A
HOLDING REGISTER (0x01389A)	=	0xFF
HOLDING REGISTER (0x01389B)	=	0xFF

After Instruction (table write completion)

TABLAT	=	0x34
TBLPTR	=	0x01389B
HOLDING REGISTER (0x01389A)	=	0xFF
HOLDING REGISTER (0x01389B)	=	0x34

TSTFSZ Test f, skip if 0

Syntax:	[*label*] TSTFSZ f [,a]
Operands:	0 ≤ f ≤ 255 a ∈ [0,1]
Operation:	skip if f = 0
Status Affected:	None
Encoding:	`0110` `011a` `ffff` `ffff`
Description:	If 'f' = 0, the next instruction, fetched during the current instruction execution, is discarded and a `NOP` is executed, making this a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1(2) **Note:** 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	No operation

If skip:

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

<u>Example</u>:

```
HERE    TSTFSZ  CNT, 1
NZERO     :
ZERO    :
```

Before Instruction
PC = Address `(HERE)`

After Instruction

If CNT	=	0x00,
PC	=	Address `(ZERO)`
If CNT	≠	0x00,
PC	=	Address `(NZERO)`

XORLW Exclusive OR literal with W

Syntax:	[*label*] XORLW k
Operands:	0 ≤ k ≤ 255
Operation:	(W) .XOR. k → W
Status Affected:	N, Z
Encoding:	`0000` `1010` `kkkk` `kkkk`
Description:	The contents of W are XORed with the 8-bit literal 'k'. The result is placed in W.
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read literal 'k'	Process Data	Write to W

<u>Example</u>: `XORLW 0xAF`

Before Instruction
W = 0xB5

After Instruction
W = 0x1A

XORWF	**Exclusive OR W with f**
Syntax:	[*label*] XORWF f [,d [,a]
Operands:	$0 \leq f \leq 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(W) .XOR. (f) $\rightarrow$ dest
Status Affected:	N, Z
Encoding:	`0001` `10da` `ffff` `ffff`
Description:	Exclusive OR the contents of W with register 'f'. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in the register 'f' (default). If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' is 1, then the bank will be selected as per the BSR value (default).
Words:	1
Cycles:	1

Q Cycle Activity:

Q1	Q2	Q3	Q4
Decode	Read register 'f'	Process Data	Write to destination

Example: `XORWF REG, 1, 0`

Before Instruction

REG	=	0xAF
W	=	0xB5

After Instruction

REG	=	0x1A
W	=	0xB5

22.0 DEVELOPMENT SUPPORT

The PICmicro® microcontrollers are supported with a full range of hardware and software development tools:

- Integrated Development Environment
  - MPLAB® IDE Software
- Assemblers/Compilers/Linkers
  - MPASM™ Assembler
  - MPLAB C17 and MPLAB C18 C Compilers
  - MPLINK™ Object Linker/ MPLIB™ Object Librarian
- Simulators
  - MPLAB SIM Software Simulator
- Emulators
  - MPLAB ICE 2000 In-Circuit Emulator
  - ICEPIC™ In-Circuit Emulator
- In-Circuit Debugger
  - MPLAB ICD
- Device Programmers
  - PRO MATE® II Universal Device Programmer
  - PICSTART® Plus Entry-Level Development Programmer
- Low Cost Demonstration Boards
  - PICDEM™ 1 Demonstration Board
  - PICDEM 2 Demonstration Board
  - PICDEM 3 Demonstration Board
  - PICDEM 17 Demonstration Board
  - KEELOQ® Demonstration Board

22.1 MPLAB Integrated Development Environment Software

The MPLAB IDE software brings an ease of software development previously unseen in the 8-bit microcontroller market. The MPLAB IDE is a Windows® based application that contains:

- An interface to debugging tools
  - simulator
  - programmer (sold separately)
  - emulator (sold separately)
  - in-circuit debugger (sold separately)
- A full-featured editor
- A project manager
- Customizable toolbar and key mapping
- A status bar
- On-line help

The MPLAB IDE allows you to:

- Edit your source files (either assembly or 'C')
- One touch assemble (or compile) and download to PICmicro emulator and simulator tools (automatically updates all project information)
- Debug using:
  - source files
  - absolute listing file
  - machine code

The ability to use MPLAB IDE with multiple debugging tools allows users to easily switch from the cost-effective simulator to a full-featured emulator with minimal retraining.

22.2 MPASM Assembler

The MPASM assembler is a full-featured universal macro assembler for all PICmicro MCU's.

The MPASM assembler has a command line interface and a Windows shell. It can be used as a stand-alone application on a Windows 3.x or greater system, or it can be used through MPLAB IDE. The MPASM assembler generates relocatable object files for the MPLINK object linker, Intel® standard HEX files, MAP files to detail memory usage and symbol reference, an absolute LST file that contains source lines and generated machine code, and a COD file for debugging.

The MPASM assembler features include:

- Integration into MPLAB IDE projects.
- User-defined macros to streamline assembly code.
- Conditional assembly for multi-purpose source files.
- Directives that allow complete control over the assembly process.

22.3 MPLAB C17 and MPLAB C18 C Compilers

The MPLAB C17 and MPLAB C18 Code Development Systems are complete ANSI 'C' compilers for Microchip's PIC17CXXX and PIC18CXXX family of microcontrollers, respectively. These compilers provide powerful integration capabilities and ease of use not found with other compilers.

For easier source level debugging, the compilers provide symbol information that is compatible with the MPLAB IDE memory display.

22.4 MPLINK Object Linker/ MPLIB Object Librarian

The MPLINK object linker combines relocatable objects created by the MPASM assembler and the MPLAB C17 and MPLAB C18 C compilers. It can also link relocatable objects from pre-compiled libraries, using directives from a linker script.

The MPLIB object librarian is a librarian for pre-compiled code to be used with the MPLINK object linker. 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. The MPLIB object librarian manages the creation and modification of library files.

The MPLINK object linker features include:

- Integration with MPASM assembler and MPLAB C17 and MPLAB C18 C compilers.
- Allows all memory areas to be defined as sections to provide link-time flexibility.

The MPLIB object librarian features include:

- Easier linking because single libraries can be included instead of many smaller files.
- Helps keep code maintainable by grouping related modules together.
- Allows libraries to be created and modules to be added, listed, replaced, deleted or extracted.

22.5 MPLAB SIM Software Simulator

The MPLAB SIM software simulator allows code development in a PC-hosted environment by simulating the PICmicro series microcontrollers on an instruction level. On any given instruction, the data areas can be examined or modified and stimuli can be applied from a file, or user-defined key press, to any of the pins. The execution can be performed in single step, execute until break, or trace mode.

The MPLAB SIM simulator fully supports symbolic debugging using the MPLAB C17 and the MPLAB C18 C compilers and the MPASM assembler. The software simulator offers the flexibility to develop and debug code outside of the laboratory environment, making it an excellent multi-project software development tool.

22.6 MPLAB ICE 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 the MPLAB ICE in-circuit emulator is provided by the MPLAB Integrated Development Environment (IDE), which allows editing, building, downloading and source debugging from a single environment.

The MPLAB ICE 2000 is a full-featured emulator system with enhanced trace, trigger and data monitoring features. Interchangeable processor modules allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the MPLAB ICE in-circuit emulator allows expansion to support new PICmicro microcontrollers.

The MPLAB ICE in-circuit emulator system has been designed as a real-time emulation system, with advanced features that are generally found on more expensive development tools. The PC platform and Microsoft® Windows environment were chosen to best make these features available to you, the end user.

22.7 ICEPIC In-Circuit Emulator

The ICEPIC low cost, in-circuit emulator is a solution for the Microchip Technology PIC16C5X, PIC16C6X, PIC16C7X and PIC16CXXX families of 8-bit One-Time-Programmable (OTP) microcontrollers. The modular system can support different subsets of PIC16C5X or PIC16CXXX products through the use of interchangeable personality modules, or daughter boards. The emulator is capable of emulating without target application circuitry being present.

22.8 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 PICmicro MCUs and can be used to develop for this and other PICmicro microcontrollers. The MPLAB ICD utilizes the in-circuit debugging capability built into the FLASH devices. This feature, along with Microchip's In-Circuit Serial Programming™ protocol, offers cost-effective in-circuit FLASH 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.

22.9 PRO MATE II Universal Device Programmer

The PRO MATE II universal device programmer is a full-featured programmer, capable of operating in stand-alone mode, as well as PC-hosted mode. The PRO MATE II device programmer is CE compliant.

The PRO MATE II device programmer has programmable VDD and VPP supplies, which allow it to verify programmed memory at VDD min and VDD max for maximum reliability. It has an LCD display for instructions and 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 device programmer can read, verify, or program PICmicro devices. It can also set code protection in this mode.

22.10 PICSTART Plus Entry Level Development Programmer

The PICSTART Plus development programmer is an easy-to-use, low cost, prototype programmer. It connects to the PC via a COM (RS-232) port. MPLAB Integrated Development Environment software makes using the programmer simple and efficient.

The PICSTART Plus development programmer supports all PICmicro devices with up to 40 pins. Larger pin count devices, such as the PIC16C92X and PIC17C76X, may be supported with an adapter socket. The PICSTART Plus development programmer is CE compliant.

22.11 PICDEM 1 Low Cost PICmicro Demonstration Board

The PICDEM 1 demonstration board 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, PIC16C8X, PIC17C42, PIC17C43 and PIC17C44. All necessary hardware and software is included to run basic demo programs. The user can program the sample microcontrollers provided with the PICDEM 1 demonstration board on a PRO MATE II device programmer, or a PICSTART Plus development programmer, and easily test firmware. The user can also connect the PICDEM 1 demonstration board to the MPLAB ICE in-circuit emulator and download the firmware to the emulator for testing. A prototype area is available for the user to build some additional hardware and connect it to the microcontroller socket(s). Some of the features include an RS-232 interface, a potentiometer for simulated analog input, push button switches and eight LEDs connected to PORTB.

22.12 PICDEM 2 Low Cost PIC16CXX Demonstration Board

The PICDEM 2 demonstration board is a simple demonstration board that supports the PIC16C62, PIC16C64, PIC16C65, PIC16C73 and PIC16C74 microcontrollers. 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 demonstration board on a PRO MATE II device programmer, or a PICSTART Plus development programmer, and easily test firmware. The MPLAB ICE in-circuit emulator may also be used with the PICDEM 2 demonstration board to test firmware. A prototype area has been provided to the user for adding additional hardware and connecting it to the microcontroller socket(s). 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.

22.13 PICDEM 3 Low Cost PIC16CXXX Demonstration Board

The PICDEM 3 demonstration board is a simple demonstration board that supports the PIC16C923 and PIC16C924 in the PLCC package. It will also support future 44-pin PLCC microcontrollers with 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 demonstration board on a PRO MATE II device programmer, or a PICSTART Plus development programmer with an adapter socket, and easily test firmware. The MPLAB ICE in-circuit emulator may also be used with the PICDEM 3 demonstration board to test firmware. A prototype area has been provided to the user for adding hardware and connecting it to the microcontroller socket(s). Some of the features include a 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 demonstration board is a LCD panel, with 4 commons and 12 segments, that is capable of displaying time, temperature and day of the week. The PICDEM 3 demonstration board provides an additional RS-232 interface and Windows software for showing the demultiplexed LCD signals on a PC. A simple serial interface allows the user to construct a hardware demultiplexer for the LCD signals.

22.14 PICDEM 17 Demonstration Board

The PICDEM 17 demonstration board is an evaluation board that demonstrates the capabilities of several Microchip microcontrollers, including PIC17C752, PIC17C756A, 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 device programmer, or the PICSTART Plus development programmer, and easily debug and test the sample code. In addition, the PICDEM 17 demonstration board supports downloading of programs to and executing out of external FLASH memory on board. The PICDEM 17 demonstration board is also usable with the MPLAB ICE in-circuit emulator, or the PICMASTER emulator and all of the sample programs can be run and modified using either emulator. Additionally, a generous prototype area is available for user hardware.

22.15 KEELOQ Evaluation and Programming Tools

KEELOQ evaluation and programming tools support Microchip's HCS Secure Data Products. The HCS evaluation kit includes a LCD display to show changing codes, a decoder to decode transmissions and a programming interface to program test transmitters.

TABLE 22-1: DEVELOPMENT TOOLS FROM MICROCHIP

		PIC12CXXX	PIC14000	PIC16C5X	PIC16C6X	PIC16CXXX	PIC16F62X	PIC16C7X	PIC16C7XX	PIC16C8X/ PIC16F8X	PIC16F8XX	PIC16C9XX	PIC17C4X	PIC17C7XX	PIC18CXX2	PIC18FXXX	24CXX/ 25CXX/ 93CXX	HCSXXX	MCRFXXX	MCP2510
Software Tools	MPLAB® Integrated Development Environment	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				
	MPLAB® C17 C Compiler												✓	✓						
	MPLAB® C18 C Compiler														✓	✓				
	MPASM™ Assembler/ MPLINK™ Object Linker	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Emulators	MPLAB® ICE In-Circuit Emulator	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓				
	ICEPIC™ In-Circuit Emulator	✓		✓	✓	✓		✓	✓	✓		✓								
Debugger	MPLAB® ICD In-Circuit Debugger				✓*			✓*			✓					✓				
Programmers	PICSTART® Plus Entry Level Development Programmer	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓				
	PRO MATE® II Universal Device Programmer	✓	✓	✓	✓	✓	✓**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
Demo Boards and Eval Kits	PICDEM™ 1 Demonstration Board			✓		✓		✓†		✓			✓							
	PICDEM™ 2 Demonstration Board				✓†			✓†							✓	✓				
	PICDEM™ 3 Demonstration Board											✓								
	PICDEM™ 14A Demonstration Board		✓																	
	PICDEM™ 17 Demonstration Board													✓						
	KEELOQ® Evaluation Kit																	✓		
	KEELOQ® Transponder Kit																	✓		
	microID™ Programmer's Kit																		✓	
	125 kHz microID™ Developer's Kit																		✓	
	125 kHz Anticollision microID™ Developer's Kit																		✓	
	13.56 MHz Anticollision microID™ Developer's Kit																		✓	
	MCP2510 CAN Developer's Kit																			✓

* Contact the Microchip Technology Inc. web site at www.microchip.com for information on how to use the MPLAB® ICD In-Circuit Debugger (DV164001) with PIC16C62, 63, 64, 65, 72, 73, 74, 76, 77.
** Contact Microchip Technology Inc. for availability date.
† Development tool is available on select devices.

NOTES:

23.0 ELECTRICAL CHARACTERISTICS

Absolute Maximum Ratings (†)

Ambient temperature under bias	-55°C to +125°C
Storage temperature	-65°C to +150°C
Voltage on any pin with respect to V_{SS} (except V_{DD}, $\overline{MCLR}$, and RA4)	-0.3V to (V_{DD} + 0.3V)
Voltage on V_{DD} with respect to V_{SS}	-0.3V to +7.5V
Voltage on $\overline{MCLR}$ with respect to V_{SS} **(Note 2)**	0V to +13.25V
Voltage on RA4 with respect to Vss	0V to +8.5V
Total power dissipation **(Note 1)**	1.0W
Maximum current out of V_{SS} pin	300 mA
Maximum current into V_{DD} pin	250 mA
Input clamp current, I_{IK} ($V_I < 0$ or $V_I > V_{DD}$)	±20 mA
Output clamp current, I_{OK} ($V_O < 0$ or $V_O > V_{DD}$)	±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 **(Note 3)** (combined)	200 mA
Maximum current sourced by PORTA, PORTB, and PORTE **(Note 3)** (combined)	200 mA
Maximum current sunk by PORTC and PORTD **(Note 3)** (combined)	200 mA
Maximum current sourced by PORTC and PORTD **(Note 3)** (combined)	200 mA

Note 1: Power dissipation is calculated as follows:
$Pdis = V_{DD} \times \{I_{DD} - \sum I_{OH}\} + \sum \{(V_{DD}-V_{OH}) \times I_{OH}\} + \sum(V_{OI} \times I_{OL})$

2: Voltage spikes below V_{SS} at the $\overline{MCLR}$/V_{PP} pin, inducing currents greater than 80 mA, may cause latchup. Thus, a series resistor of 50-100Ω should be used when applying a "low" level to the $\overline{MCLR}$/V_{PP} pin, rather than pulling this pin directly to V_{SS}.

3: PORTD and PORTE not available on the PIC18F2X39 devices.

† 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.

FIGURE 23-1: PIC18FXX39 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL)

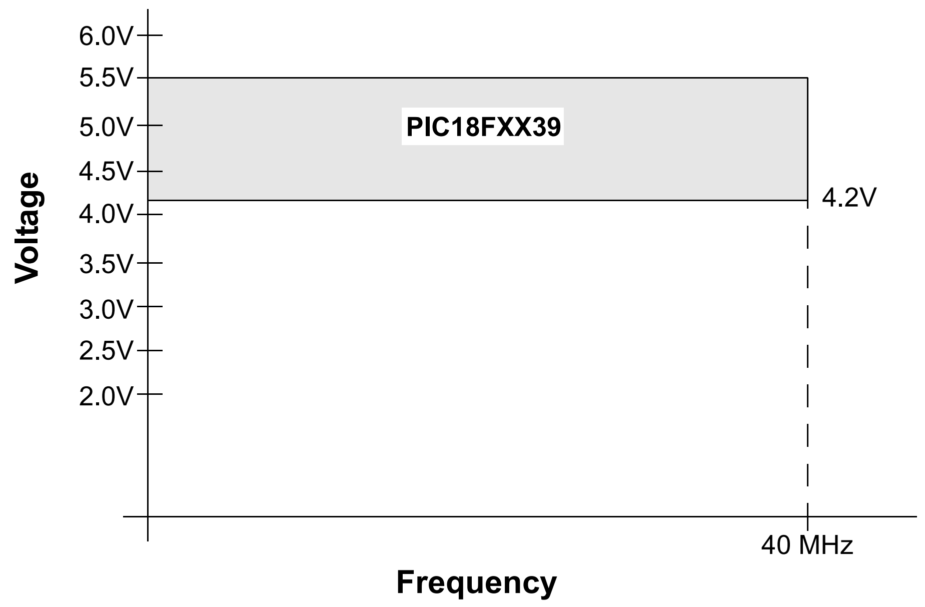
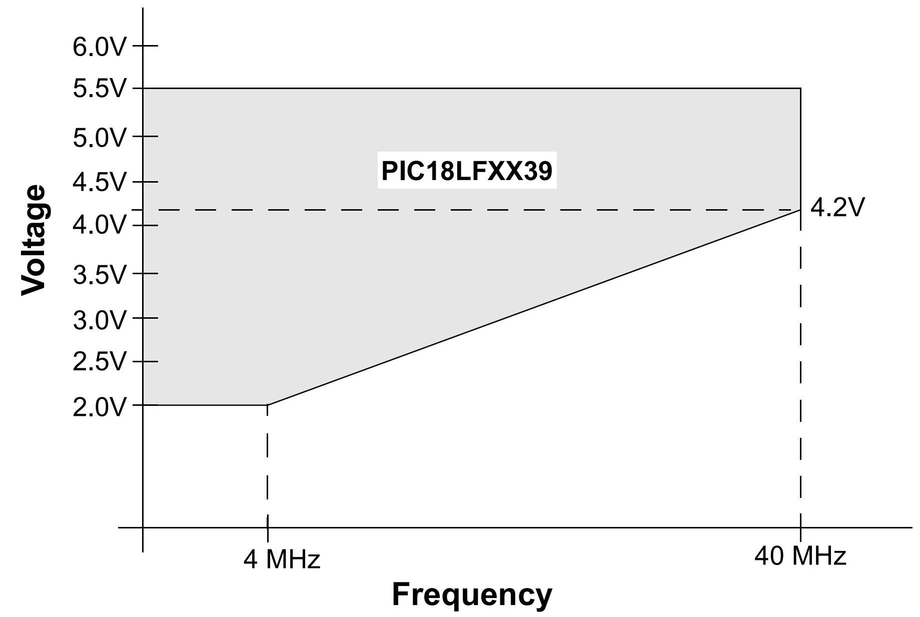


FIGURE 23-2: PIC18LFXX39 VOLTAGE-FREQUENCY GRAPH (INDUSTRIAL)



FMAX = (16.36 MHz/V) (VDDAPPMIN – 2.0V) + 4 MHz

Note: VDDAPPMIN is the minimum voltage of the PICmicro® device in the application.

23.1 DC Characteristics: PIC18FXX39 (Industrial, Extended) PIC18LFXX39 (Industrial)

PIC18LFXX39 (Industrial)	**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ TA ≤ +85°C for industrial
PIC18FXX39 (Industrial, Extended)	**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		PIC18LFXX39	2.0	—	5.5	V	HS Osc mode
D001		PIC18FXX39	4.2	—	5.5	V	
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 3.1 (Power-on Reset) for details
D004	SVDD	**VDD Rise Rate** to ensure internal Power-on Reset signal	0.05	—	—	V/ms	See Section 3.1 (Power-on Reset) for details
	VBOR	**Brown-out Reset Voltage**					
D005		PIC18LFXX39					
		BORV1:BORV0 = `11`	1.98	—	2.14	V	85°C ≥ T ≥ 25°C
		BORV1:BORV0 = `10`	2.67	—	2.89	V	
		BORV1:BORV0 = `01`	4.16	—	4.5	V	
		BORV1:BORV0 = `00`	4.45	—	4.83	V	
D005		PIC18FXX39					
		BORV1:BORV0 = `1x`	N.A.	—	N.A.	V	Not in operating voltage range of device
		BORV1:BORV0 = `01`	4.16	—	4.5	V	
		BORV1:BORV0 = `00`	4.45	—	4.83	V	

Legend: Shading of rows is to assist in readability of the table.

Note 1: This is the limit to which VDD can be lowered in SLEEP mode, or during a device RESET, without losing RAM data.

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:
$\overline{\text{OSC1}}$ = external square wave, from rail-to-rail; all I/O pins tri-stated, pulled to VDD
$\overline{\text{MCLR}}$ = 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 in hi-impedance state and tied to VDD or VSS, and all features that add delta current disabled (such as WDT, Timer1 Oscillator, BOR, etc.).

4: The LVD and BOR modules share a large portion of circuitry. The ΔIBOR and ΔILVD currents are not additive. Once one of these modules is enabled, the other may also be enabled without further penalty.

23.1 DC Characteristics: PIC18FXX39 (Industrial, Extended) PIC18LFXX39 (Industrial) (Continued)

PIC18LFXX39 (Industrial)			**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ T_A ≤ +85°C for industrial				
PIC18FXX39 (Industrial, Extended)			**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ T_A ≤ +85°C for industrial; -40°C ≤ T_A ≤ +125°C for extended				
Param No.	**Symbol**	**Characteristic**	**Min**	**Typ**	**Max**	**Units**	**Conditions**
	I_{DD}	**Supply Current**(2)					
D010C		PIC18LFXX39	—	10	25	mA	EC, ECIO osc configurations V_{DD} = 4.2V, -40°C to +85°C
D010C		PIC18FXX39	—	10	25	mA	EC, ECIO osc configurations V_{DD} = 4.2V, -40°C to +125°C
D013		PIC18LFXX39	— —	10 15	15 25	mA mA	HS osc configuration F_{OSC} = 25 MHz, V_{DD} = 5.5V HS + PLL osc configurations F_{OSC} = 10 MHz, V_{DD} = 5.5V
D013		PIC18FXX39	— —	10 15	15 25	mA mA	HS osc configuration F_{OSC} = 25 MHz, V_{DD} = 5.5V HS + PLL osc configurations F_{OSC} = 10 MHz, V_{DD} = 5.5V
	I_{PD}	**Power-down Current**(3)					
D020		PIC18LFXX39	— — —	0.08 0.1 3	0.9 4 10	μA μA μA	V_{DD} = 2.0V, +25°C V_{DD} = 2.0V, -40°C to +85°C V_{DD} = 4.2V, -40°C to +85°C
D020 D021B		PIC18FXX39	— — —	.1 3 15	.9 10 25	μA μA μA	V_{DD} = 4.2V, +25°C V_{DD} = 4.2V, -40°C to +85°C V_{DD} = 4.2V, -40°C to +125°C

Legend: Shading of rows is to assist in readability of the table.

Note 1: This is the limit to which V_{DD} can be lowered in SLEEP mode, or during a device RESET, without losing RAM data.

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 I_{DD} measurements in active Operation mode are:
$\overline{\text{OSC1}}$ = external square wave, from rail-to-rail; all I/O pins tri-stated, pulled to V_{DD}
$\overline{\text{MCLR}}$ = V_{DD}; 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 in hi-impedance state and tied to V_{DD} or V_{SS}, and all features that add delta current disabled (such as WDT, Timer1 Oscillator, BOR, etc.).

4: The LVD and BOR modules share a large portion of circuitry. The ΔI_{BOR} and ΔI_{LVD} currents are not additive. Once one of these modules is enabled, the other may also be enabled without further penalty.

23.1 DC Characteristics: PIC18FXX39 (Industrial, Extended) PIC18LFXX39 (Industrial) (Continued)

PIC18LFXX39 (Industrial)			**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ TA ≤ +85°C for industrial				
PIC18FXX39 (Industrial, Extended)			**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** PIC18LFXX39	—	0.75	1.5	μA	VDD = 2.0V, +25°C
			—	2	8	μA	VDD = 2.0V, -40°C to +85°C
			—	10	25	μA	VDD = 4.2V, -40°C to +85°C
D022		**Watchdog Timer** PIC18FXX39	—	7	15	μA	VDD = 4.2V, +25°C
			—	10	25	μA	VDD = 4.2V, -40°C to +85°C
			—	25	40	μA	VDD = 4.2V, -40°C to +125°C
D022A	ΔIBOR	**Brown-out Reset**(4) PIC18LFXX39	—	29	35	μA	VDD = 2.0V, +25°C
			—	29	45	μA	VDD = 2.0V, -40°C to +85°C
			—	33	50	μA	VDD = 4.2V, -40°C to +85°C
D022A		**Brown-out Reset**(4) PIC18FXX39	—	36	40	μA	VDD = 4.2V, +25°C
			—	36	50	μA	VDD = 4.2V, -40°C to +85°C
			—	36	65	μA	VDD = 4.2V, -40°C to +125°C
D022B	ΔILVD	**Low Voltage Detect**(4) PIC18LFXX39	—	29	35	μA	VDD = 2.0V, +25°C
			—	29	45	μA	VDD = 2.0V, -40°C to +85°C
			—	33	50	μA	VDD = 4.2V, -40°C to +85°C
D022B		**Low Voltage Detect**(4) PIC18FXX39	—	33	40	μA	VDD = 4.2V, +25°C
			—	33	50	μA	VDD = 4.2V, -40°C to +85°C
			—	33	65	μA	VDD = 4.2V, -40°C to +125°C

Legend: Shading of rows is to assist in readability of the table.

Note 1: This is the limit to which VDD can be lowered in SLEEP mode, or during a device RESET, without losing RAM data.

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:
$\overline{\text{OSC1}}$ = external square wave, from rail-to-rail; all I/O pins tri-stated, pulled to VDD
$\overline{\text{MCLR}}$ = 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 in hi-impedance state and tied to VDD or VSS, and all features that add delta current disabled (such as WDT, Timer1 Oscillator, BOR, etc.).

4: The LVD and BOR modules share a large portion of circuitry. The ΔIBOR and ΔILVD currents are not additive. Once one of these modules is enabled, the other may also be enabled without further penalty.

23.2 DC Characteristics: PIC18FXX39 (Industrial, Extended) PIC18LFXX39 (Industrial)

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**	**Max**	**Units**	**Conditions**
	V_{IL}	**Input Low Voltage**				
		I/O ports:				
D030		with TTL buffer	Vss	0.15 V_{DD}	V	V_{DD} < 4.5V
D030A			—	0.8	V	4.5V ≤ V_{DD} ≤ 5.5V
D031		with Schmitt Trigger buffer RC3 and RC4	Vss Vss	0.2 V_{DD} 0.3 V_{DD}	V V	
D032		$\overline{\text{MCLR}}$	V_{SS}	0.2 V_{DD}	V	
D032A		OSC1 (HS mode)	V_{SS}	0.3 V_{DD}	V	
D033		OSC1 (EC mode)	V_{SS}	0.2 V_{DD}	V	
	V_{IH}	**Input High Voltage**				
		I/O ports:				
D040		with TTL buffer	0.25 V_{DD} + 0.8V	V_{DD}	V	V_{DD} < 4.5V
D040A			2.0	V_{DD}	V	4.5V ≤ V_{DD} ≤ 5.5V
D041		with Schmitt Trigger buffer RC3 and RC4	0.8 V_{DD} 0.7 V_{DD}	V_{DD} V_{DD}	V V	
D042		$\overline{\text{MCLR}}$, OSC1 (EC mode)	0.8 V_{DD}	V_{DD}	V	
D042A		OSC1 (HS mode)	0.7 V_{DD}	V_{DD}	V	
	I_{IL}	**Input Leakage Current**(1,2)				
D060		I/O ports	.02	±1	μA	V_{SS} ≤ V_{PIN} ≤ V_{DD}, Pin at hi-impedance
D061		$\overline{\text{MCLR}}$	—	±1	μA	Vss ≤ V_{PIN} ≤ V_{DD}
D063		OSC1	—	±1	μA	Vss ≤ V_{PIN} ≤ V_{DD}
	I_{PU}	**Weak Pull-up Current**				
D070	I_{PURB}	PORTB weak pull-up current	50	450	μA	V_{DD} = 5V, V_{PIN} = V_{SS}

Note 1: The leakage current on the $\overline{\text{MCLR}}$ 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.

2: Negative current is defined as current sourced by the pin.

3: Parameter is characterized but not tested.

23.2 DC Characteristics: PIC18FXX39 (Industrial, Extended) PIC18LFXX39 (Industrial) (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**	**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
	V_{OH}	**Output High Voltage**(2)				
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
D150	V_{OD}	**Open Drain High Voltage**	—	8.5	V	RA4 pin
		Capacitive Loading Specs on Output Pins				
D100(3)	C_{OSC2}	OSC2 pin	—	15	pF	In HS mode when external clock is used to drive OSC1
D101	C_{IO}	All I/O pins	—	50	pF	To meet the AC Timing Specifications
D102	C_B	SCL, SDA	—	400	pF	In I^{2}C mode

Note 1: The leakage current on the $\overline{\text{MCLR}}$ 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.

2: Negative current is defined as current sourced by the pin.

3: Parameter is characterized but not tested.

FIGURE 23-3: LOW VOLTAGE DETECT CHARACTERISTICS

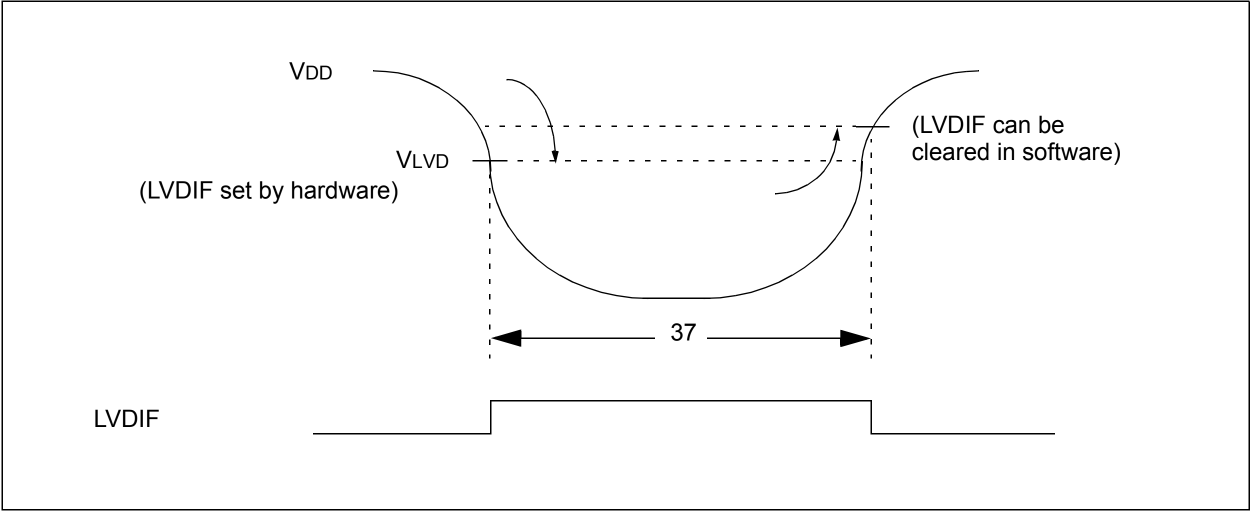


TABLE 23-1: LOW VOLTAGE DETECT CHARACTERISTICS

				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**
D420	VLVD	LVD Voltage on VDD transition high to low	LVV = `0001`	1.98	2.06	2.14	V	T ≥ 25°C
			LVV = `0010`	2.18	2.27	2.36	V	T ≥ 25°C
			LVV = `0011`	2.37	2.47	2.57	V	T ≥ 25°C
			LVV = `0100`	2.48	2.58	2.68	V	
			LVV = `0101`	2.67	2.78	2.89	V	
			LVV = `0110`	2.77	2.89	3.01	V	
			LVV = `0111`	2.98	3.1	3.22	V	
			LVV = `1000`	3.27	3.41	3.55	V	
			LVV = `1001`	3.47	3.61	3.75	V	
			LVV = `1010`	3.57	3.72	3.87	V	
			LVV = `1011`	3.76	3.92	4.08	V	
			LVV = `1100`	3.96	4.13	4.3	V	
			LVV = `1101`	4.16	4.33	4.5	V	
			LVV = `1110`	4.45	4.64	4.83	V	

TABLE 23-2: MEMORY PROGRAMMING REQUIREMENTS

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.	**Sym**	**Characteristic**	**Min**	**Typ†**	**Max**	**Units**	**Conditions**
		Internal Program Memory Programming Specifications					
D110	VPP	Voltage on $\overline{\text{MCLR}}$/VPP pin	9.00	—	13.25	V	
D113	IDDP	Supply Current during Programming	—	—	10	mA	
		Data EEPROM Memory					
D120	ED	Cell Endurance	100K	1M	—	E/W	-40°C to +85°C
D121	VDRW	VDD for Read/Write	VMIN	—	5.5	V	Using EECON to read/write VMIN = Minimum operating voltage
D122	TDEW	Erase/Write Cycle Time	—	4	—	ms	
D123	TRETD	Characteristic Retention	40	—	—	Year	Provided no other specifications are violated
D123A	TRETD	Characteristic Retention	100	—	—	Year	25°C **(Note 1)**
D124	TREF	Number of Total Erase/Write Cycles before Refresh**(2)**	1M	10M	—	E/W	-40°C to +85°C
		Program FLASH Memory					
D130	EP	Cell Endurance	10K	100K	—	E/W	-40°C to +85°C
D131	VPR	VDD for Read	VMIN	—	5.5	V	VMIN = Minimum operating voltage
D132	VIE	VDD for Block Erase	4.5	—	5.5	V	Using ICSP port
D132A	VIW	VDD for Externally Timed Erase or Write	4.5	—	5.5	V	Using ICSP port
D132B	VPEW	VDD for Self-timed Write	VMIN	—	5.5	V	VMIN = Minimum operating voltage
D133	TIE	ICSP Block Erase Cycle Time	—	4	—	ms	VDD ≥ 4.5V
D133A	TIW	ICSP Erase or Write Cycle Time (externally timed)	1	—	—	ms	VDD ≥ 4.5V
D133A	TIW	Self-timed Write Cycle Time	—	2	—	ms	
D134	TRETD	Characteristic Retention	40	—	—	Year	Provided no other specifications are violated
D134A	TRETD	Characteristic Retention	100	—	—	Year	25°C **(Note 1)**

† Data in “Typ” column is at 5.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: Retention time is valid, provided no other specifications are violated.

2: Refer to Section 6.8 for a more detailed discussion on data EEPROM endurance.

23.3 AC (Timing) Characteristics

23.3.1 TIMING PARAMETER SYMBOLOGY

The timing parameter symbols have been created following one of the following formats:

1. TppS2ppS		3. TCC:ST	(I^2C specifications only)
2. TppS		4. Ts	(I^2C specifications only)
T			
F	Frequency	T	Time

Lowercase letters (pp) and their meanings:

pp			
cc	CCP1	osc	OSC1
ck	CLKO	rd	$\overline{\text{RD}}$
cs	$\overline{\text{CS}}$	rw	$\overline{\text{RD}}$ or $\overline{\text{WR}}$
di	SDI	sc	SCK
do	SDO	ss	$\overline{\text{SS}}$
dt	Data in	t0	T0CKI
io	I/O port	t1	T13CKI
mc	$\overline{\text{MCLR}}$	wr	$\overline{\text{WR}}$

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		

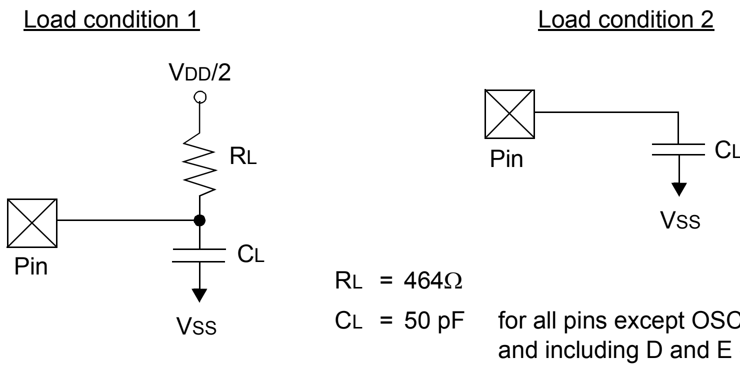
23.3.2 TIMING CONDITIONS

The temperature and voltages specified in Table 23-3 apply to all timing specifications unless otherwise noted. Figure 23-4 specifies the load conditions for the timing specifications.

TABLE 23-3: TEMPERATURE AND VOLTAGE SPECIFICATIONS - AC

AC CHARACTERISTICS	**Standard Operating Conditions (unless otherwise stated)** Operating temperature -40°C ≤ T_A ≤ +85°C for industrial -40°C ≤ T_A ≤ +125°C for extended Operating voltage V_{DD} range as described in DC spec Section 23.1 and Section 23.2. LC parts operate for industrial temperatures only.

FIGURE 23-4: LOAD CONDITIONS FOR DEVICE TIMING SPECIFICATIONS



R_L = 464Ω

C_L = 50 pF for all pins except OSC2/CLKO and including D and E outputs as ports

23.3.3 TIMING DIAGRAMS AND SPECIFICATIONS

FIGURE 23-5: EXTERNAL CLOCK TIMING (ALL MODES EXCEPT PLL)

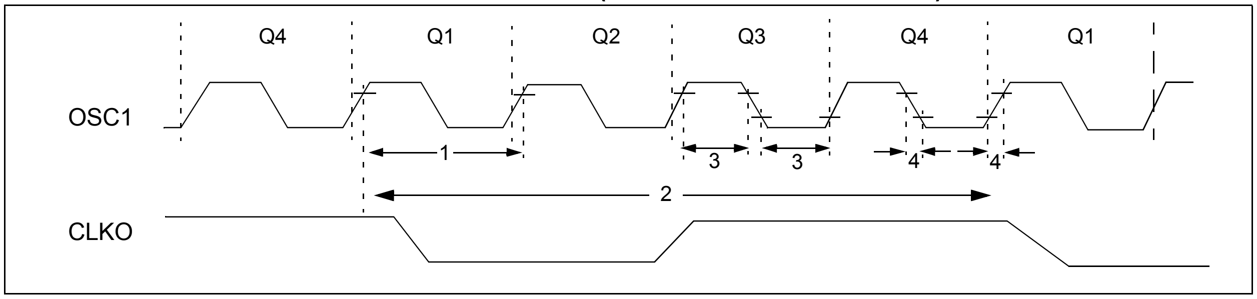


TABLE 23-4: EXTERNAL CLOCK TIMING REQUIREMENTS

Param. No.	Symbol	Characteristic	Min	Max	Units	Conditions
1A	FOSC	External CLKI Frequency(1) Oscillator Frequency(1)	DC	40	MHz	EC, ECIO, -40°C to +85°C
			DC	25	MHz	EC, ECIO, +85°C to +125°C
			4	25	MHz	HS osc
			4	10	MHz	HS + PLL osc, -40°C to +85°C
			4	6.25	MHz	HS + PLL osc, +85°C to +125°C
1	TOSC	External CLKI Period(1) Oscillator Period(1)	25	—	ns	EC, ECIO, -40°C to +85°C
			40	—	ns	EC, ECIO, +85°C to +125°C
			40	250	ns	HS osc
			100	250	ns	HS + PLL osc, -40°C to +85°C
			160	250	ns	HS + PLL osc, +85°C to +125°C
2	TCY	Instruction Cycle Time(1)	100	—	ns	TCY = 4/FOSC, -40°C to +85°C
			160	—	ns	TCY = 4/FOSC, +85°C to +125°C
3	TosL, TosH	External Clock in (OSC1) High or Low Time	10	—	ns	HS osc
4	TosR, TosF	External Clock in (OSC1) Rise or Fall Time	—	7.5	ns	HS osc

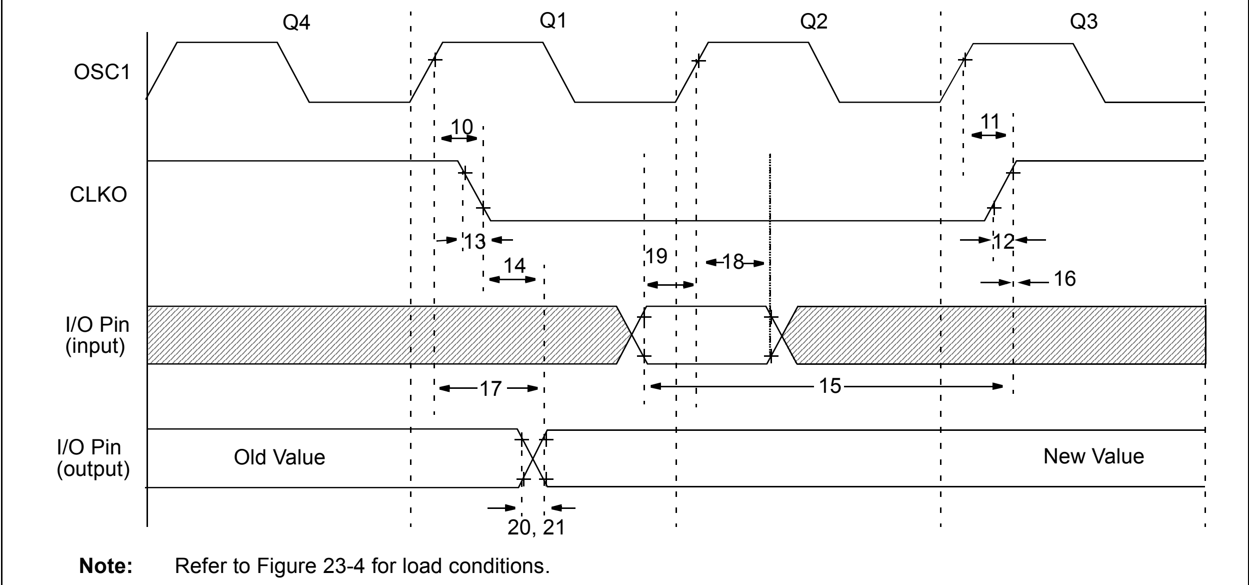
Note 1: Instruction cycle period (TCY) equals four times the input oscillator time-base period for all configurations except PLL. All specified values are based on characterization data for that particular oscillator type under standard operating conditions with the device executing code. Exceeding these 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/CLKI pin. When an external clock input is used, the "max." cycle time limit is "DC" (no clock) for all devices.

TABLE 23-5: PLL CLOCK TIMING SPECIFICATIONS (VDD = 4.2 TO 5.5V)

Param No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
—	FOSC	Oscillator Frequency Range	4	—	10	MHz	HS mode only
—	FSYS	On-Chip VCO System Frequency	16	—	40	MHz	HS mode only
—	t_{rc}	PLL Start-up Time (Lock Time)	—	—	2	ms	
—	ΔCLK	CLKO Stability (Jitter)	-2	—	+2	%	

† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

FIGURE 23-6: CLKO AND I/O TIMING



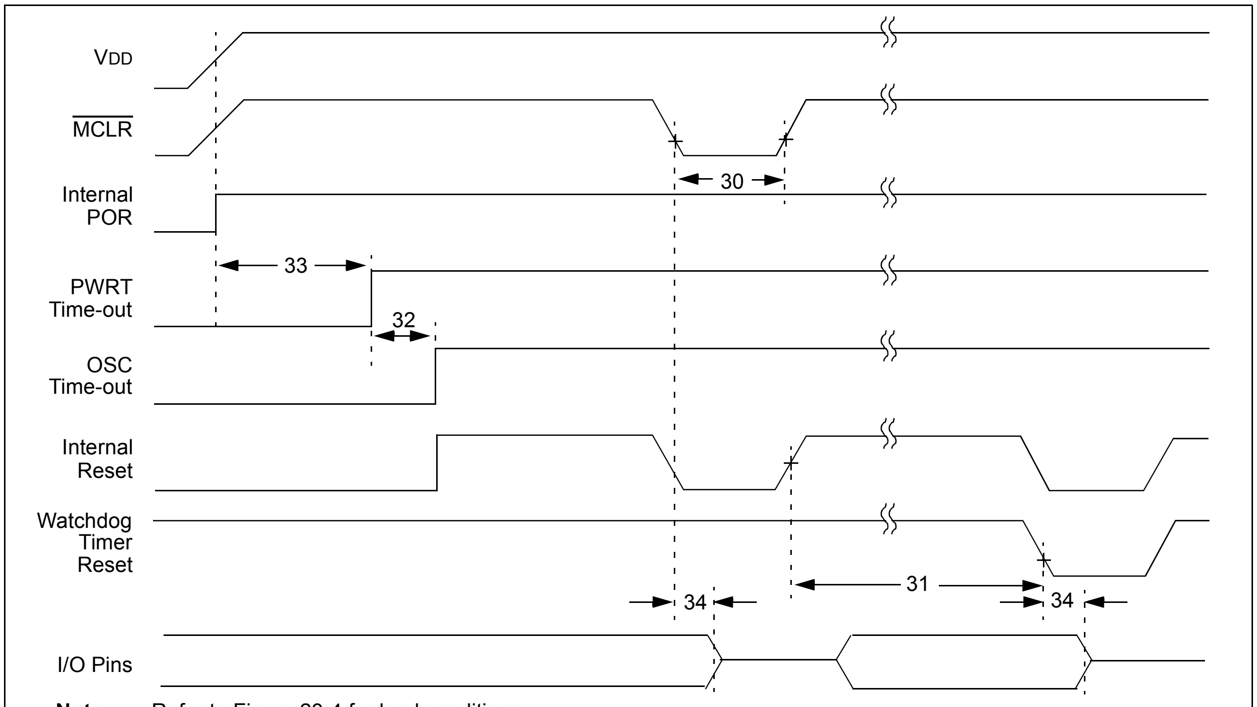
Note: Refer to Figure 23-4 for load conditions.

TABLE 23-6: CLKO AND I/O TIMING REQUIREMENTS

Param. No.	Symbol	Characteristic		Min	Typ	Max	Units	Conditions
10	TosH2ckL	OSC1↑ to CLKO↓		—	75	200	ns	**(Note 1)**
11	TosH2ckH	OSC1↑ to CLKO↑		—	75	200	ns	**(Note 1)**
12	TckR	CLKO rise time		—	35	100	ns	**(Note 1)**
13	TckF	CLKO fall time		—	35	100	ns	**(Note 1)**
14	TckL2ioV	CLKO↓ to Port out valid		—	—	0.5 TCY + 20	ns	**(Note 1)**
15	TioV2ckH	Port in valid before CLKO ↑		0.25 TCY + 25	—	—	ns	**(Note 1)**
16	TckH2ioI	Port in hold after CLKO ↑		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**F**XXXX	100	—	—	ns	
18A			PIC18**LF**XXXX	200	—	—	ns	
19	TioV2osH	Port input valid to OSC1↑ (I/O in setup time)		0	—	—	ns	
20	TioR	Port output rise time	PIC18**F**XXXX	—	10	25	ns	
20A			PIC18**LF**XXXX	—	—	60	ns	VDD = 2V
21	TioF	Port output fall time	PIC18**F**XXXX	—	10	25	ns	
21A			PIC18**LF**XXXX	—	—	60	ns	VDD = 2V
22††	TINP	INT pin high or low time		TCY	—	—	ns	
23††	TRBP	RB7:RB4 change INT high or low time		TCY	—	—	ns	
24††	TRCP	RC7:RC4 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 CLKO output is 4 x TOSC.

FIGURE 23-7: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER AND POWER-UP TIMER TIMING



Note: Refer to Figure 23-4 for load conditions.

FIGURE 23-8: BROWN-OUT RESET TIMING

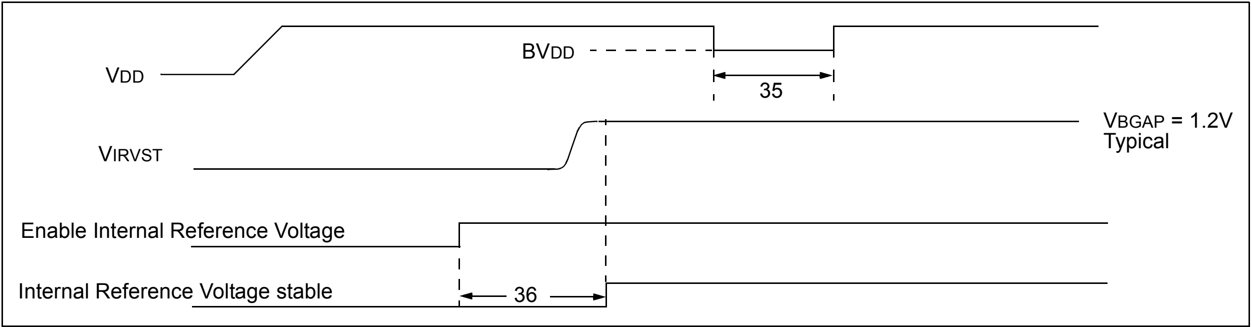
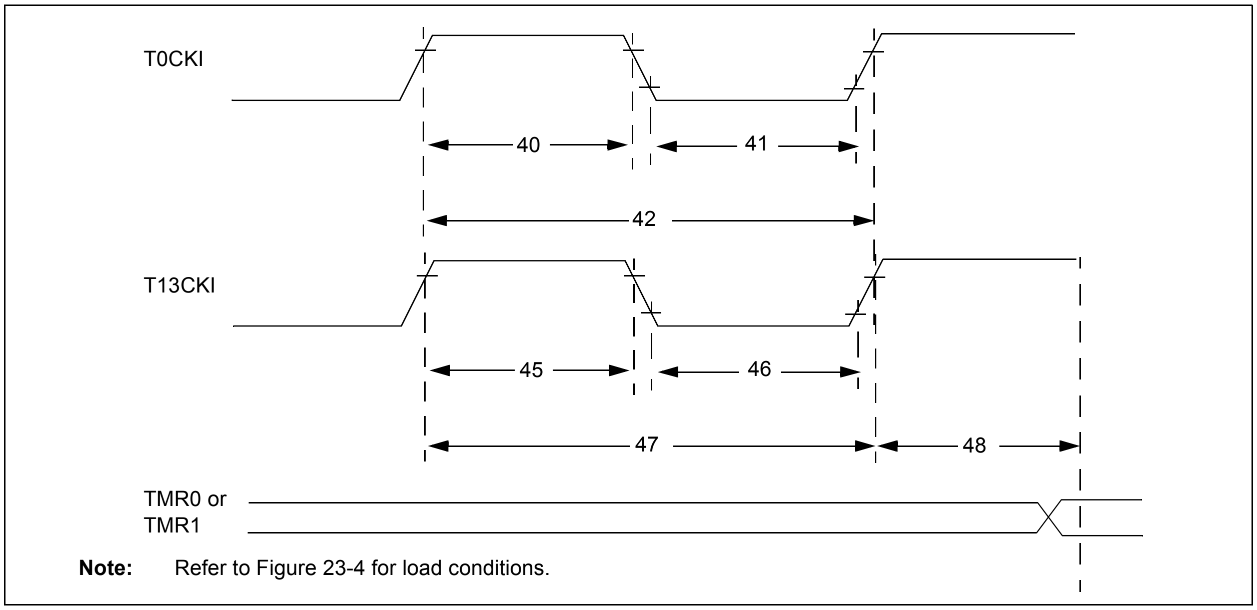

VDD
BVDD
35
VIRVST
VBGAP = 1.2V Typical
Enable Internal Reference Voltage
Internal Reference Voltage stable
36

TABLE 23-7: RESET, WATCHDOG TIMER, OSCILLATOR START-UP TIMER, POWER-UP TIMER AND BROWN-OUT RESET 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 Postscaler)	7	18	33	ms	
32	TOST	Oscillation Start-up Timer Period	1024 TOSC	—	1024 TOSC	—	TOSC = OSC1 period
33	TPWRT	Power up Timer Period	28	72	132	ms	
34	TIOZ	I/O high impedance from $\overline{\text{MCLR}}$ Low or Watchdog Timer Reset	—	2	—	μs	
35	TBOR	Brown-out Reset Pulse Width	200	—	—	μs	VDD ≤ BVDD (see D005)
36	TIVRST	Time for Internal Reference Voltage to become stable	—	20	500	μs	
37	TLVD	Low Voltage Detect Pulse Width	200	—	—	μs	VDD ≤ VLVD (see D420)

FIGURE 23-9: TIMER0 AND TIMER1 EXTERNAL CLOCK TIMINGS

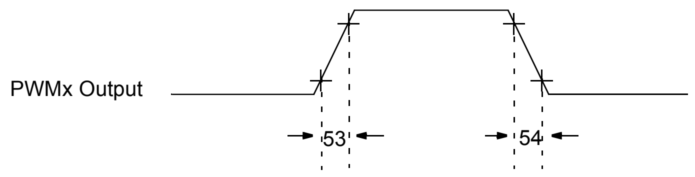


Note: Refer to Figure 23-4 for load conditions.

TABLE 23-8: 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 $\frac{T_{CY} + 40}{N}$	—	ns	N = prescale value (1, 2, 4,..., 256)
45	Tt1H	T13CKI High Time	Synchronous, no prescaler		0.5TCY + 20	—	ns	
			Synchronous, with prescaler	PIC18**F**XXXX	10	—	ns	
				PIC18**LF**XXXX	25	—	ns	
			Asynchronous	PIC18**F**XXXX	30	—	ns	
				PIC18**LF**XXXX	50	—	ns	
46	Tt1L	T13CKI Low Time	Synchronous, no prescaler		0.5TCY + 5	—	ns	
			Synchronous, with prescaler	PIC18**F**XXXX	10	—	ns	
				PIC18**LF**XXXX	25	—	ns	
			Asynchronous	PIC18**F**XXXX	30	—	ns	
				PIC18**LF**XXXX	50	—	ns	
47	Tt1P	T13CKI input period	Synchronous		Greater of: 20 ns or $\frac{T_{CY} + 40}{N}$	—	ns	N = prescale value (1, 2, 4, 8)
			Asynchronous		60	—	ns	
	Ft1	T13CKI oscillator input frequency range			DC	50	kHz	
48	Tcke2tmrI	Delay from external T13CKI clock edge to timer increment			2 TOSC	7 TOSC	—	

FIGURE 23-10: PWM TIMINGS (PWM1 AND PWM2)

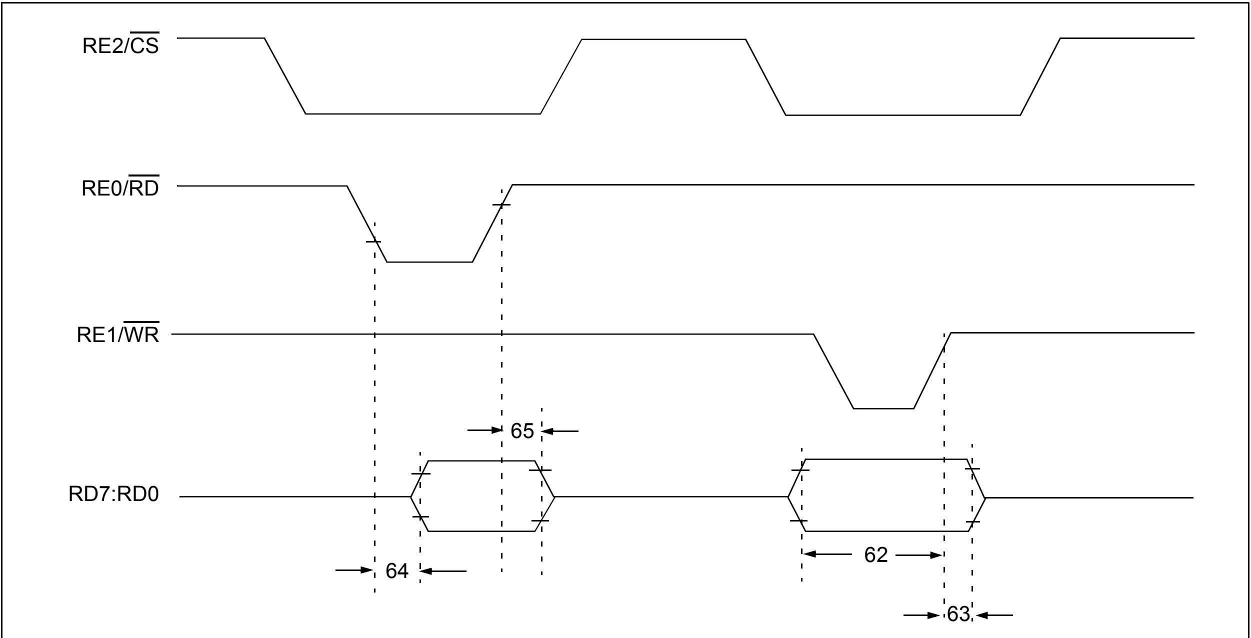


Note: Refer to Figure 23-4 for load conditions.

TABLE 23-9: PWM TIMING REQUIREMENTS (PWM1 AND PWM2)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
53	TccR	PWMx Output Rise Time	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
54	TccF	PWMx Output Fall Time	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V

FIGURE 23-11: PARALLEL SLAVE PORT TIMING (PIC18F4X39)

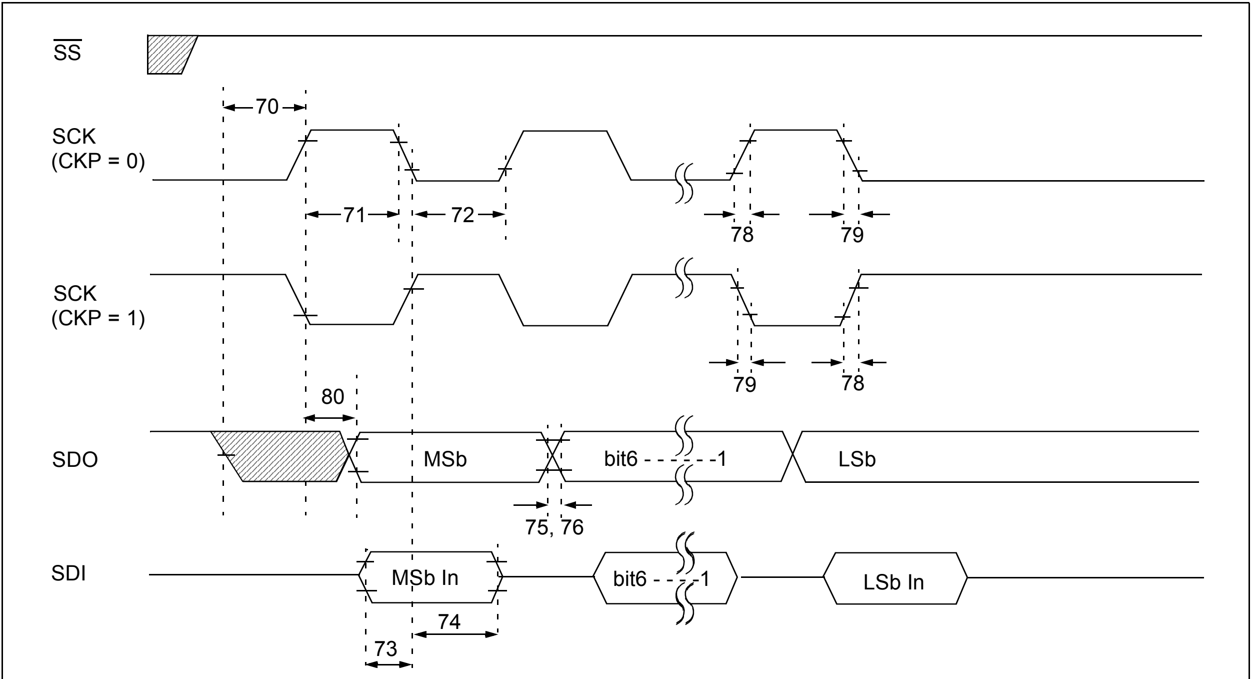


Note: Refer to Figure 23-4 for load conditions.

TABLE 23-10: PARALLEL SLAVE PORT REQUIREMENTS (PIC18F4X39)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
62	TdtV2wrH	Data in valid before $\overline{WR}$↑ or $\overline{CS}$↑ (setup time)		20 25	— —	ns ns	 Extended Temp. Range
63	TwrH2dtI	$\overline{WR}$↑ or $\overline{CS}$↑ to data–in invalid (hold time)	PIC18**F**XXXX	20	—	ns	
			PIC18**LF**XXXX	35	—	ns	VDD = 2V
64	TrdL2dtV	$\overline{RD}$↓ and $\overline{CS}$↓ to data–out valid		— —	80 90	ns ns	 Extended Temp. Range
65	TrdH2dtI	$\overline{RD}$↑ or $\overline{CS}$↓ to data–out invalid		10	30	ns	
66	TibfINH	Inhibit of the IBF flag bit being cleared from $\overline{WR}$↑ or $\overline{CS}$↑		—	3 TCY		

FIGURE 23-12: EXAMPLE SPI MASTER MODE TIMING (CKE = 0)



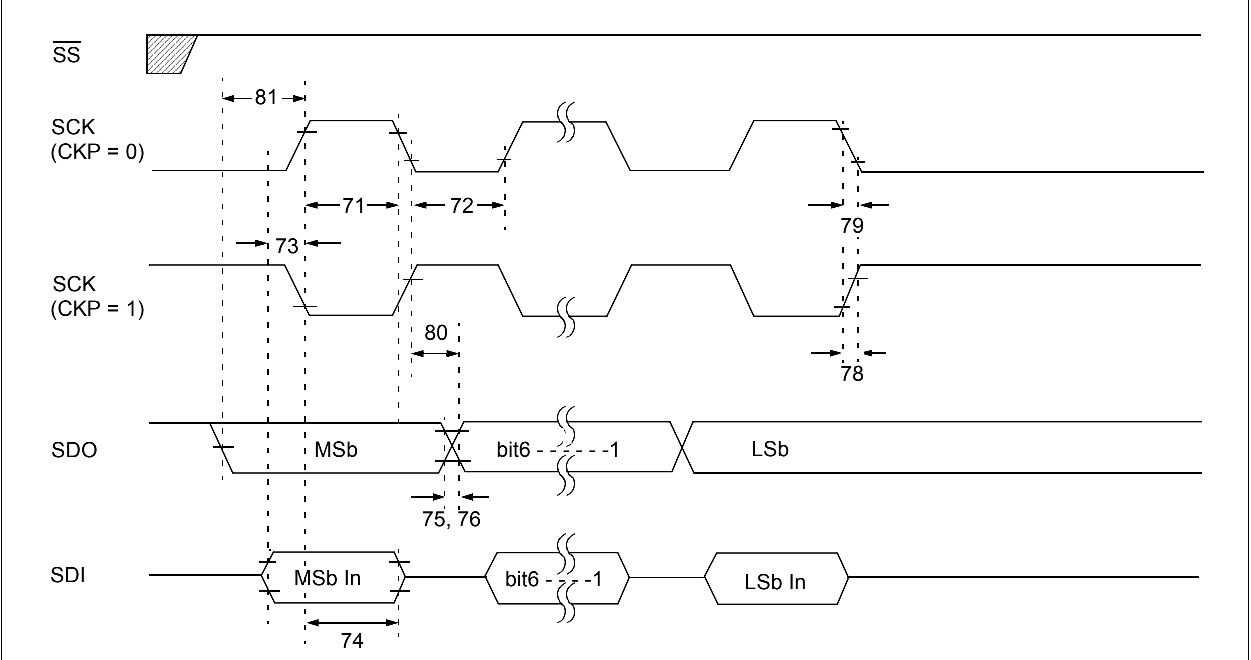
Note: Refer to Figure 23-4 for load conditions.

TABLE 23-11: EXAMPLE SPI MODE REQUIREMENTS (MASTER MODE, CKE = 0)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
70	TssL2scH, TssL2scL	$\overline{SS}$↓ to SCK↓ or SCK↑ input		TCY	—	ns	
71	TscH	SCK input high time (Slave mode)	Continuous	1.25 TCY + 30	—	ns	
71A			Single Byte	40	—	ns	**(Note 1)**
72	TscL	SCK input low time (Slave mode)	Continuous	1.25 TCY + 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 Byte 1 to the 1st clock edge of Byte 2		1.5 TCY + 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**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
76	TdoF	SDO data output fall time	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
78	TscR	SCK output rise time (Master mode)	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
79	TscF	SCK output fall time (Master mode)	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
80	TscH2doV, TscL2doV	SDO data output valid after SCK edge	PIC18**F**XXXX	—	50	ns	
			PIC18**LF**XXXX	—	150	ns	VDD = 2V

Note 1: Requires the use of Parameter # 73A.
2: Only if Parameter # 71A and # 72A are used.

FIGURE 23-13: EXAMPLE SPI MASTER MODE TIMING (CKE = 1)



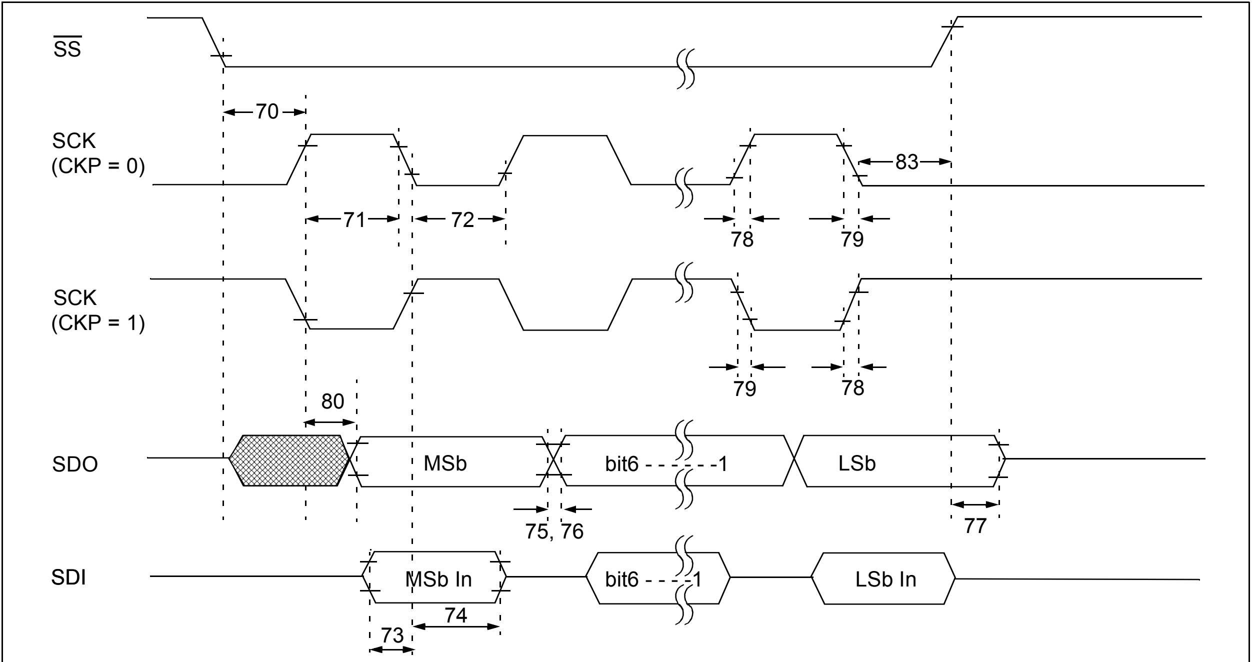
Note: Refer to Figure 23-4 for load conditions.

TABLE 23-12: 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	1.25 TCY + 30	—	ns	
71A			Single Byte	40	—	ns	**(Note 1)**
72	TscL	SCK input low time (Slave mode)	Continuous	1.25 TCY + 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 Byte 1 to the 1st clock edge of Byte 2		1.5 TCY + 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**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
76	TdoF	SDO data output fall time	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
78	TscR	SCK output rise time (Master mode)	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
79	TscF	SCK output fall time (Master mode)	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
80	TscH2doV, TscL2doV	SDO data output valid after SCK edge	PIC18**F**XXXX	—	50	ns	
			PIC18**LF**XXXX	—	150	ns	VDD = 2V
81	TdoV2scH, TdoV2scL	SDO data output setup to SCK edge		TCY	—	ns	

Note 1: Requires the use of Parameter # 73A.
2: Only if Parameter # 71A and # 72A are used.

FIGURE 23-14: EXAMPLE SPI SLAVE MODE TIMING (CKE = 0)



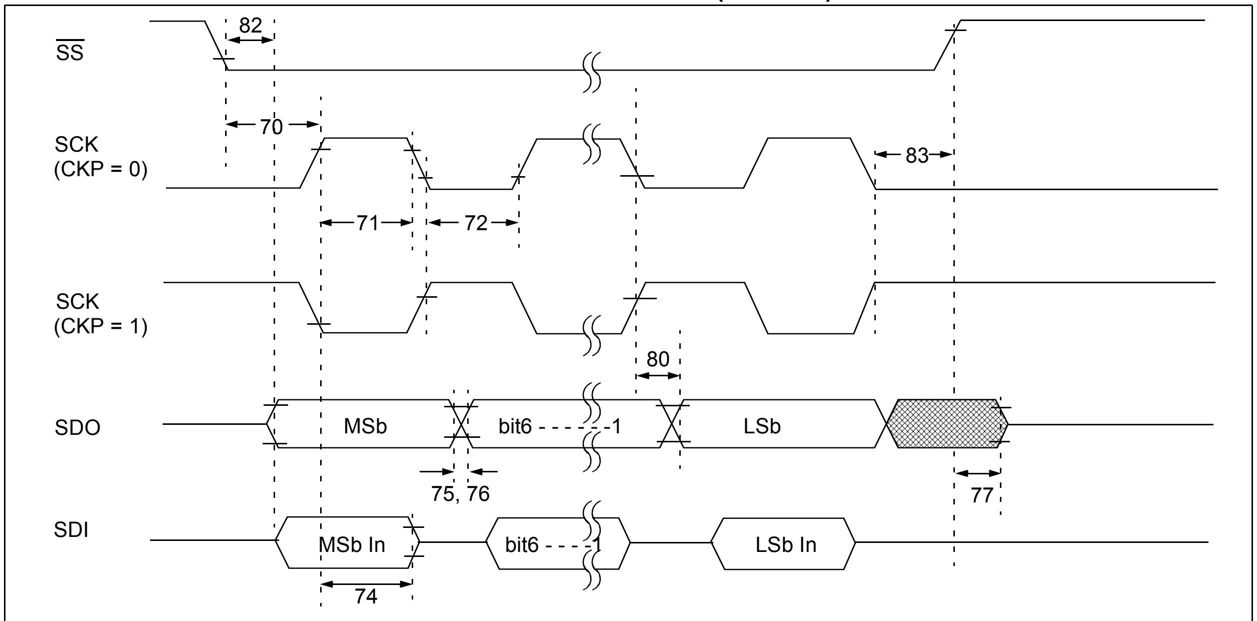
Note: Refer to Figure 23-4 for load conditions.

TABLE 23-13: EXAMPLE SPI MODE REQUIREMENTS (SLAVE MODE TIMING (CKE = 0))

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
70	TssL2scH, TssL2scL	$\overline{SS}$↓ to SCK↓ or SCK↑ input		TCY	—	ns	
71	TscH	SCK input high time (Slave mode)	Continuous	1.25 TCY + 30	—	ns	
71A			Single Byte	40	—	ns	**(Note 1)**
72	TscL	SCK input low time (Slave mode)	Continuous	1.25 TCY + 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 Byte 1 to the first clock edge of Byte 2		1.5 TCY + 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**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
76	TdoF	SDO data output fall time	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
77	TssH2doZ	$\overline{SS}$↑ to SDO output hi-impedance		10	50	ns	
78	TscR	SCK output rise time (Master mode)	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
79	TscF	SCK output fall time (Master mode)	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
80	TscH2doV, TscL2doV	SDO data output valid after SCK edge	PIC18**F**XXXX	—	50	ns	
			PIC18**LF**XXXX	—	150	ns	VDD = 2V
83	TscH2ssH, TscL2ssH	$\overline{SS}$ ↑ after SCK edge		1.5 TCY + 40	—	ns	

Note 1: Requires the use of Parameter # 73A.
2: Only if Parameter # 71A and # 72A are used.

FIGURE 23-15: EXAMPLE SPI SLAVE MODE TIMING (CKE = 1)



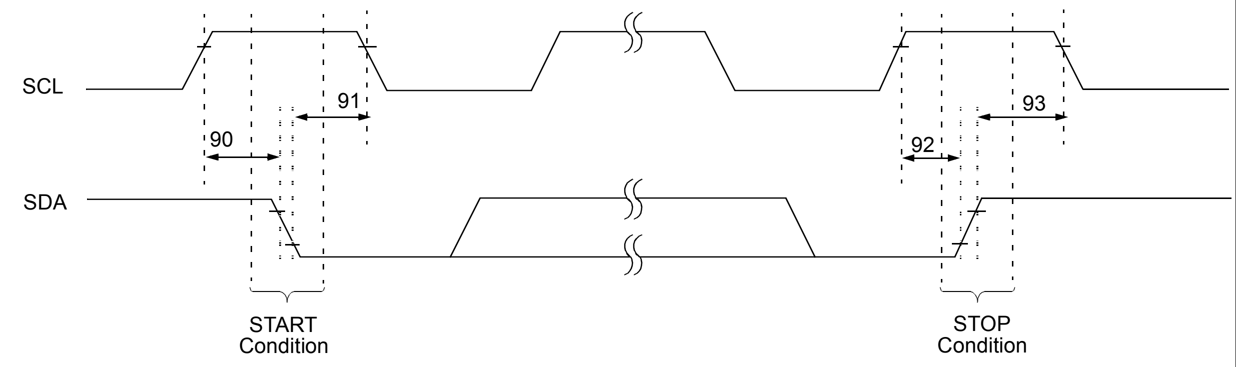
Note: Refer to Figure 23-4 for load conditions.

TABLE 23-14: EXAMPLE SPI SLAVE MODE REQUIREMENTS (CKE = 1)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
70	TssL2scH, TssL2scL	$\overline{SS}$↓ to SCK↓ or SCK↑ input		TCY	—	ns	
71	TscH	SCK input high time (Slave mode)	Continuous	1.25 TCY + 30	—	ns	
71A			Single Byte	40	—	ns	**(Note 1)**
72	TscL	SCK input low time (Slave mode)	Continuous	1.25 TCY + 30	—	ns	
72A			Single Byte	40	—	ns	**(Note 1)**
73A	TB2B	Last clock edge of Byte 1 to the first clock edge of Byte 2		1.5 TCY + 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**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
76	TdoF	SDO data output fall time	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
77	TssH2doZ	$\overline{SS}$↑ to SDO output hi-impedance		10	50	ns	
78	TscR	SCK output rise time (Master mode)	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
79	TscF	SCK output fall time (Master mode)	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
80	TscH2doV, TscL2doV	SDO data output valid after SCK edge	PIC18**F**XXXX	—	50	ns	
			PIC18**LF**XXXX	—	150	ns	VDD = 2V
82	TssL2doV	SDO data output valid after $\overline{SS}$↓ edge	PIC18**F**XXXX	—	50	ns	
			PIC18**LF**XXXX	—	150	ns	VDD = 2V
83	TscH2ssH, TscL2ssH	$\overline{SS}$ ↑ after SCK edge		1.5 TCY + 40	—	ns	

Note 1: Requires the use of Parameter # 73A.
2: Only if Parameter # 71A and # 72A are used.

FIGURE 23-16: I^{2}C BUS START/STOP BITS TIMING

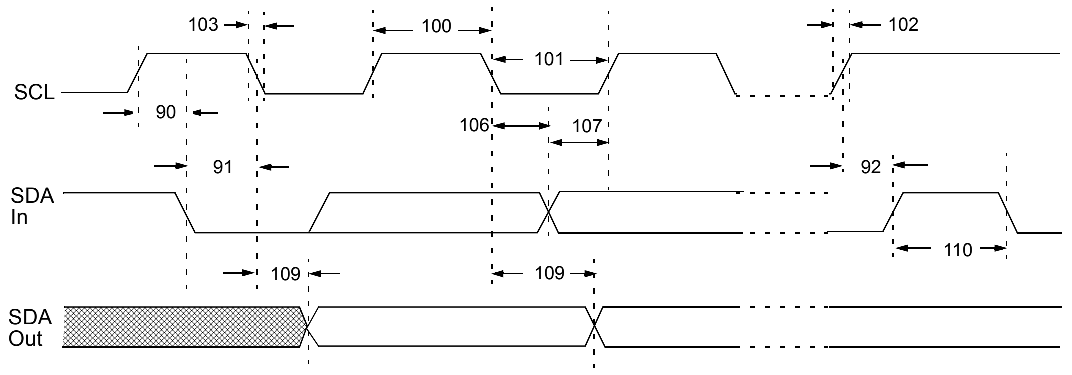


Note: Refer to Figure 23-4 for load conditions.

TABLE 23-15: I^{2}C 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 23-17: I^{2}C BUS DATA TIMING



Note: Refer to Figure 23-4 for load conditions.

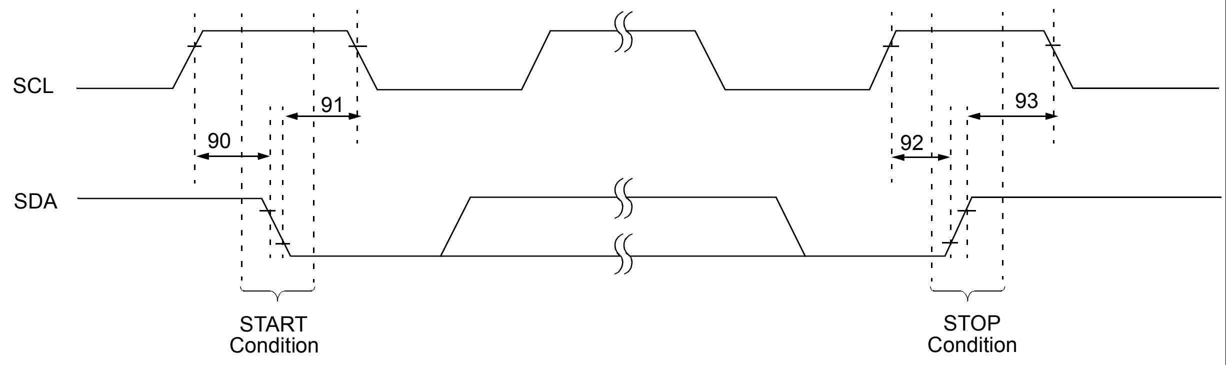
TABLE 23-16: I^{2}C BUS DATA REQUIREMENTS (SLAVE MODE)

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
100	THIGH	Clock high time	100 kHz mode	4.0	—	μs	PIC18FXXX must operate at a minimum of 1.5 MHz
			400 kHz mode	0.6	—	μs	PIC18FXXX must operate at a minimum of 10 MHz
			SSP Module	1.5 TCY	—		
101	TLOW	Clock low time	100 kHz mode	4.7	—	μs	PIC18FXXX must operate at a minimum of 1.5 MHz
			400 kHz mode	1.3	—	μs	PIC18FXXX must operate at a minimum of 10 MHz
			SSP Module	1.5 TCY	—		
102	TR	SDA and SCL rise time	100 kHz mode	—	1000	ns	
			400 kHz mode	20 + 0.1 CB	300	ns	CB is specified to be from 10 to 400 pF
103	TF	SDA and SCL fall time	100 kHz mode	—	1000	ns	VDD ≥ 4.2V
			400 kHz mode	20 + 0.1 CB	300	ns	VDD ≥ 4.2V
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 I^{2}C bus device can be used in a Standard mode I^{2}C 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 I^{2}C bus specification) before the SCL line is released.

FIGURE 23-18: MASTER SSP I^2C BUS START/STOP BITS TIMING WAVEFORMS



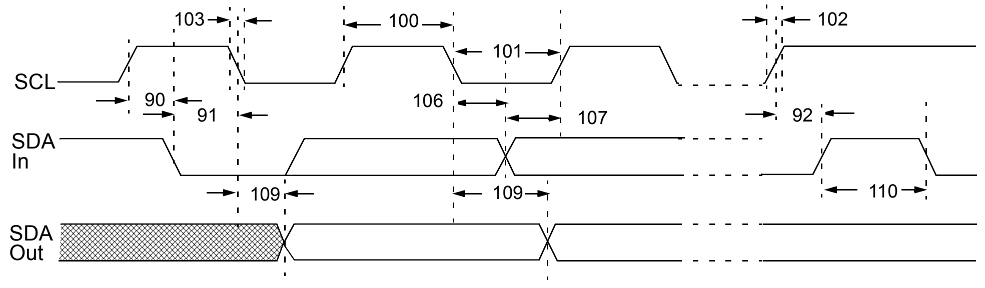
Note: Refer to Figure 23-4 for load conditions.

TABLE 23-17: MASTER SSP I^2C 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)	—		

Note 1: Maximum pin capacitance = 10 pF for all I^2C pins.

FIGURE 23-19: MASTER SSP I^2C BUS DATA TIMING



Note: Refer to Figure 23-4 for load conditions.

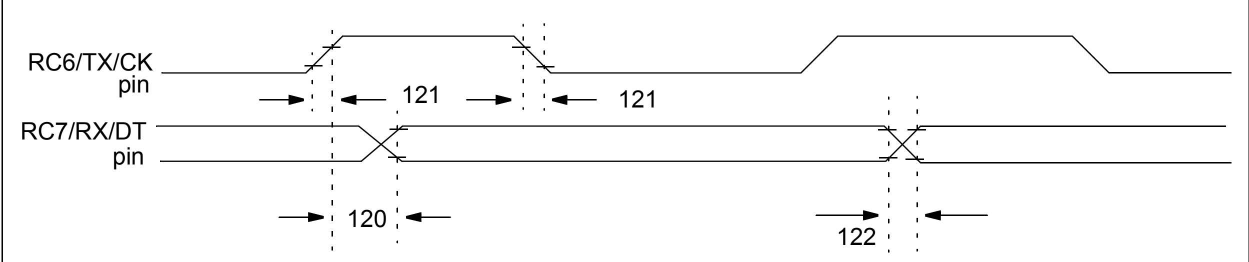
TABLE 23-18: MASTER SSP I²C BUS DATA REQUIREMENTS

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 + 0.1 CB	300	ns	
			1 MHz mode(1)	—	300	ns	
103	TF	SDA and SCL fall time	100 kHz mode	—	1000	ns	VDD ≥ 4.2V
			400 kHz mode	20 + 0.1 CB	300	ns	VDD ≥ 4.2V
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	
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	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	
D102	CB	Bus capacitive loading		—	400	pF	

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.

FIGURE 23-20: USART SYNCHRONOUS TRANSMISSION (MASTER/SLAVE) TIMING

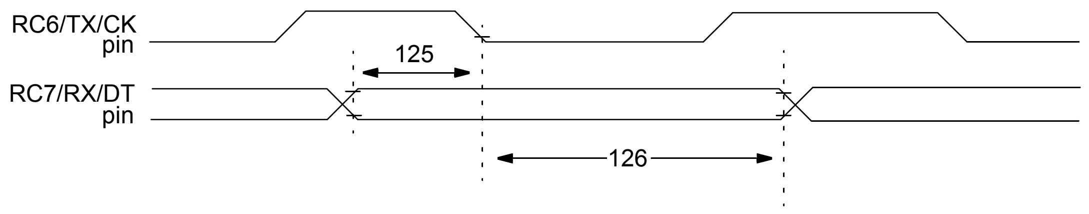


Note: Refer to Figure 23-4 for load conditions.

TABLE 23-19: USART SYNCHRONOUS TRANSMISSION REQUIREMENTS

Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
120	TckH2dtV	<u>SYNC XMIT (MASTER & SLAVE)</u> Clock high to data out valid	PIC18**F**XXXX	—	50	ns	
			PIC18**LF**XXXX	—	150	ns	VDD = 2V
121	Tckr	Clock out rise time and fall time (Master mode)	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V
122	Tdtr	Data out rise time and fall time	PIC18**F**XXXX	—	25	ns	
			PIC18**LF**XXXX	—	60	ns	VDD = 2V

FIGURE 23-21: USART SYNCHRONOUS RECEIVE (MASTER/SLAVE) TIMING



Note: Refer to Figure 23-4 for load conditions.

TABLE 23-20: USART SYNCHRONOUS RECEIVE REQUIREMENTS

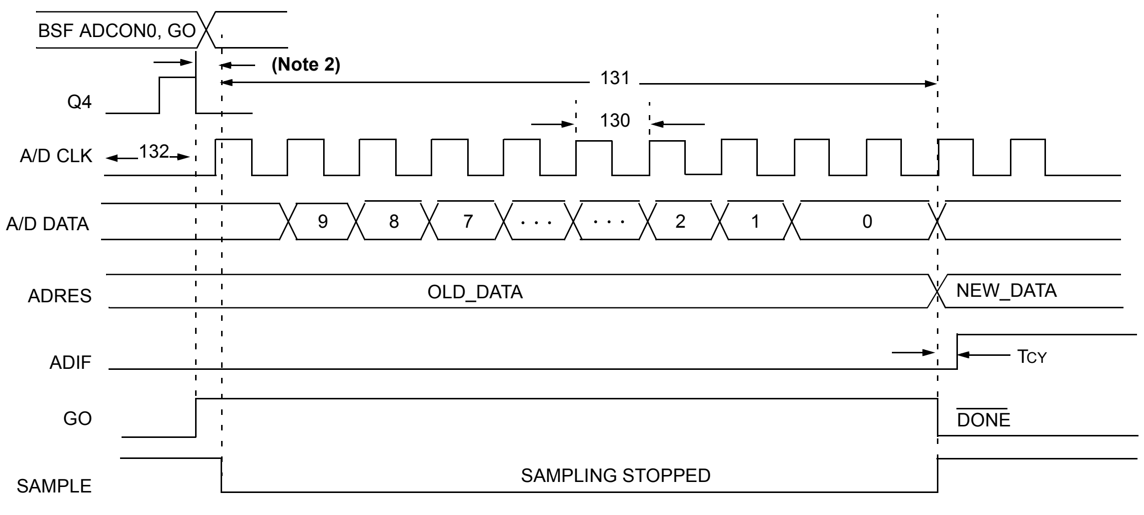
Param. No.	Symbol	Characteristic		Min	Max	Units	Conditions
125	TdtV2ckl	<u>SYNC RCV (MASTER & SLAVE)</u> Data hold before CK ↓ (DT hold time)		10	—	ns	
126	TckL2dtl	Data hold after CK ↓ (DT hold time)	PIC18**F**XXXX	15	—	ns	
			PIC18**LF**XXXX	20	—	ns	VDD = 2V

TABLE 23-21: A/D CONVERTER CHARACTERISTICS: PIC18FXX39 (INDUSTRIAL, EXTENDED) PIC18LFXX39 (INDUSTRIAL)

Param No.	Symbol	Characteristic	Min	Typ	Max	Units	Conditions
A01	NR	Resolution	—	—	10	bit	
A03	EIL	Integral linearity error	—	—	<±1	LSb	VREF = VDD = 5.0V
A04	EDL	Differential linearity error	—	—	<±1	LSb	VREF = VDD = 5.0V
A05	EG	Gain error	—	—	<±1	LSb	VREF = VDD = 5.0V
A06	EOFF	Offset error	—	—	<±1.5	LSb	VREF = VDD = 5.0V
A10	—	Monotonicity	guaranteed(2)			—	VSS ≤ VAIN ≤ VREF
A20 A20A	VREF	Reference Voltage (VREFH – VREFL)	1.8V 3V	— —	— —	V V	VDD < 3.0V VDD ≥ 3.0V
A21	VREFH	Reference voltage High	AVSS	—	AVDD + 0.3V	V	
A22	VREFL	Reference voltage Low	AVSS – 0.3V	—	VREFH	V	
A25	VAIN	Analog input voltage	AVSS – 0.3V	—	AVDD + 0.3V	V	VDD ≥ 2.5V **(Note 3)**
A30	ZAIN	Recommended impedance of analog voltage source	—	—	2.5	kΩ	**(Note 4)**
A50	IREF	VREF input current **(Note 1)**	— —	— —	5 150	μA μA	During VAIN acquisition During A/D conversion cycle

Note 1: Vss ≤ VAIN ≤ VREF
2: The A/D conversion result never decreases with an increase in the Input Voltage, and has no missing codes.
3: For VDD < 2.5V, VAIN should be limited to < .5 VDD.
4: Maximum allowed impedance for analog voltage source is 10 kΩ. This requires higher acquisition times.

FIGURE 23-22: A/D CONVERSION TIMING



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 23-22: A/D CONVERSION REQUIREMENTS

Param No.	Symbol	Characteristic		Min	Max	Units	Conditions
130	TAD	A/D clock period	PIC18**F**XXXX	1.6	20[4]	μs	TOSC based
			PIC18**LF**XXXX	2.0	6.0	μs	A/D RC mode
131	TCNV	Conversion time (not including acquisition time) **(Note 1)**		11	12	TAD	
132	TACQ	Acquisition time **(Note 2)**		5 10	— —	μs μs	VREF = VDD = 5.0V VREF = VDD = 2.5V
135	TSWC	Switching Time from convert → sample		—	**(Note 3)**		

Note 1: ADRES register may be read on the following TCY cycle.

2: The time for the holding capacitor to acquire the “New” input voltage, when the new input value has not changed by more than 1 LSB from the last sampled voltage. The source impedance (*R*S) on the input channels is 50Ω. See Section 18.0 for more information on acquisition time consideration.

3: On the next Q4 cycle of the device clock.

4: The time of the A/D clock period is dependent on the device frequency and the TAD clock divider.

24.0 DC AND AC CHARACTERISTICS GRAPHS AND TABLES

Note: The graphs and tables provided following this note are a statistical summary based on a limited number of samples and are provided for informational purposes only. The performance characteristics listed herein are not tested or guaranteed. In some graphs or tables, the data presented may be outside the specified operating range (e.g., outside specified power supply range) and therefore, outside the warranted range.

"Typical" represents the mean of the distribution at 25°C. "Maximum" or "minimum" represents (mean + 3σ) or (mean - 3σ) respectively, where σ is a standard deviation, over the whole temperature range.

FIGURE 24-1: TYPICAL IDD vs. FOSC OVER VDD (HS MODE)

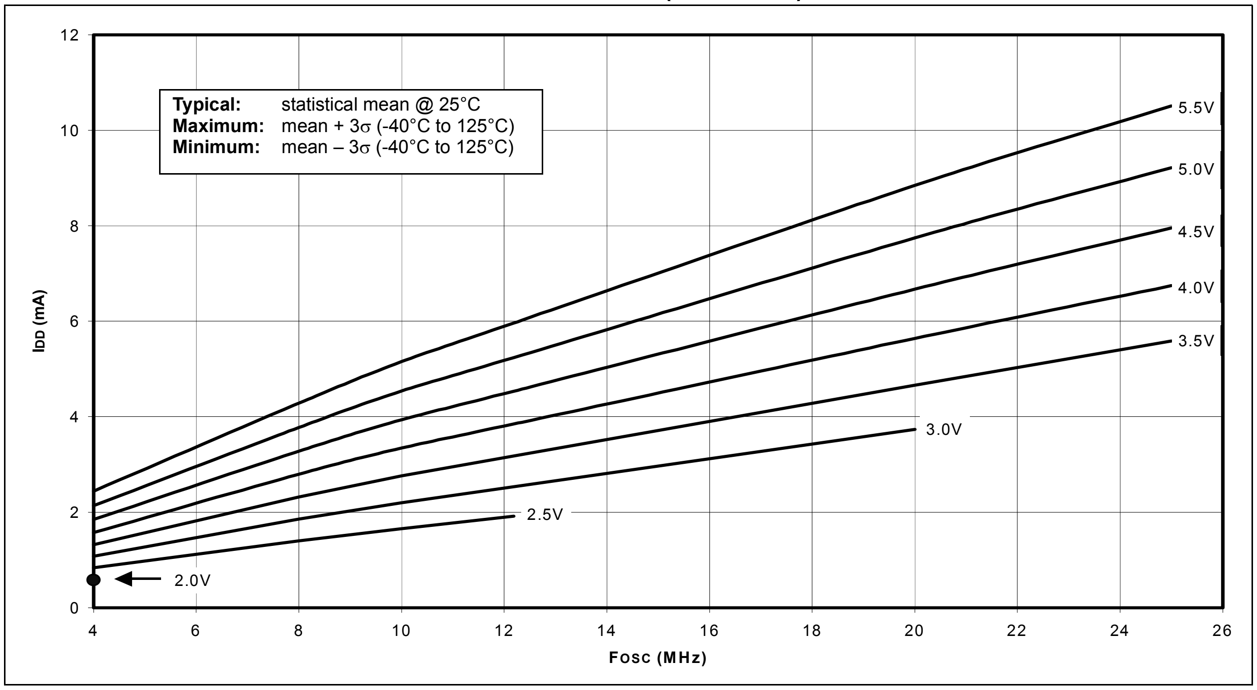


FIGURE 24-2: MAXIMUM IDD vs. FOSC OVER VDD (HS MODE)

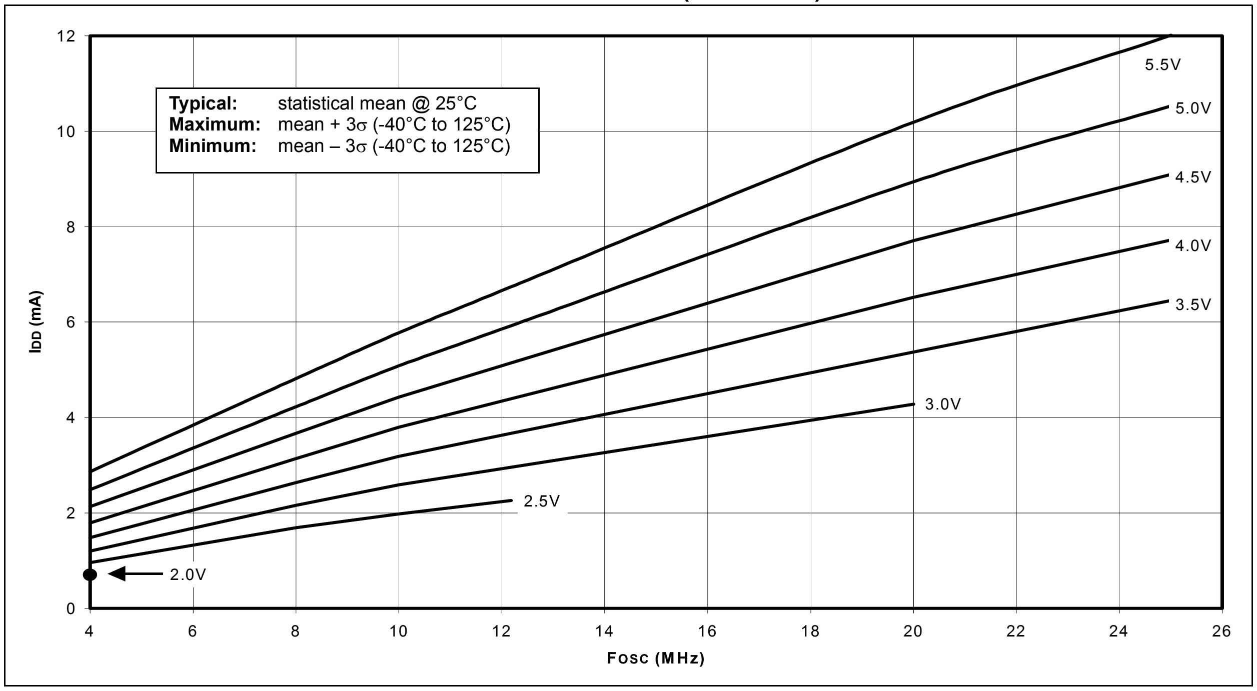


FIGURE 24-3: TYPICAL IDD vs. FOSC OVER VDD (HS/PLL MODE)

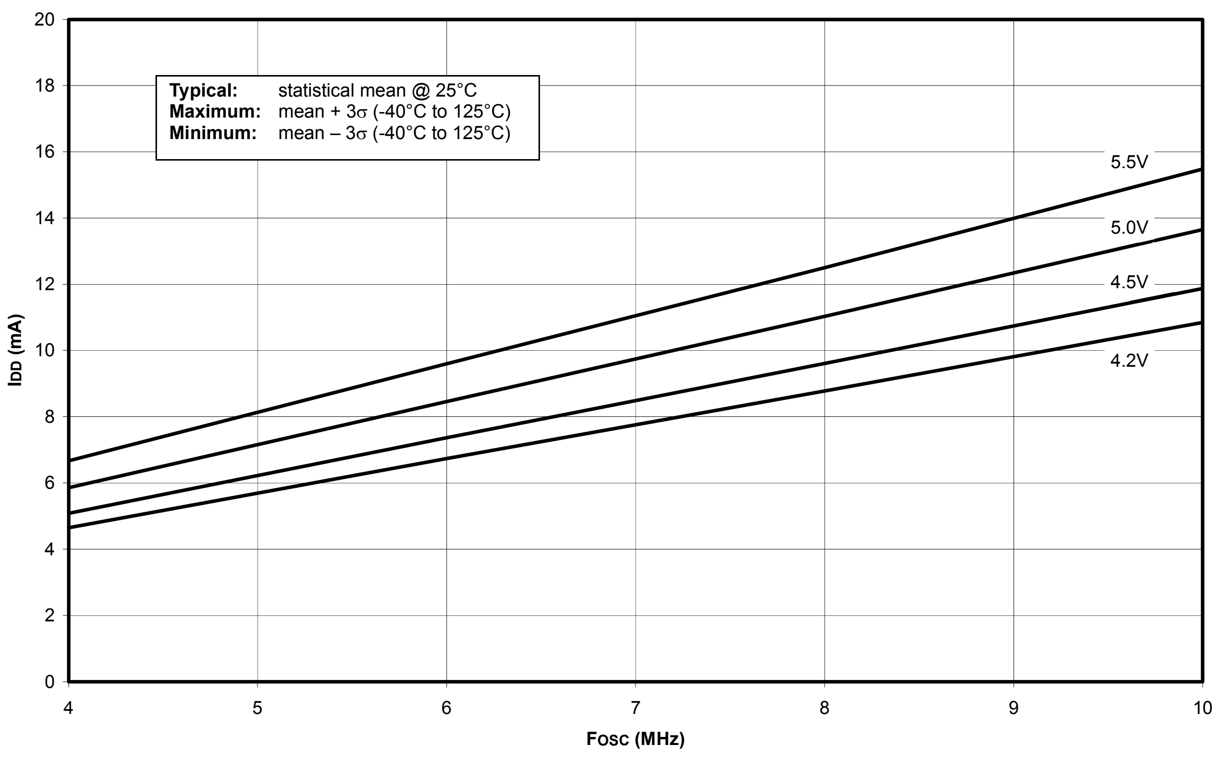


FIGURE 24-4: MAXIMUM IDD vs. FOSC OVER VDD (HS/PLL MODE)

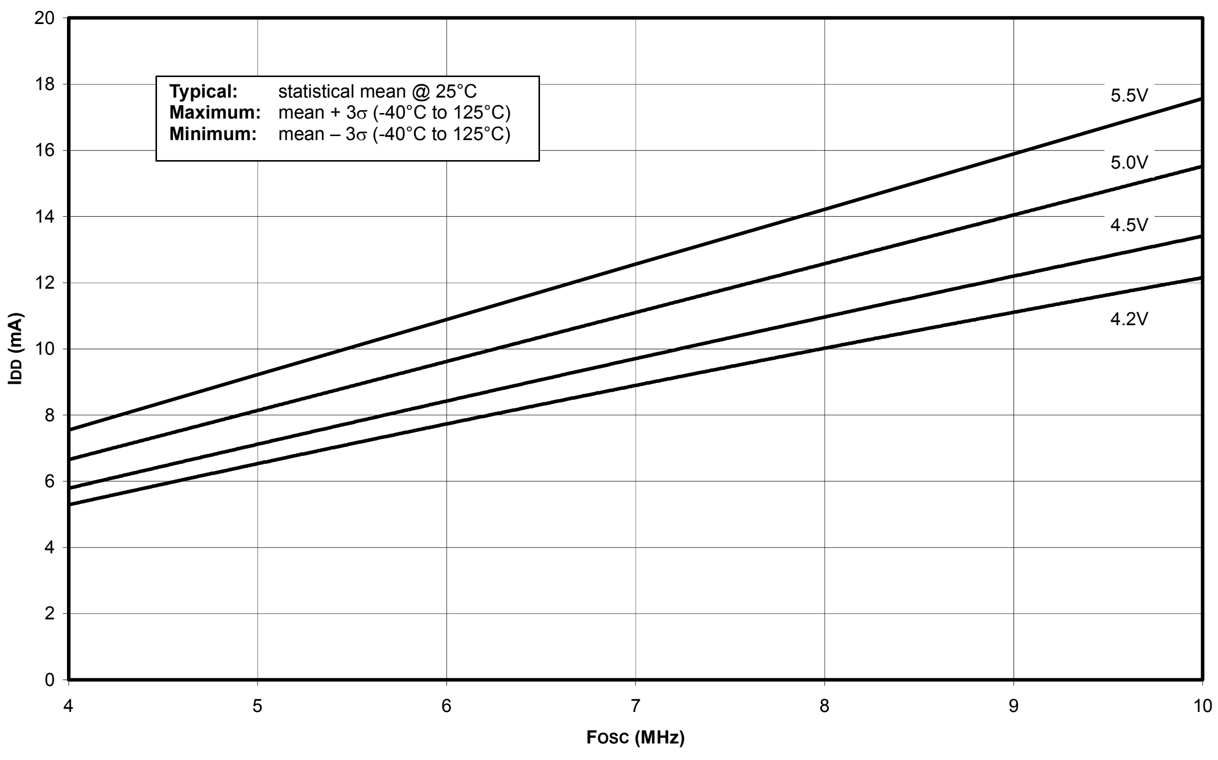


FIGURE 24-5: TYPICAL IDD vs. FOSC OVER VDD (EC MODE)

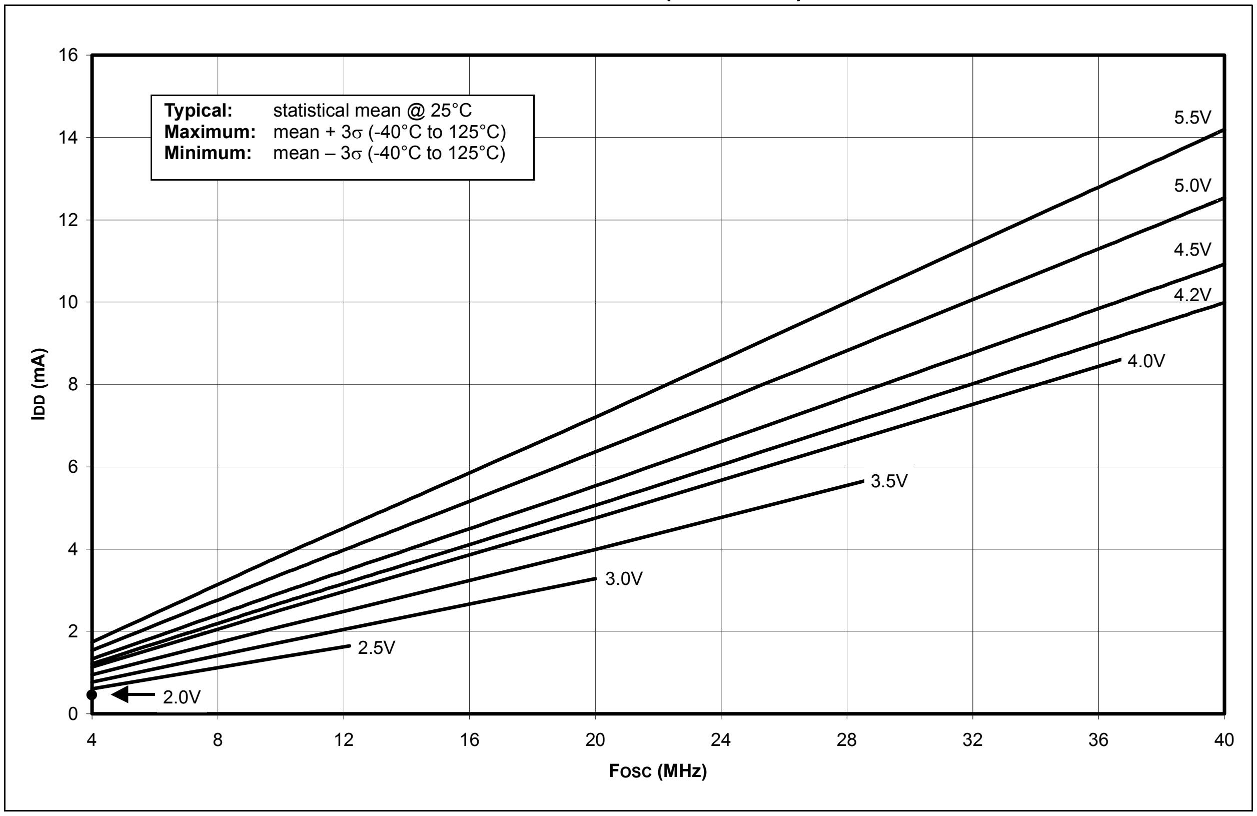


FIGURE 24-6: MAXIMUM IDD vs. FOSC OVER VDD (EC MODE)

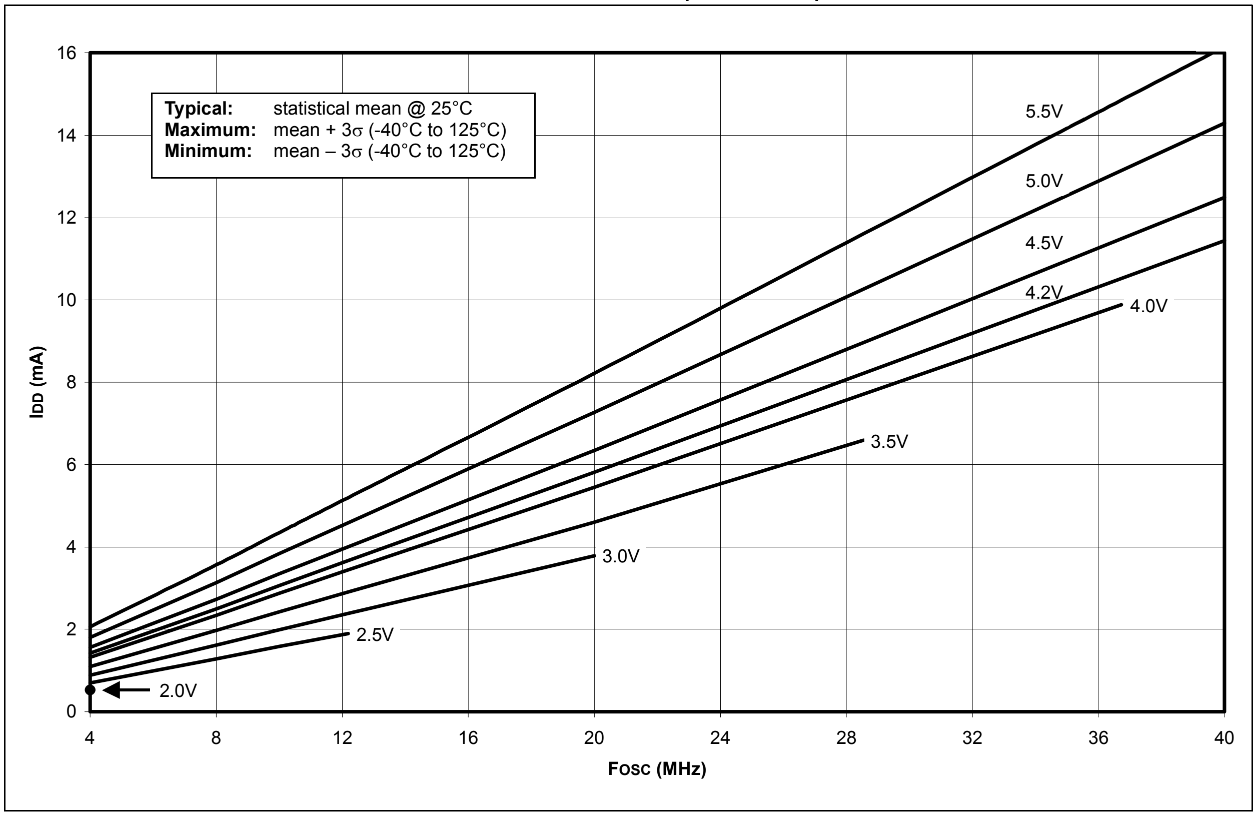


FIGURE 24-7: IPD vs. VDD, -40°C TO +125°C (SLEEP MODE, ALL PERIPHERALS DISABLED)

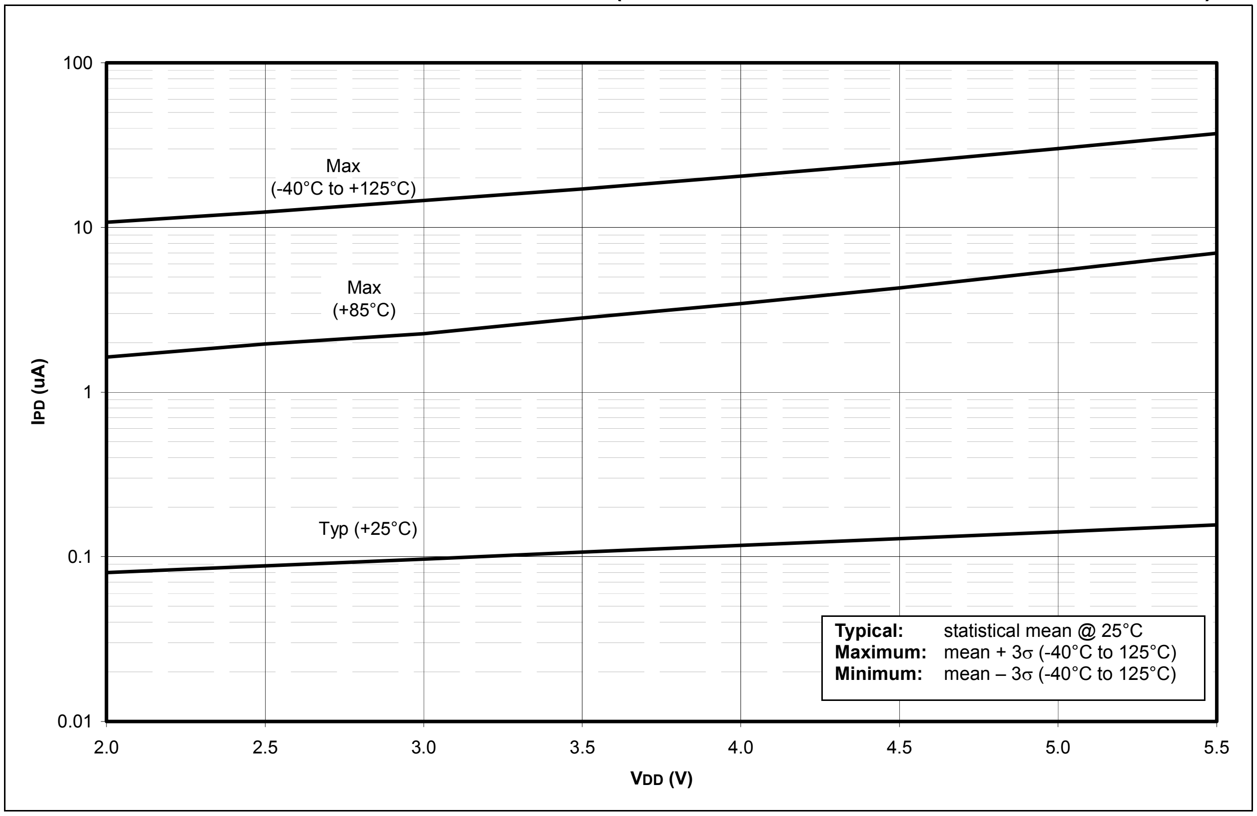
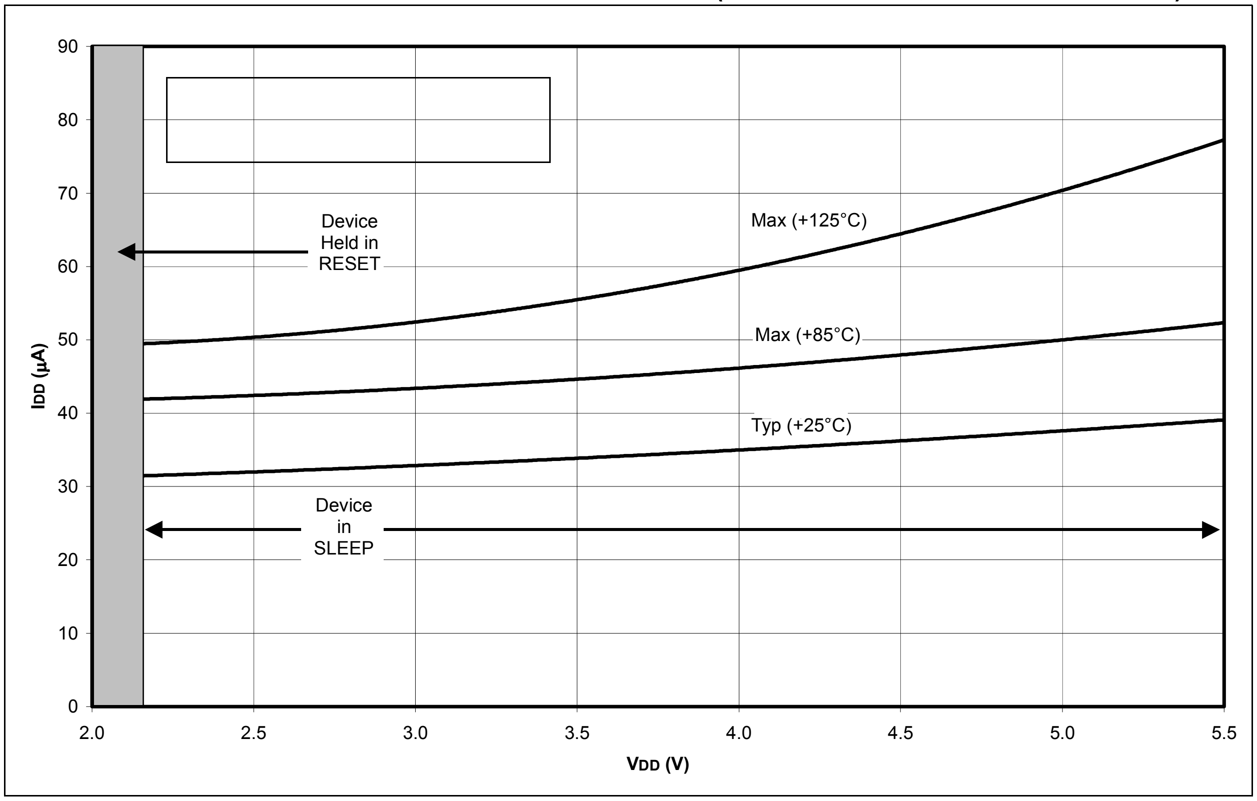


FIGURE 24-8: ΔIBOR vs. VDD OVER TEMPERATURE (BOR ENABLED, VBOR = 2.00 - 2.16V)



Preliminary

FIGURE 24-9: TYPICAL AND MAXIMUM ΔIWDT vs. VDD OVER TEMPERATURE (WDT ENABLED)

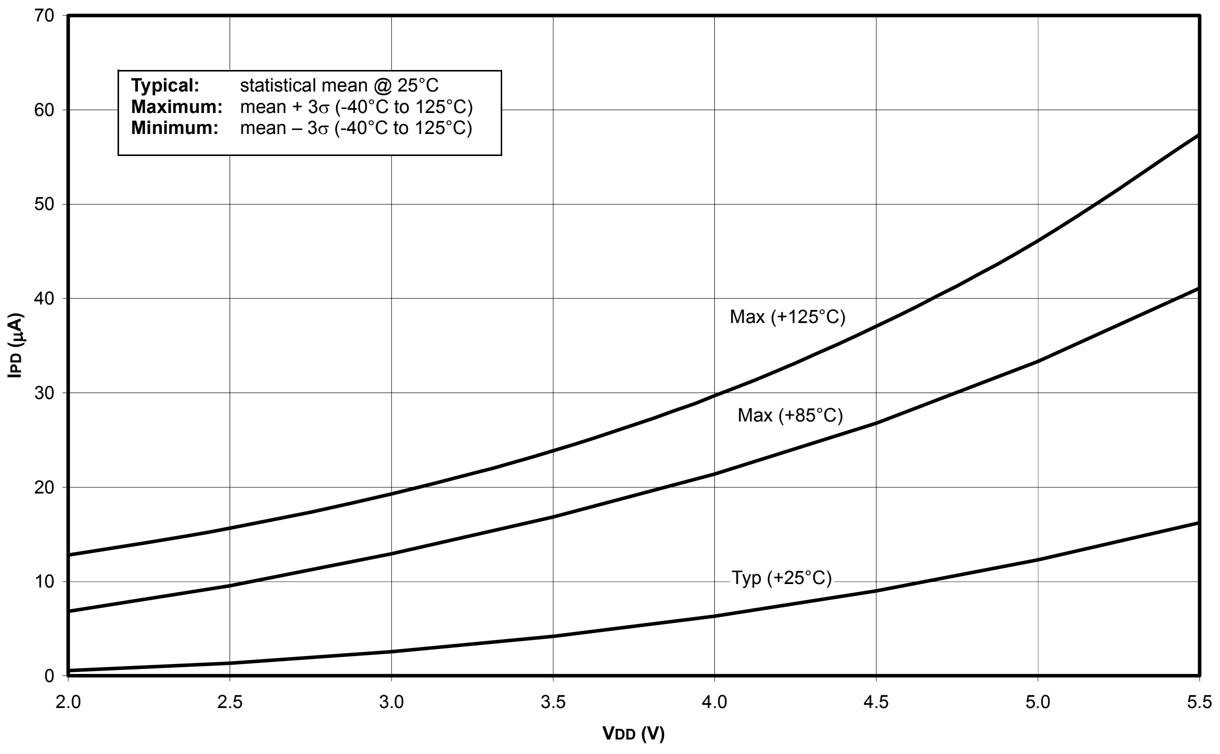


FIGURE 24-10: TYPICAL, MINIMUM AND MAXIMUM WDT PERIOD vs. VDD (-40°C TO +125°C)

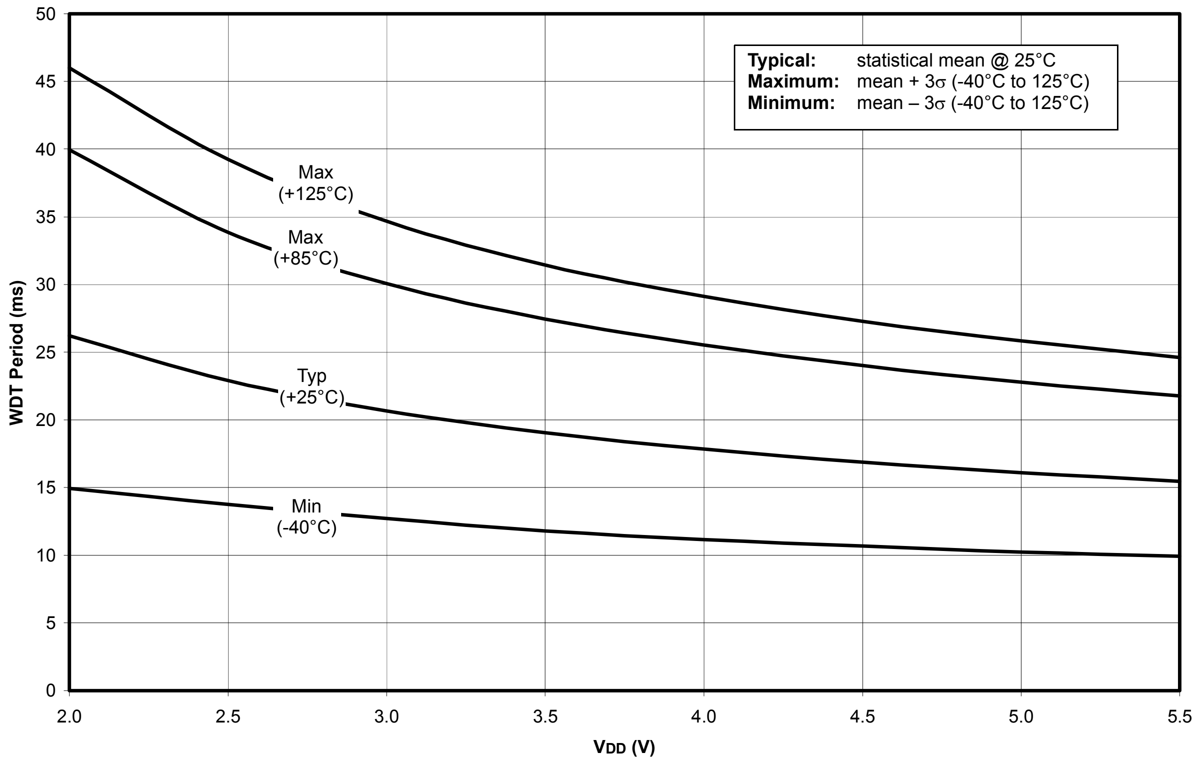


FIGURE 24-11: ΔI_{LVD} vs. V_{DD} OVER TEMPERATURE (LVD ENABLED, V_{LVD} = 4.5 - 4.78V)

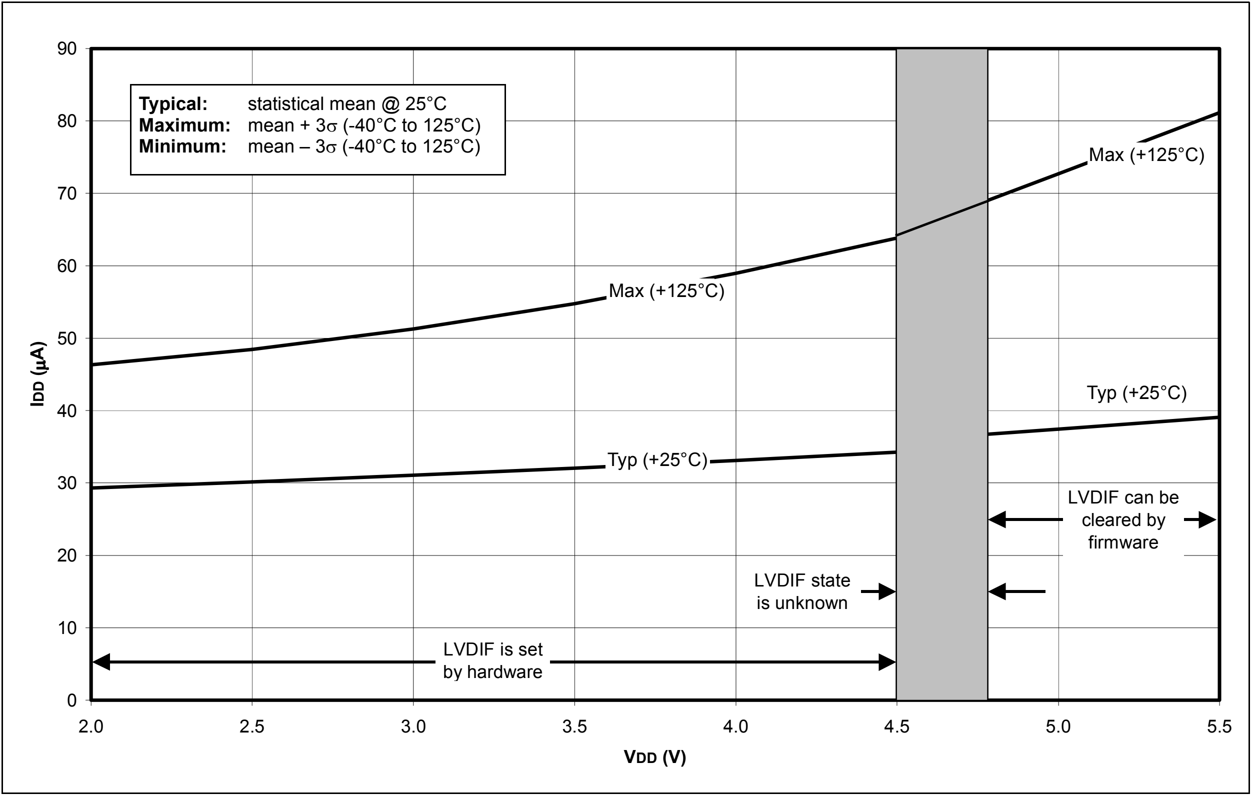


FIGURE 24-12: TYPICAL, MINIMUM AND MAXIMUM V_{OH} vs. I_{OH} (V_{DD} = 5V, -40°C TO +125°C)

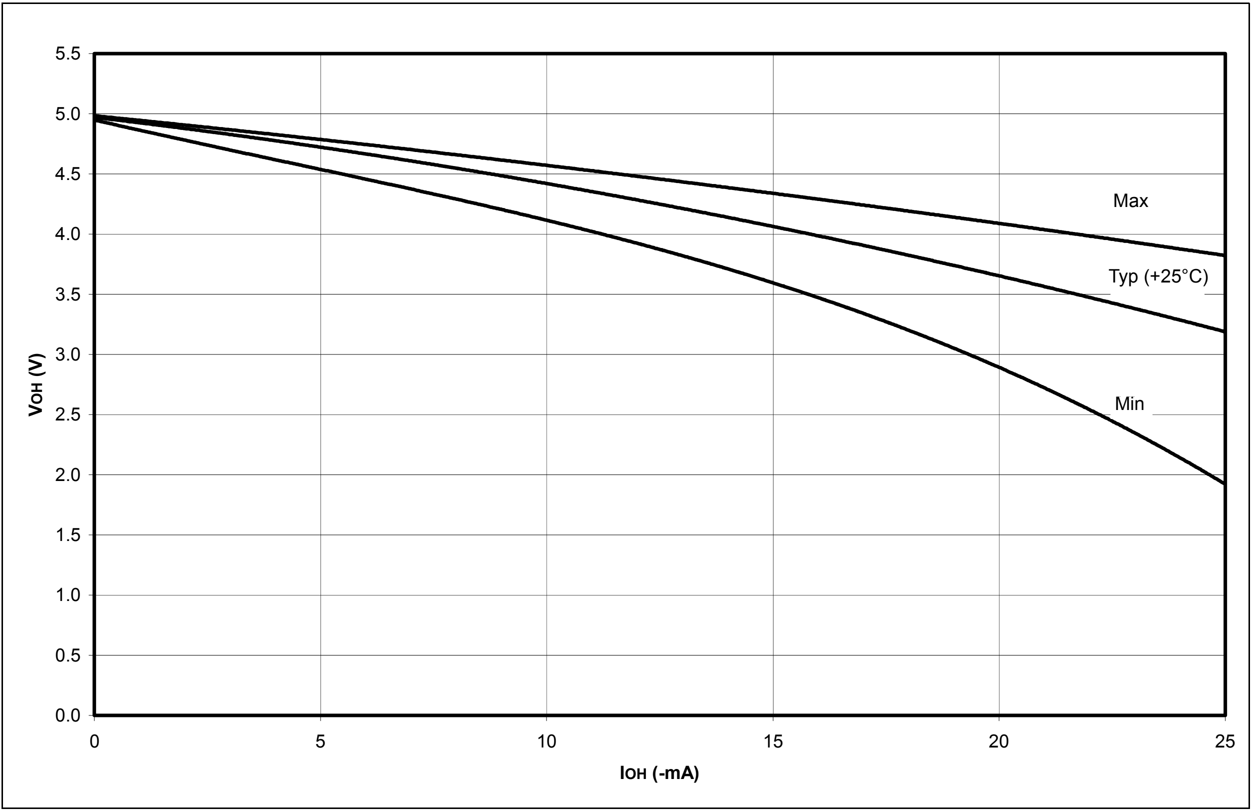


FIGURE 24-13: TYPICAL, MINIMUM AND MAXIMUM V_{OH} vs. I_{OH} (V_{DD} = 3V, -40°C TO +125°C)

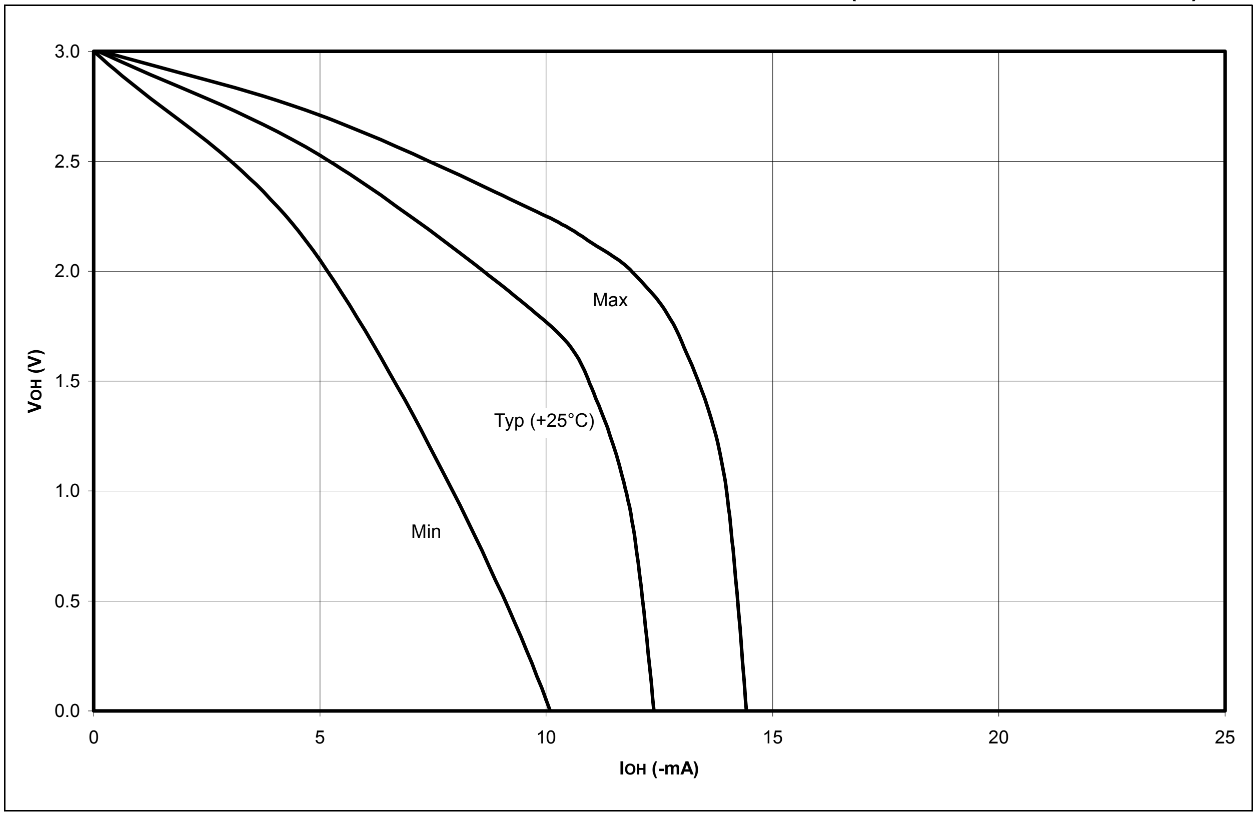
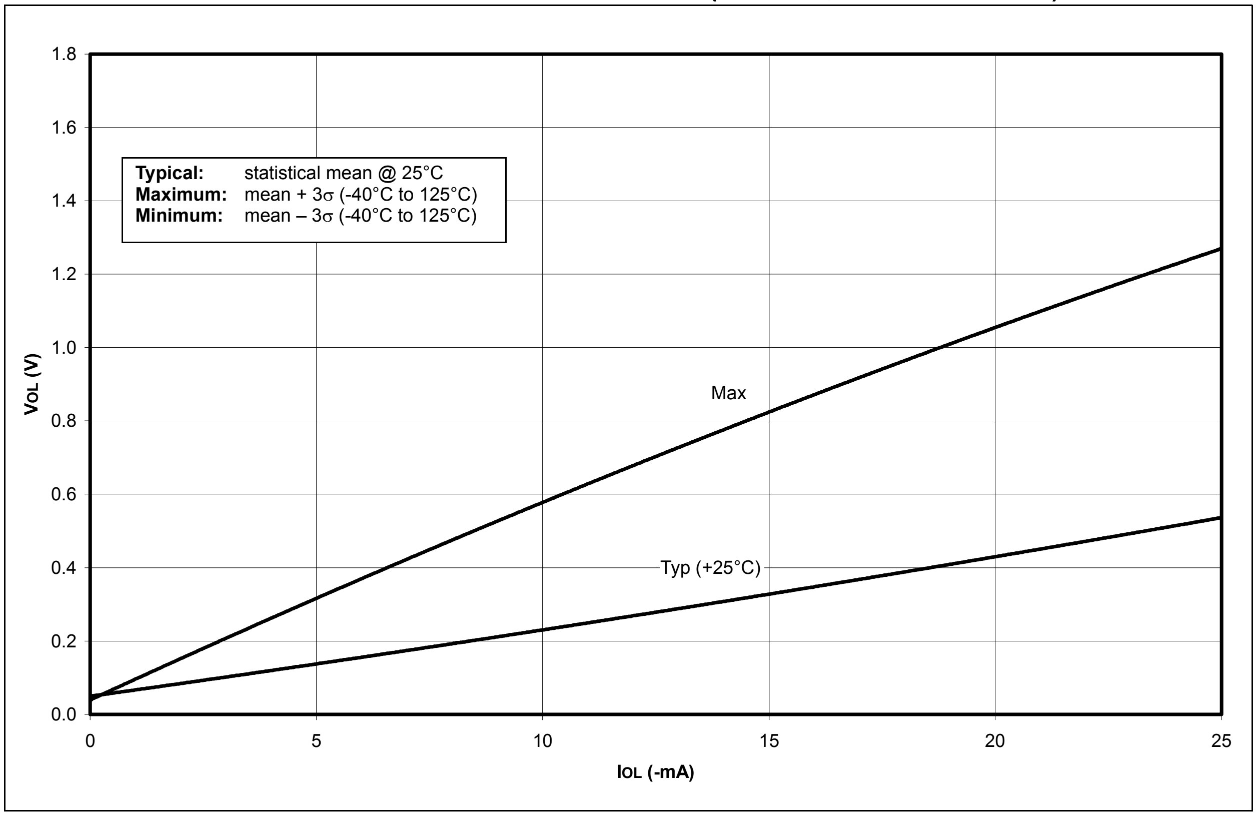


FIGURE 24-14: TYPICAL AND MAXIMUM V_{OL} vs. I_{OL} (V_{DD} = 5V, -40°C TO +125°C)



Typical:	statistical mean @ 25°C
Maximum:	mean + 3σ (-40°C to 125°C)
Minimum:	mean – 3σ (-40°C to 125°C)

FIGURE 24-15: TYPICAL AND MAXIMUM V_{OL} vs. I_{OL} (V_{DD} = 3V, -40°C TO +125°C)

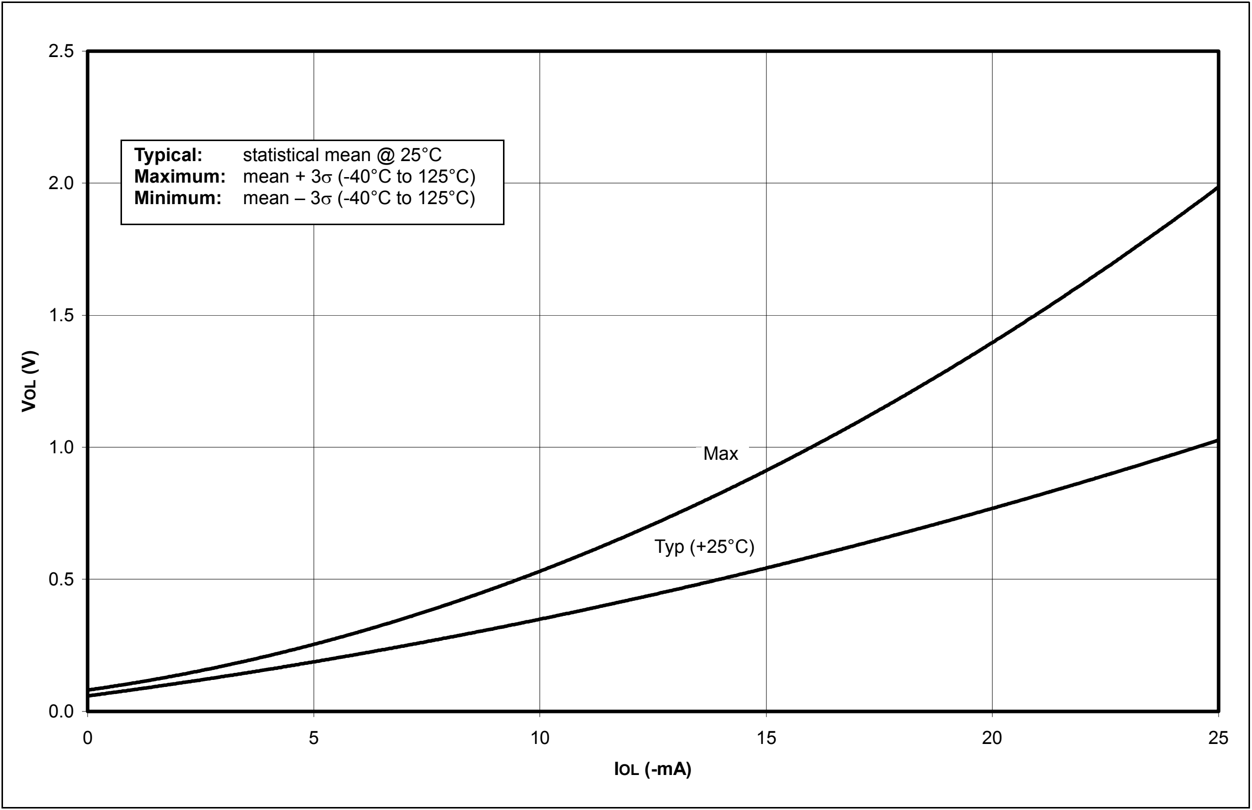
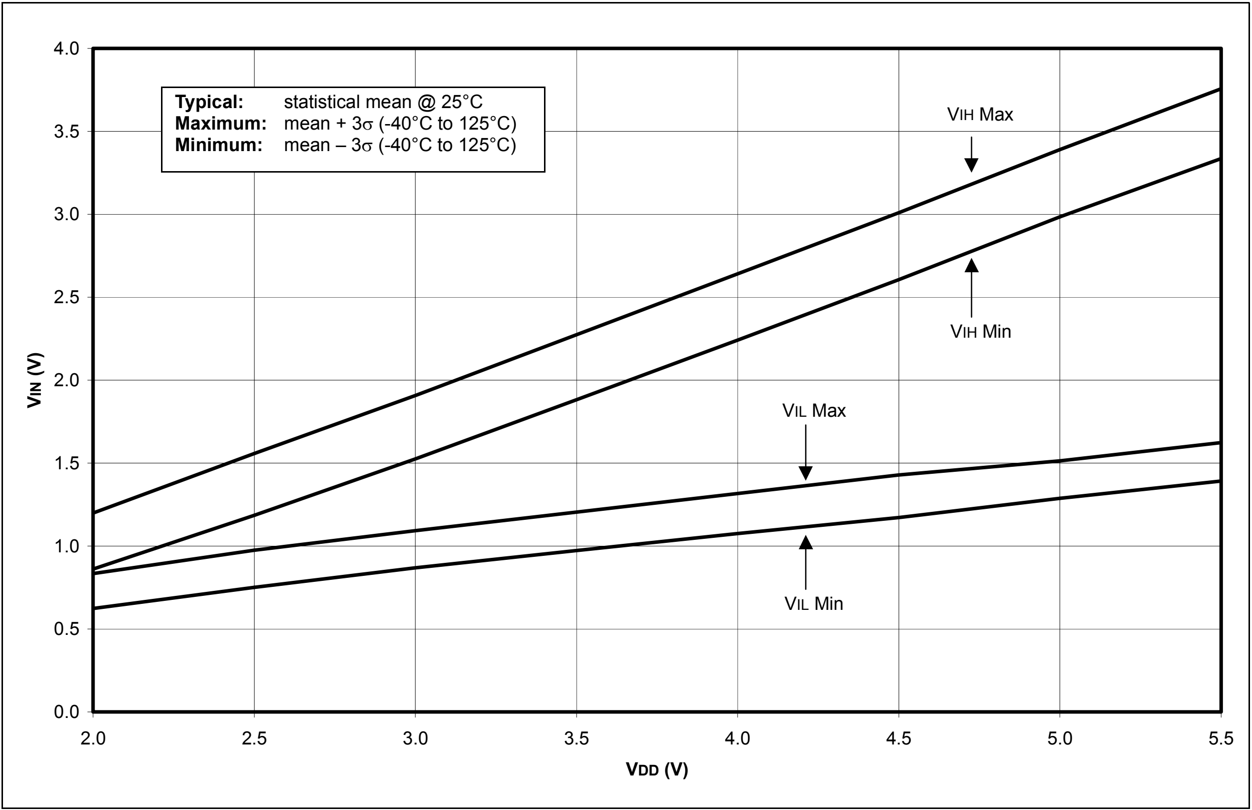


FIGURE 24-16: MINIMUM AND MAXIMUM V_{IN} vs. V_{DD} (ST INPUT, -40°C TO +125°C)



Preliminary

FIGURE 24-17: MINIMUM AND MAXIMUM V_{IN} vs. V_{DD} (TTL INPUT, -40°C TO +125°C)

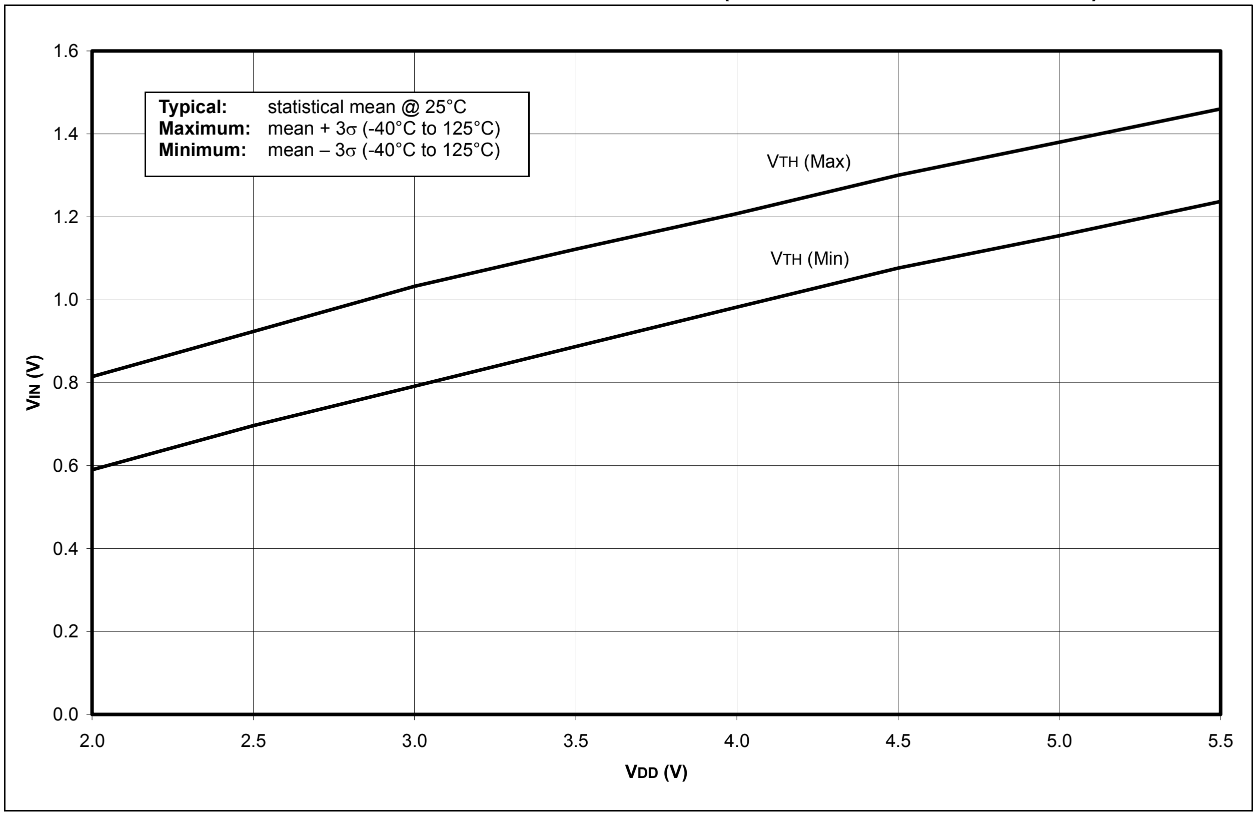


FIGURE 24-18: MINIMUM AND MAXIMUM V_{IN} vs. V_{DD} (I^2C INPUT, -40°C TO +125°C)

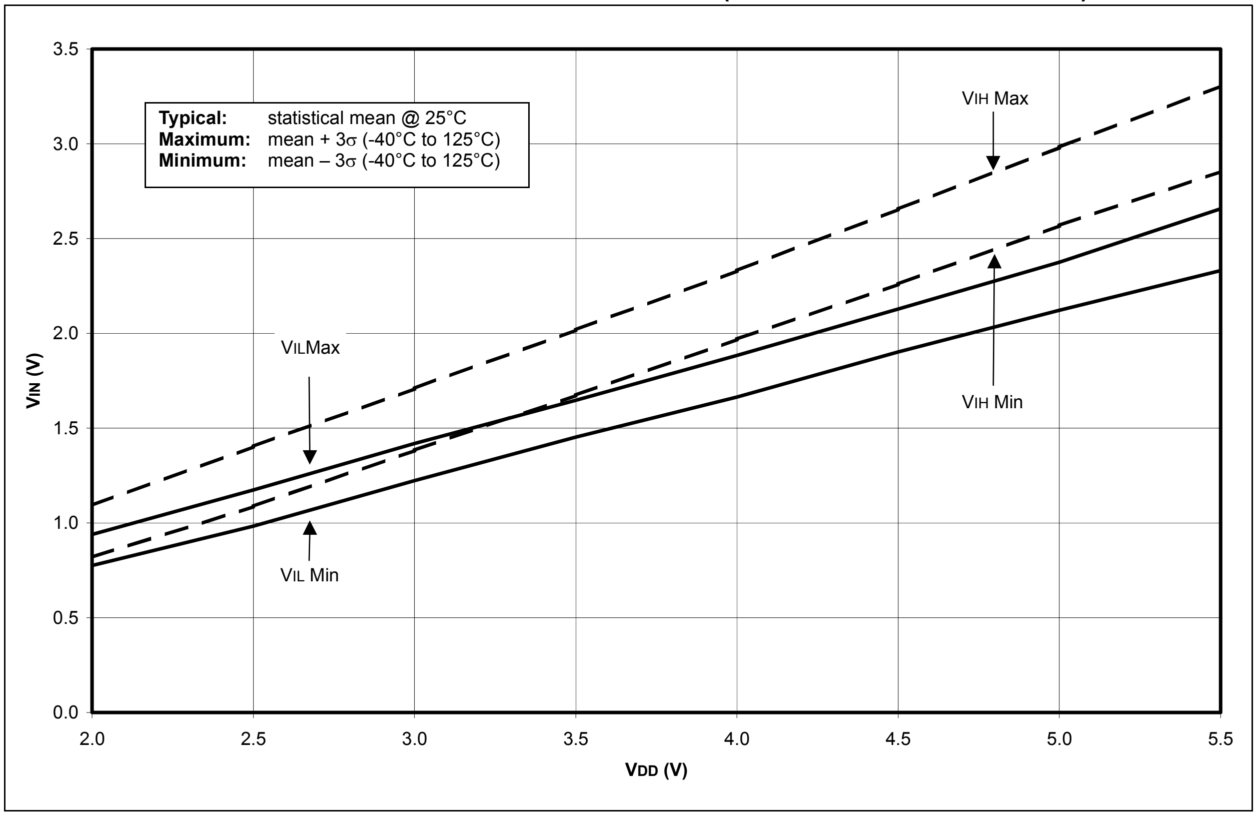


FIGURE 24-19: A/D NON-LINEARITY vs. VREFH (VDD = VREFH, -40°C TO +125°C)

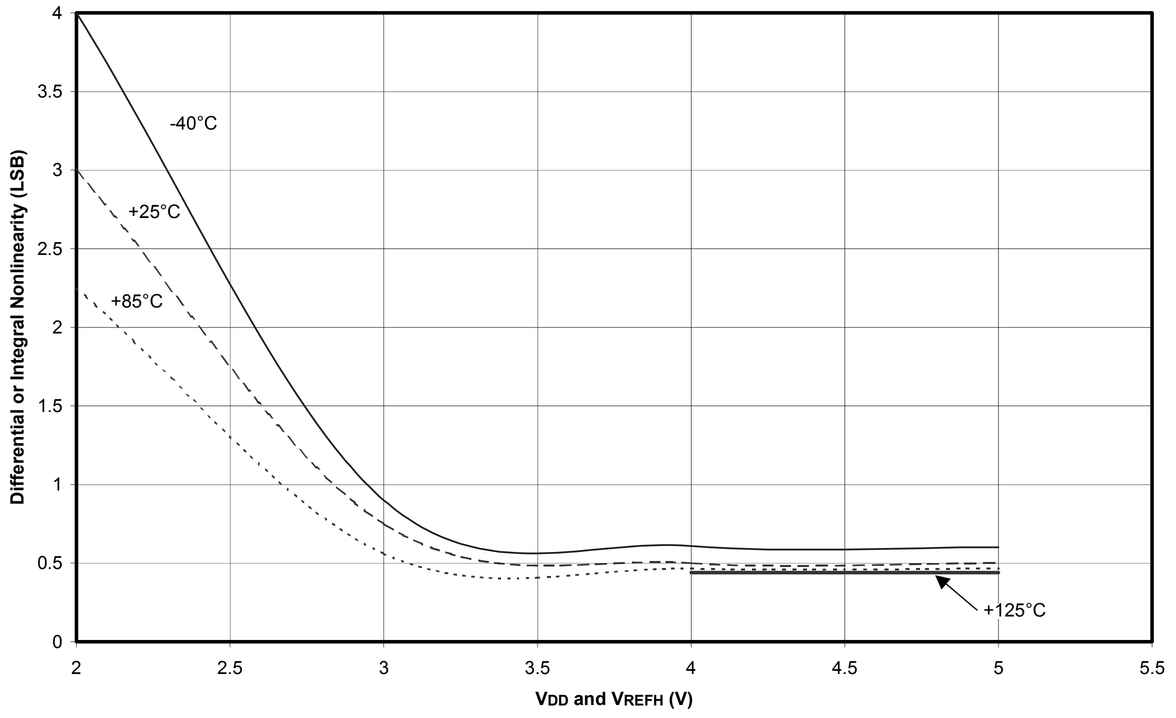
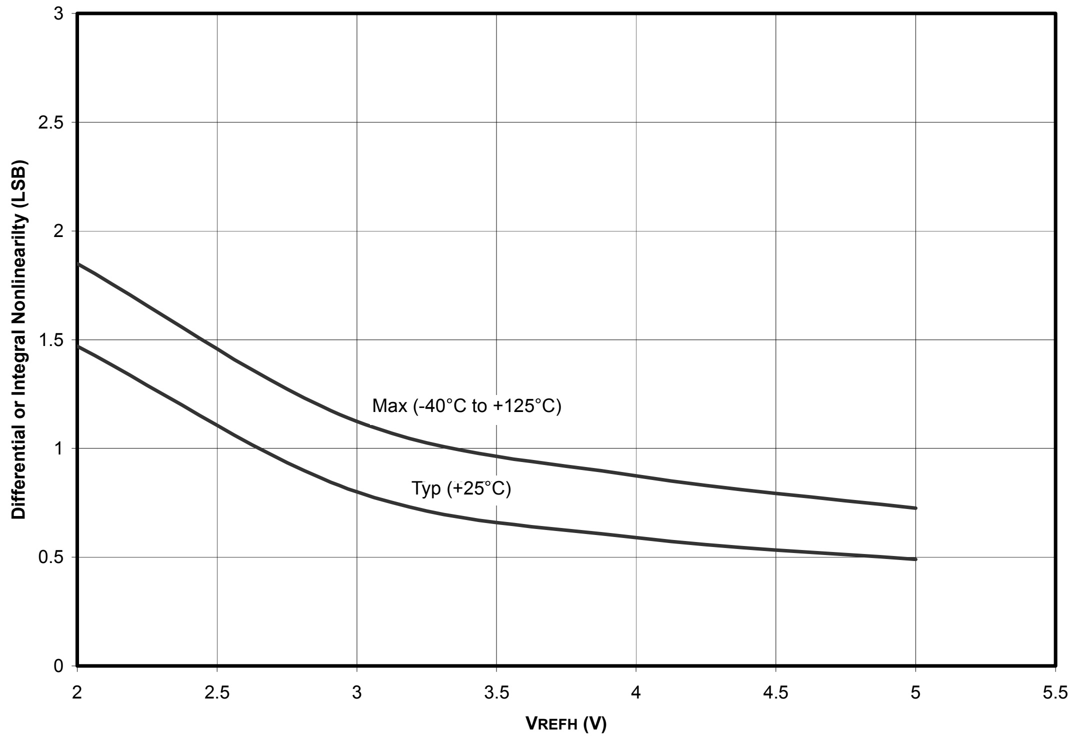


FIGURE 24-20: A/D NON-LINEARITY vs. VREFH (VDD = 5V, -40°C TO +125°C)

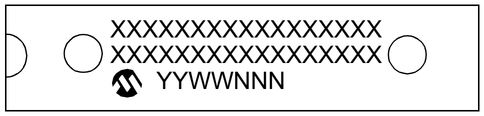


Preliminary

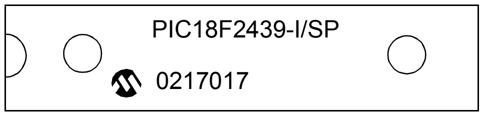
25.0 PACKAGING INFORMATION

25.1 Package Marking Information

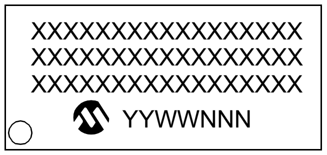
28-Lead PDIP (Skinny DIP)



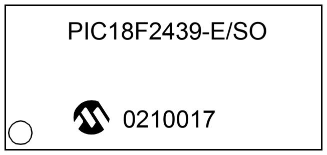
Example



28-Lead SOIC



Example



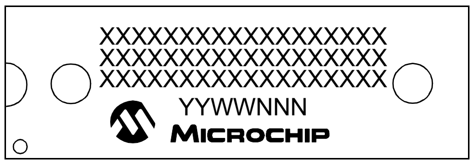
Legend:	XX...X	Customer specific information*
	Y	Year code (last digit of calendar year)
	YY	Year code (last 2 digits of calendar year)
	WW	Week code (week of January 1 is week ‘01’)
	NNN	Alphanumeric traceability code

Note: In the event the full Microchip part number cannot be marked on one line, it will be carried over to the next line thus limiting the number of available characters for customer specific information.

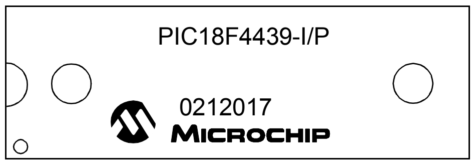
* Standard PICmicro device marking consists of Microchip part number, year code, week code, and traceability code. For PICmicro device marking beyond this, certain price adders apply. Please check with your Microchip Sales Office. For QTP devices, any special marking adders are included in QTP price.

Package Marking Information (Cont'd)

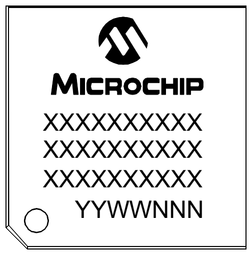
40-Lead PDIP



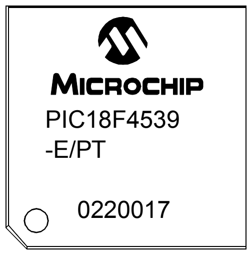
Example



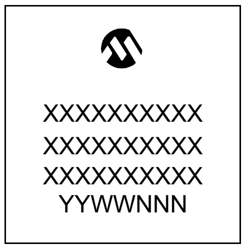
44-Lead TQFP



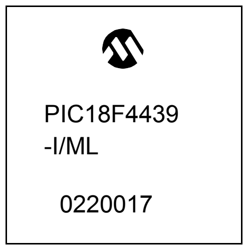
Example



44-Lead QFN



Example

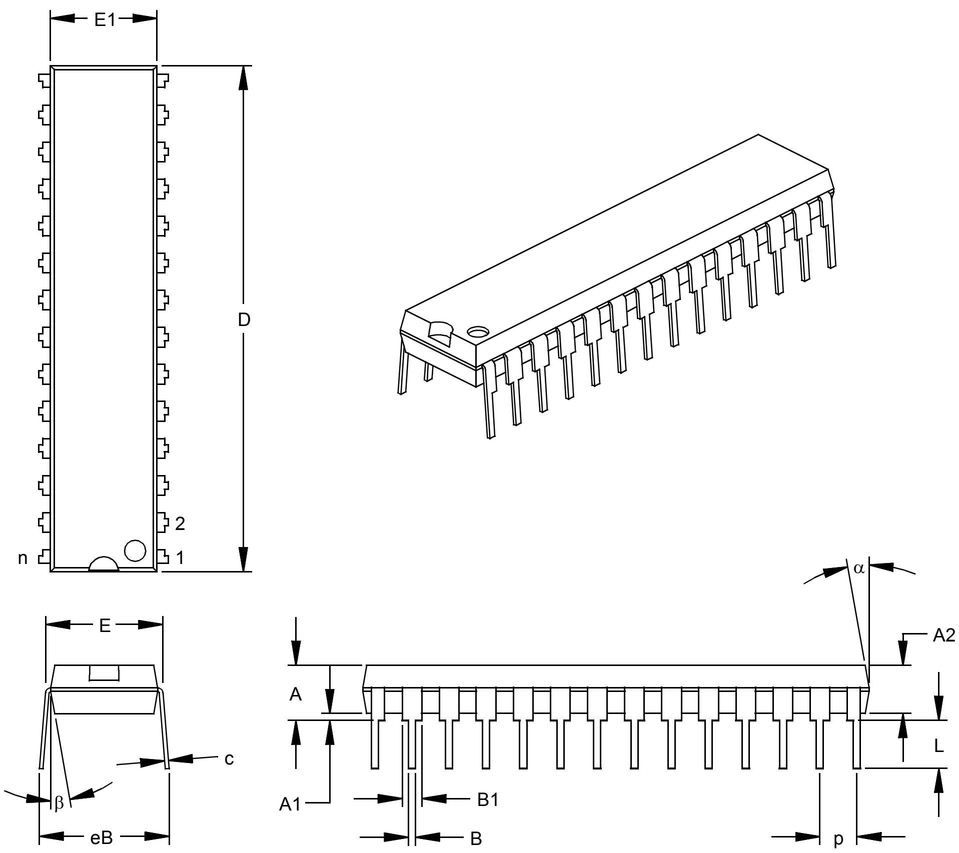


Preliminary

25.2 Package Details

The following sections give the technical details of the packages.

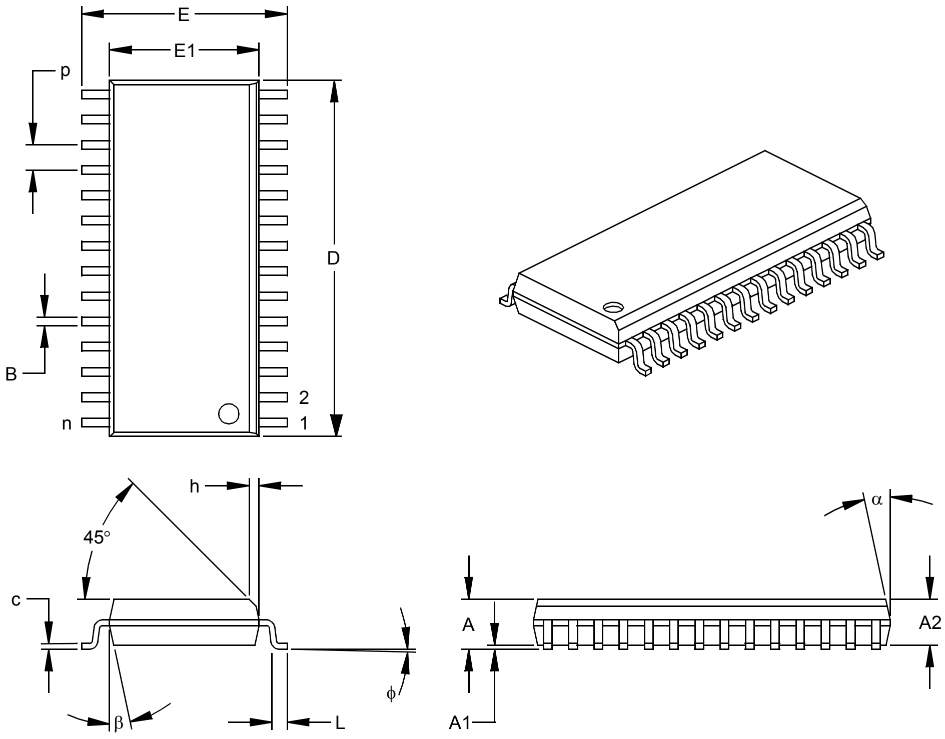
28-Lead Skinny Plastic Dual In-line (SP) – 300 mil (PDIP)



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		28			28	
Pitch	p		.100			2.54	
Top to Seating Plane	A	.140	.150	.160	3.56	3.81	4.06
Molded Package Thickness	A2	.125	.130	.135	3.18	3.30	3.43
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.300	.310	.325	7.62	7.87	8.26
Molded Package Width	E1	.275	.285	.295	6.99	7.24	7.49
Overall Length	D	1.345	1.365	1.385	34.16	34.67	35.18
Tip to Seating Plane	L	.125	.130	.135	3.18	3.30	3.43
Lead Thickness	c	.008	.012	.015	0.20	0.29	0.38
Upper Lead Width	B1	.040	.053	.065	1.02	1.33	1.65
Lower Lead Width	B	.016	.019	.022	0.41	0.48	0.56
Overall Row Spacing §	eB	.320	.350	.430	8.13	8.89	10.92
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter
§ Significant Characteristic
Notes:
Dimension D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MO-095
Drawing No. C04-070

28-Lead Plastic Small Outline (SO) – Wide, 300 mil (SOIC)

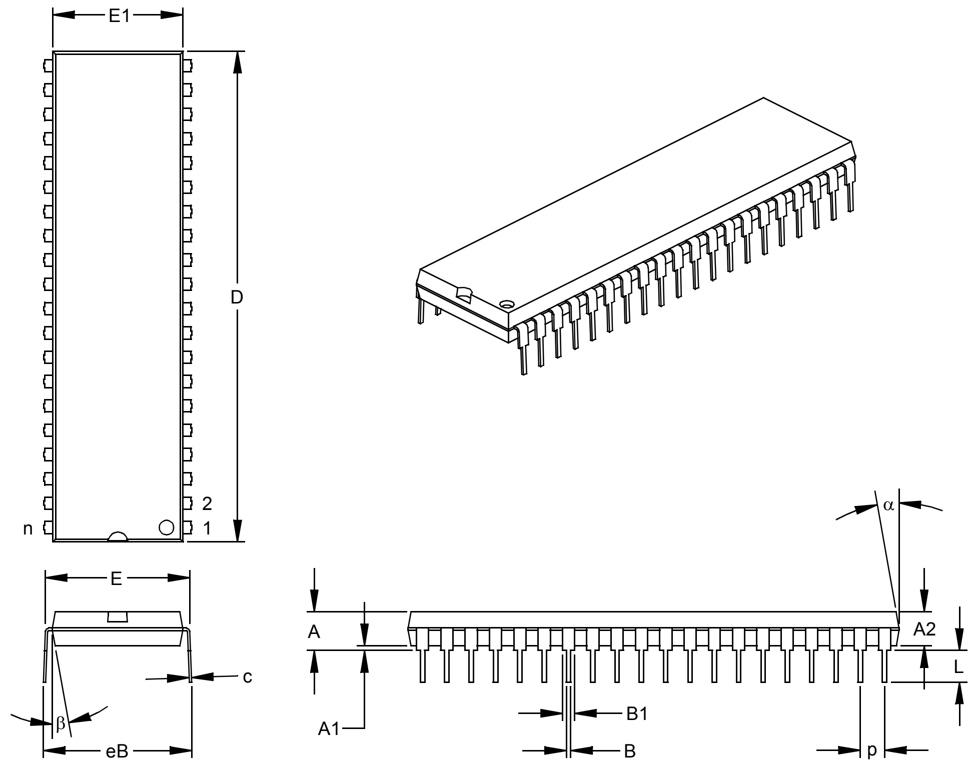


Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		28			28	
Pitch	p		.050			1.27	
Overall Height	A	.093	.099	.104	2.36	2.50	2.64
Molded Package Thickness	A2	.088	.091	.094	2.24	2.31	2.39
Standoff §	A1	.004	.008	.012	0.10	0.20	0.30
Overall Width	E	.394	.407	.420	10.01	10.34	10.67
Molded Package Width	E1	.288	.295	.299	7.32	7.49	7.59
Overall Length	D	.695	.704	.712	17.65	17.87	18.08
Chamfer Distance	h	.010	.020	.029	0.25	0.50	0.74
Foot Length	L	.016	.033	.050	0.41	0.84	1.27
Foot Angle Top	φ	0	4	8	0	4	8
Lead Thickness	c	.009	.011	.013	0.23	0.28	0.33
Lead Width	B	.014	.017	.020	0.36	0.42	0.51
Mold Draft Angle Top	α	0	12	15	0	12	15
Mold Draft Angle Bottom	β	0	12	15	0	12	15

* Controlling Parameter
§ Significant Characteristic

Notes:
Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MS-013
Drawing No. C04-052

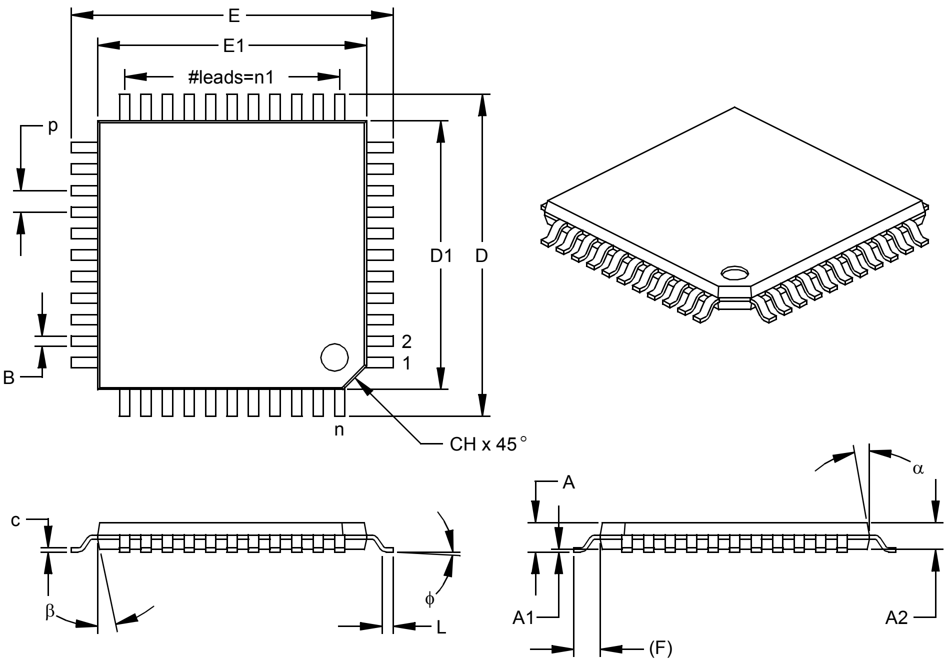
40-Lead Plastic Dual In-line (P) – 600 mil (PDIP)



Units		INCHES*			MILLIMETERS		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		40			40	
Pitch	p		.100			2.54	
Top to Seating Plane	A	.160	.175	.190	4.06	4.45	4.83
Molded Package Thickness	A2	.140	.150	.160	3.56	3.81	4.06
Base to Seating Plane	A1	.015			0.38		
Shoulder to Shoulder Width	E	.595	.600	.625	15.11	15.24	15.88
Molded Package Width	E1	.530	.545	.560	13.46	13.84	14.22
Overall Length	D	2.045	2.058	2.065	51.94	52.26	52.45
Tip to Seating Plane	L	.120	.130	.135	3.05	3.30	3.43
Lead Thickness	c	.008	.012	.015	0.20	0.29	0.38
Upper Lead Width	B1	.030	.050	.070	0.76	1.27	1.78
Lower Lead Width	B	.014	.018	.022	0.36	0.46	0.56
Overall Row Spacing §	eB	.620	.650	.680	15.75	16.51	17.27
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter
§ Significant Characteristic
Notes:
Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010” (0.254mm) per side.
JEDEC Equivalent: MO-011
Drawing No. C04-016

44-Lead Plastic Thin Quad Flatpack (PT) 10x10x1 mm Body, 1.0/0.10 mm Lead Form (TQFP)

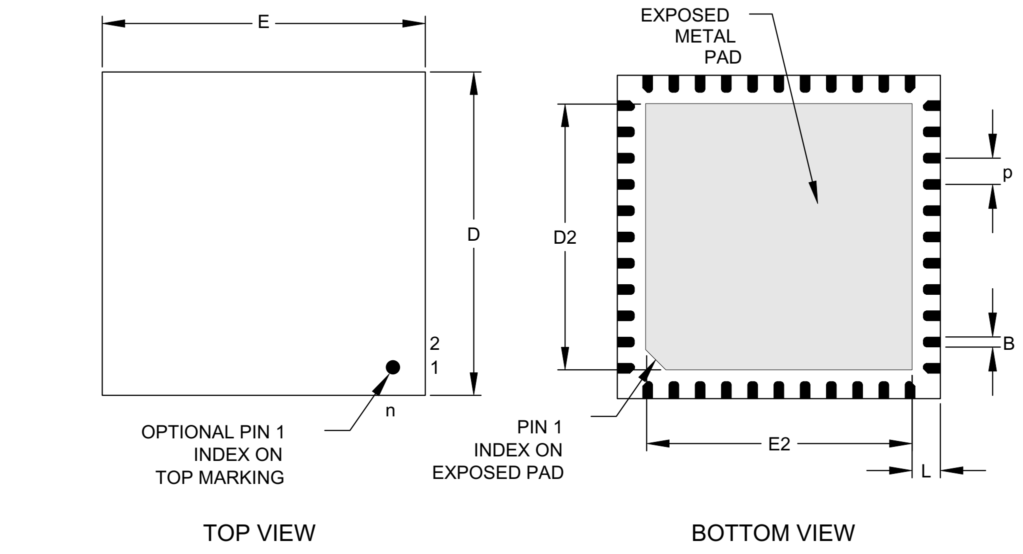


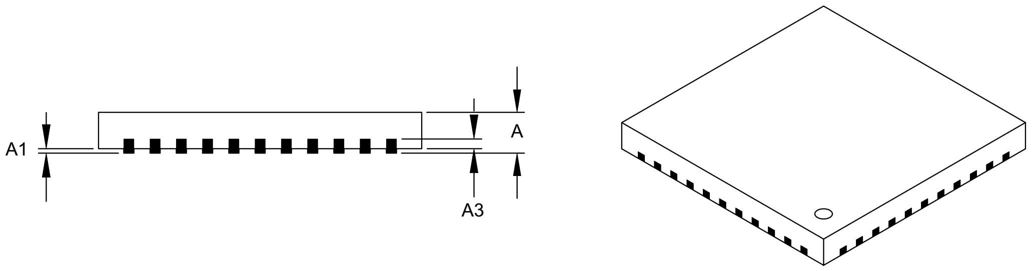
Units		INCHES			MILLIMETERS*		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		44			44	
Pitch	p		.031			0.80	
Pins per Side	n1		11			11	
Overall Height	A	.039	.043	.047	1.00	1.10	1.20
Molded Package Thickness	A2	.037	.039	.041	0.95	1.00	1.05
Standoff §	A1	.002	.004	.006	0.05	0.10	0.15
Foot Length	L	.018	.024	.030	0.45	0.60	0.75
Footprint (Reference)	(F)		.039		1.00		
Foot Angle	ϕ	0	3.5	7	0	3.5	7
Overall Width	E	.463	.472	.482	11.75	12.00	12.25
Overall Length	D	.463	.472	.482	11.75	12.00	12.25
Molded Package Width	E1	.390	.394	.398	9.90	10.00	10.10
Molded Package Length	D1	.390	.394	.398	9.90	10.00	10.10
Lead Thickness	c	.004	.006	.008	0.09	0.15	0.20
Lead Width	B	.012	.015	.017	0.30	0.38	0.44
Pin 1 Corner Chamfer	CH	.025	.035	.045	0.64	0.89	1.14
Mold Draft Angle Top	α	5	10	15	5	10	15
Mold Draft Angle Bottom	β	5	10	15	5	10	15

* Controlling Parameter
§ Significant Characteristic

Notes:
Dimensions D1 and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" (0.254mm) per side.
JEDEC Equivalent: MS-026
Drawing No. C04-076

44-Lead Plastic Quad Flat No Lead Package (ML) 8x8 mm Body (QFN)


E
D
2
1
n
OPTIONAL PIN 1 INDEX ON TOP MARKING
TOP VIEW
EXPOSED METAL PAD
D2
p
B
PIN 1 INDEX ON EXPOSED PAD
E2
L
BOTTOM VIEW



Units		INCHES			MILLIMETERS*		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Number of Pins	n		44			44	
Pitch	p		.026 BSC			0.65 BSC	
Overall Height	A	.031	.035	.039	0.80	0.90	1.00
Standoff	A1	.000	.001	.002	0	0.02	0.05
Base Thickness	A3		.010 REF			0.25 REF	
Overall Width	E		.315 BSC			8.00 BSC	
Exposed Pad Width	E2	.262	.268	.274	6.65	6.80	6.95
Overall Length	D		.315 BSC			8.00 BSC	
Exposed Pad Length	D2	.262	.268	.274	6.65	6.80	6.95
Lead Width	B	.012	.013	.013	0.30	0.33	0.35
Lead Length	L	.014	.016	.018	0.35	0.40	0.45

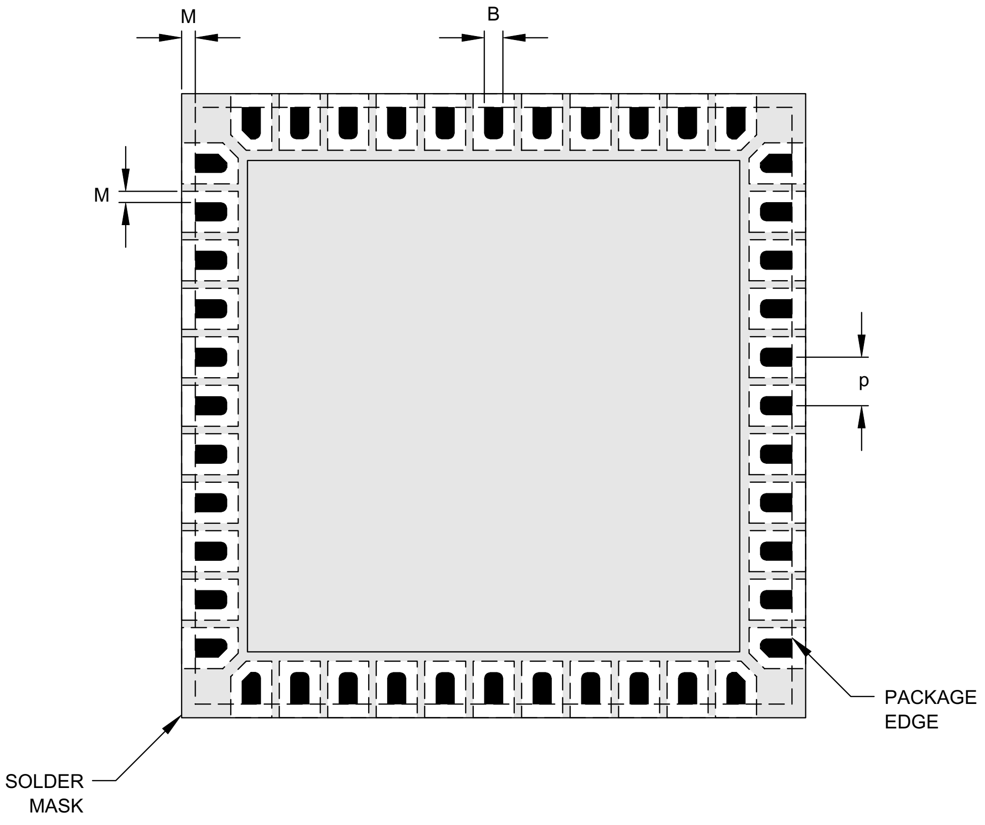
*Controlling Parameter

Notes:

Dimensions D and E1 do not include mold flash or protrusions. Mold flash or protrusions shall not exceed .010" (0.254mm) per side.
JEDEC equivalent: M0-220

Drawing No. C04-103

44-Lead Quad Flat No Lead Package (ML) 8x8 mm Body (QFN) Land Pattern and Solder Mask



Units		INCHES			MILLIMETERS*		
Dimension Limits		MIN	NOM	MAX	MIN	NOM	MAX
Pitch	p		.026 BSC			0.65 BSC	
Pad Width	B	__	__	__	__	__	__
Pad Length	L	__	__	__	__	__	__
Pad to Solder Mask	M	.005		.006	0.13		0.15

*Controlling Parameter

Drawing No. C04-2103

APPENDIX A: REVISION HISTORY

Revision A (November 2002)

Original data sheet for the PIC18FXX39 family.

APPENDIX B: DEVICE DIFFERENCES

The differences between the devices listed in this data sheet are shown in Table B-1.

TABLE B-1: DEVICE DIFFERENCES

Feature	PIC18F2439	PIC18F2539	PIC18F4439	PIC18F4539
Program Memory (Kbytes)	12	24	12	24
Data Memory (Bytes)	640	1408	640	1408
A/D Channels	5	5	8	8
Parallel Slave Port (PSP)	No	No	Yes	Yes
Package Types	28-pin DIP 28-pin SOIC	28-pin DIP 28-pin SOIC	40-pin DIP 44-pin TQFP 44-pin QFN	40-pin DIP 44-pin TQFP 44-pin QFN

APPENDIX C: CONVERSION CONSIDERATIONS

The considerations for converting applications from previous versions of PIC18 microcontrollers (i.e., PIC18FXX2 devices) are listed in Table C-1.

A specific list of resources that are unavailable to PIC18FXX2 applications in PIC18FXX39 devices is presented in Table C-2.

TABLE C-1: CONVERSION CONSIDERATIONS BETWEEN PIC18FXX2 AND PIC18FXX39 DEVICES

Characteristic	PIC18FXX2	PIC18FXX39
Pins	28/40/44	28/40/44
Available Packages	DIP, PDIP, SOIC, PLCC, QFN, TQFP	DIP, PDIP, SOIC, QFN, TQFP
Voltage Range	2.0 - 5.5V	2.0 - 5.5V
Frequency Range	DC - 40 MHz	4 - 40 MHz (20 MHz optimal)
Available Program Memory (bytes)	16K or 32K	12K or 24K
Available Data RAM (bytes)	768 or 1536	640 or 1408
Data EEPROM	256	256
Interrupt Sources	17 or 18	15 or 16
Interrupt Priority Levels	Two levels: low priority (vector at 0008h) high priority (vector at 0018h)	One level when using Motor Control: vector at 0008h
Timers (available to users)	4	3
Timer1 Oscillator option	yes	no
Oscillator Switching	yes	no
Capture/Compare/PWM	2 CCP	2 PWM only, available only through Motor Control kernel
Motor Control Kernel	no	yes
A/D	10-bit, 5 or 8 channels, 7 conversion clock selects	10-bit, 5 or 8 channels, 7 conversion clock selects
Communications	PSP, AUSART, MSSP (SPI and I^2C)	PSP, AUSART, MSSP (SPI and I^2C)
Code Protection	By 8K block with separate 512-byte boot block; protection from external reads and writes, Table Read and intra-block Table Read	By 8K block with separate 512-byte boot block; protection from external reads and writes, Table Read and intra-block Table Read; Block 3 not protected on PIC18FX539

TABLE C-2: UNAVAILABLE RESOURCES (COMPARED TO PIC18FXX2)

Resource Type	Item(s)
I/O Resources	RC1; RC2; T1OSO; T1OSI
Registers	CCP1CON; CCP2CON; CCPR1L; CCPR2L; TMR2; PR2; T2CON; OSCCON
SFR bits	CCP1IE; CCP1IF; CCP1IP; CCP21E; CCP21F; CCP2IP; T1OSCEN; T3CCP1; TMR2ON; TOUTPS<3:0>; T2CKPS<1:0>; T3CCP2; SFS; RC1; RC2; TRISC1; TRISC2; LATC1; LATC2
Interrupts and Interrupt Resources	CCP1 Capture/Compare match; CCP2 Capture/Compare match; High priority interrupts (when Motor Control is used; reserved for Timer2)
Timer Resources	Timer2 (available only through the Motor Control kernel); Timer2 as a clock source for MSSP module (SPI mode)
CCP Resources	Capture and Compare functionality; Timer1 reset on special event; Timer3 reset on special event; A/D conversion on special event; Interrupt on special event
Configuration Word bits	OSCEN; CCP2MX; CP3; WRT3; EBTR3

APPENDIX D: MIGRATION FROM HIGH-END TO ENHANCED DEVICES

A detailed discussion of the migration pathway and differences between the high-end MCU devices (i.e., PIC17CXXX) and the enhanced devices (i.e., PIC18FXXX) is provided in AN726, “PIC17CXXX to PIC18CXXX Migration”. This Application Note is available as Literature Number DS00726.

NOTES:

INDEX

ON-LINE SUPPORT

Microchip provides on-line support on the Microchip World Wide Web site.

The web site is used by Microchip as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Netscape® or Microsoft® Internet Explorer. Files are also available for FTP download from our FTP site.

Connecting to the Microchip Internet Web Site

The Microchip web site is available at the following URL:

www.microchip.com

The file transfer site is available by using an FTP service to connect to:

ftp://ftp.microchip.com

The web site and file transfer site provide a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, Articles and Sample Programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices, distributors and factory representatives. Other data available for consideration is:

- Latest Microchip Press Releases
- Technical Support Section with Frequently Asked Questions
- Design Tips
- Device Errata
- Job Postings
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip Products
- Conferences for products, Development Systems, technical information and more
- Listing of seminars and events

SYSTEMS INFORMATION AND UPGRADE HOT LINE

The Systems Information and Upgrade Line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive the most current upgrade kits.The Hot Line Numbers are:

1-800-755-2345 for U.S. and most of Canada, and

1-480-792-7302 for the rest of the world.

092002

READER RESPONSE

It is our intention to provide you with the best documentation possible to ensure successful use of your Microchip product. If you wish to provide your comments on organization, clarity, subject matter, and ways in which our documentation can better serve you, please FAX your comments to the Technical Publications Manager at (480) 792-4150.

Please list the following information, and use this outline to provide us with your comments about this document.

To: Technical Publications Manager Total Pages Sent ________

RE: Reader Response

From: Name ______________________________

Company ______________________________

Address ______________________________

City / State / ZIP / Country ______________________________

Telephone: (_______) _________ - _________ FAX: (______) _________ - _________

Application (optional):

Would you like a reply?___Y ___N

Device: PIC18FXX39 Literature Number: DS30485A

Questions:

1. What are the best features of this document?

2. How does this document meet your hardware and software development needs?

3. Do you find the organization of this document easy to follow? If not, why?

4. What additions to the document do you think would enhance the structure and subject?

5. What deletions from the document could be made without affecting the overall usefulness?

6. Is there any incorrect or misleading information (what and where)?

7. How would you improve this document?

PIC18FXX39 PRODUCT IDENTIFICATION SYSTEM

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.

PART NO.	–	X	/XX	XXX
Device		Temperature Range	Package	Pattern

Device	PIC18FXX39(1), PIC18FXX39T(2); VDD range 4.2V to 5.5V PIC18LFXX39(1), PIC18LFXX39T(2); VDD range 2.0V to 5.5V

Temperature Range	I = -40°C to +85°C (Industrial)
	E = -40°C to +125°C (Extended)

Package	ML = QFN (Quad Flatpack, No Leads)
	P = PDIP
	PT = TQFP (Plastic Thin Quad Flatpack)
	SO = SOIC
	SP = Skinny Plastic DIP

Pattern	QTP, SQTP, Code or Special Requirements (blank otherwise)

Examples:

a) PIC18LF4539 - I/P 301 = Industrial temp., PDIP package, Extended VDD limits, QTP pattern #301.

b) PIC18LF2439 - I/SO = Industrial temp., SOIC package, Extended VDD limits.

c) PIC18F4439 - E/P = Extended temp., PDIP package, normal VDD limits.

Note 1: F = Standard Voltage range
LF = Wide Voltage Range

2: T = in tape and reel - SOIC, QFN, and TQFP packages only.

Sales and Support

Data Sheets
Products supported by a preliminary Data Sheet may have an errata sheet describing minor operational differences and recommended workarounds. To determine if an errata sheet exists for a particular device, please contact one of the following:

1. Your local Microchip sales office
2. The Microchip Corporate Literature Center U.S. FAX: (480) 792-7277
3. The Microchip Worldwide Site (www.microchip.com)

Please specify which device, revision of silicon and Data Sheet (include Literature #) you are using.

New Customer Notification System
Register on our web site (www.microchip.com/cn) to receive the most current information on our products.

MICROCHIP

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200 Fax: 480-792-7277
Technical Support: 480-792-7627
Web Address: http://www.microchip.com

Rocky Mountain
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7966 Fax: 480-792-4338

Atlanta
3780 Mansell Road, Suite 130
Alpharetta, GA 30022
Tel: 770-640-0034 Fax: 770-640-0307

Boston
2 Lan Drive, Suite 120
Westford, MA 01886
Tel: 978-692-3848 Fax: 978-692-3821

Chicago
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
4570 Westgrove Drive, Suite 160
Addison, TX 75001
Tel: 972-818-7423 Fax: 972-818-2924

Detroit
Tri-Atria Office Building
32255 Northwestern Highway, Suite 190
Farmington Hills, MI 48334
Tel: 248-538-2250 Fax: 248-538-2260

Kokomo
2767 S. Albright Road
Kokomo, Indiana 46902
Tel: 765-864-8360 Fax: 765-864-8387

Los Angeles
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 949-263-1888 Fax: 949-263-1338

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
6285 Northam Drive, Suite 108
Mississauga, Ontario L4V 1X5, Canada
Tel: 905-673-0699 Fax: 905-673-6509

ASIA/PACIFIC

Australia
Microchip Technology Australia Pty Ltd
Suite 22, 41 Rawson Street
Epping 2121, NSW
Australia
Tel: 61-2-9868-6733 Fax: 61-2-9868-6755

China - Beijing
Microchip Technology Consulting (Shanghai) Co., Ltd., Beijing Liaison Office
Unit 915
Bei Hai Wan Tai Bldg.
No. 6 Chaoyangmen Beidajie
Beijing, 100027, No. China
Tel: 86-10-85282100 Fax: 86-10-85282104

China - Chengdu
Microchip Technology Consulting (Shanghai) Co., Ltd., Chengdu Liaison Office
Rm. 2401-2402, 24th Floor,
Ming Xing Financial Tower
No. 88 TIDU Street
Chengdu 610016, China
Tel: 86-28-86766200 Fax: 86-28-86766599

China - Fuzhou
Microchip Technology Consulting (Shanghai) Co., Ltd., Fuzhou Liaison Office
Unit 28F, World Trade Plaza
No. 71 Wusi Road
Fuzhou 350001, China
Tel: 86-591-7503506 Fax: 86-591-7503521

China - Hong Kong SAR
Microchip Technology Hongkong Ltd.
Unit 901-6, Tower 2, Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2401-1200 Fax: 852-2401-3431

China - Shanghai
Microchip Technology Consulting (Shanghai) Co., Ltd.
Room 701, Bldg. B
Far East International Plaza
No. 317 Xian Xia Road
Shanghai, 200051
Tel: 86-21-6275-5700 Fax: 86-21-6275-5060

China - Shenzhen
Microchip Technology Consulting (Shanghai) Co., Ltd., Shenzhen Liaison Office
Rm. 1812, 18/F, Building A, United Plaza
No. 5022 Binhe Road, Futian District
Shenzhen 518033, China
Tel: 86-755-82901380 Fax: 86-755-82966626

China - Qingdao
Rm. B503, Fullhope Plaza,
No. 12 Hong Kong Central Rd.
Qingdao 266071, China
Tel: 86-532-5027355 Fax: 86-532-5027205

India
Microchip Technology Inc.
India Liaison Office
Divyasree Chambers
1 Floor, Wing A (A3/A4)
No. 11, O'Shaugnessey Road
Bangalore, 560 025, India
Tel: 91-80-2290061 Fax: 91-80-2290062

Japan
Microchip Technology Japan K.K.
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 135-882
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Singapore
Microchip Technology Singapore Pte Ltd.
200 Middle Road
#07-02 Prime Centre
Singapore, 188980
Tel: 65-6334-8870 Fax: 65-6334-8850

Taiwan
Microchip Technology (Barbados) Inc.,
Taiwan Branch
11F-3, No. 207
Tung Hua North Road
Taipei, 105, Taiwan
Tel: 886-2-2717-7175 Fax: 886-2-2545-0139

EUROPE

Austria
Microchip Technology Austria GmbH
Durisolstrasse 2
A-4600 Wels
Austria
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark
Microchip Technology Nordic ApS
Regus Business Centre
Lautrup hoj 1-3
Ballerup DK-2750 Denmark
Tel: 45 4420 9895 Fax: 45 4420 9910

France
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
Microchip Technology GmbH
Steinheilstrasse 10
D-85737 Ismaning, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
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
Microchip Ltd.
505 Eskdale Road
Winnersh Triangle
Wokingham
Berkshire, England RG41 5TU
Tel: 44 118 921 5869 Fax: 44-118 921-5820

12/05/02