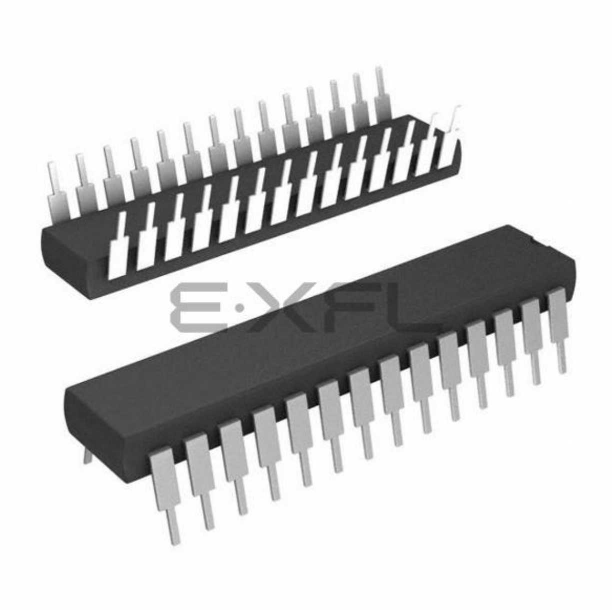


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	PIC
Core Size	8-Bit
Speed	40MHz
Connectivity	CANbus, I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, HLVD, POR, PWM, WDT
Number of I/O	25
Program Memory Size	32KB (16K x 16)
Program Memory Type	FLASH
EEPROM Size	256 x 8
RAM Size	1.5K x 8
Voltage - Supply (Vcc/Vdd)	4.2V ~ 5.5V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Through Hole
Package / Case	28-DIP (0.300", 7.62mm)
Supplier Device Package	28-SPDIP
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic18f2580-i-sp

1.0 DEVICE OVERVIEW

This document contains device specific information for the following devices:

- PIC18F2480
- PIC18F2580
- PIC18F4480
- PIC18F4580

This family of devices 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. In addition to these features, the PIC18F2480/2580/4480/4580 family introduces design enhancements that make these microcontrollers a logical choice for many high-performance, power-sensitive applications.

1.1 New Core Features

1.1.1 nanoWatt TECHNOLOGY

All of the devices in the PIC18F2480/2580/4480/4580 family incorporate a range of features that can significantly reduce power consumption during operation. Key items include:

- **Alternate Run Modes:** By clocking the controller from the Timer1 source or the internal oscillator block, power consumption during code execution can be reduced by as much as 90%.
- **Multiple Idle Modes:** The controller can also run with its CPU core disabled but the peripherals still active. In these states, power consumption can be reduced even further, to as little as 4% of normal operation requirements.
- **On-the-Fly Mode Switching:** The power-managed modes are invoked by user code during operation, allowing the user to incorporate power-saving ideas into their application's software design.
- **Lower Consumption in Key Modules:** The power requirements for both Timer1 and the Watchdog Timer have been reduced by up to 80%, with typical values of 1.1 and 2.1 μA, respectively.
- **Extended Instruction Set:** In addition to the standard 75 instructions of the PIC18 instruction set, PIC18F2480/2580/4480/4580 devices also provide an optional extension to the core CPU functionality. The added features include eight additional instructions that augment indirect and indexed addressing operations and the implementation of Indexed Literal Offset Addressing mode for many of the standard PIC18 instructions.

1.1.2 MULTIPLE OSCILLATOR OPTIONS AND FEATURES

All of the devices in the PIC18F2480/2580/4480/4580 family offer ten different oscillator options, allowing users a wide range of choices in developing application hardware. These include:

- Four Crystal modes, using crystals or ceramic resonators
- Two External Clock modes, offering the option of using two pins (oscillator input and a divide-by-4 clock output) or one pin (oscillator input, with the second pin reassigned as general I/O)
- Two External RC Oscillator modes with the same pin options as the External Clock modes
- An internal oscillator block which provides an 8 MHz clock (±2% accuracy) and an INTRC source (approximately 31 kHz, stable over temperature and V_{DD}), as well as a range of 6 user-selectable clock frequencies, between 125 kHz to 4 MHz, for a total of 8 clock frequencies. This option frees the two oscillator pins for use as additional general purpose I/O.
- A Phase Lock Loop (PLL) frequency multiplier, available to both the high-speed crystal and internal oscillator modes, which allows clock speeds of up to 40 MHz. Used with the internal oscillator, the PLL gives users a complete selection of clock speeds, from 31 kHz to 32 MHz – all without using an external crystal or clock circuit.

Besides its availability as a clock source, the internal oscillator block provides a stable reference source that gives the family additional features for robust operation:

- **Fail-Safe Clock Monitor:** This option constantly monitors the main clock source against a reference signal provided by the internal oscillator. If a clock failure occurs, the controller is switched to the internal oscillator block, allowing for continued low-speed operation or a safe application shutdown.
- **Two-Speed Start-up:** This option allows the internal oscillator to serve as the clock source from Power-on Reset, or wake-up from Sleep mode, until the primary clock source is available.

TABLE 1-3: PIC18F4480/4580 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	PDIP	QFN	TQFP			
						PORTE is a bidirectional I/O port.
RE0/$\overline{RD}$/AN5	8	25	25			
RE0				I/O	ST	Digital I/O.
$\overline{RD}$				I	TTL	Read control for Parallel Slave Port (see also $\overline{WR}$ and $\overline{CS}$ pins).
AN5				I	Analog	Analog Input 5.
RE1/$\overline{WR}$/AN6/C1OUT	9	26	26			
RE1				I/O	ST	Digital I/O.
$\overline{WR}$				I	TTL	Write control for Parallel Slave Port (see $\overline{CS}$ and $\overline{RD}$ pins).
AN6				I	Analog	Analog Input 6.
C1OUT				O	TTL	Comparator 1 output.
RE2/$\overline{CS}$/AN7/C2OUT	10	27	27			
RE2				I/O	ST	Digital I/O.
$\overline{CS}$				I	TTL	Chip select control for Parallel Slave Port (see related $\overline{RD}$ and $\overline{WR}$).
AN7				I	Analog	Analog Input 7.
C2OUT				O	TTL	Comparator 2 output.
RE3	—	—	—	—	—	See $\overline{MCLR}$/VPP/RE3 pin.
VSS	12, 31	6, 30, 31	6, 29	P	—	Ground reference for logic and I/O pins.
VDD	11, 32	7, 8, 28, 29	7, 28	P	—	Positive supply for logic and I/O pins.
NC	—	13	12, 13, 33, 34	—	—	No connect.

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output; ST = Schmitt Trigger input with CMOS levels; I = Input; O = Output; P = Power; I^2C = I^2C™/SMBus input buffer

TABLE 5-4: INITIALIZATION CONDITIONS FOR ALL REGISTERS (CONTINUED)

Register	Applicable Devices				Power-on Reset, Brown-out Reset	MCLR Resets, WDT Reset, RESET Instruction, Stack Resets	Wake-up via WDT or Interrupt
B0EIDL(6)	2480	2580	4480	4580	xxxx xxxx	uuuu uuuu	uuuu uuuu
B0EIDH(6)	2480	2580	4480	4580	xxxx xxxx	uuuu uuuu	uuuu uuuu
B0SIDL(6)	2480	2580	4480	4580	xxxx x-xx	uuuu u-uu	uuuu u-uu
B0SIDH(6)	2480	2580	4480	4580	xxxx xxxx	uuuu uuuu	uuuu uuuu
B0CON(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
TXBIE(6)	2480	2580	4480	4580	---0 00--	---u uu--	---u uu--
BIE0(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
BSEL0(6)	2480	2580	4480	4580	0000 00--	0000 00--	uuuu uu--
MSEL3(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
MSEL2(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
MSEL1(6)	2480	2580	4480	4580	0000 0101	0000 0101	uuuu uuuu
MSEL0(6)	2480	2580	4480	4580	0101 0000	0101 0000	uuuu uuuu
SDFLC(6)	2480	2580	4480	4580	---0 0000	---0 0000	-u-- uuuu
RXFCON1(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
RXFCON0(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
RXFBCON7(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
RXFBCON6(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
RXFBCON5(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
RXFBCON4(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
RXFBCON3(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
RXFBCON2(6)	2480	2580	4480	4580	0001 0001	0001 0001	uuuu uuuu
RXFBCON1(6)	2480	2580	4480	4580	0001 0001	0001 0001	uuuu uuuu
RXFBCON0(6)	2480	2580	4480	4580	0000 0000	0000 0000	uuuu uuuu
RXF15EIDL(6)	2480	2580	4480	4580	xxxx xxxx	uuuu uuuu	uuuu uuuu
RXF15EIDH(6)	2480	2580	4480	4580	xxxx xxxx	uuuu uuuu	uuuu uuuu
RXF15SIDL(6)	2480	2580	4480	4580	xxx- x-xx	uuu- u-uu	uuu- u-uu
RXF15SIDH(6)	2480	2580	4480	4580	xxxx xxxx	uuuu uuuu	uuuu uuuu
RXF14EIDL(6)	2480	2580	4480	4580	xxxx xxxx	uuuu uuuu	uuuu uuuu
RXF14EIDH(6)	2480	2580	4480	4580	xxxx xxxx	uuuu uuuu	uuuu uuuu
RXF14SIDL(6)	2480	2580	4480	4580	xxx- x-xx	uuu- u-uu	uuu- u-uu
RXF14SIDH(6)	2480	2580	4480	4580	xxxx xxxx	uuuu uuuu	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.

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

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

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

4: See Table 5-3 for Reset value for specific condition.

5: Bits 6 and 7 of PORTA, LATA and TRISA are enabled, depending on the oscillator mode selected. When not enabled as PORTA pins, they are disabled and read '0'.

6: This register reads all '0's until ECAN™ technology is set up in Mode 1 or Mode 2.

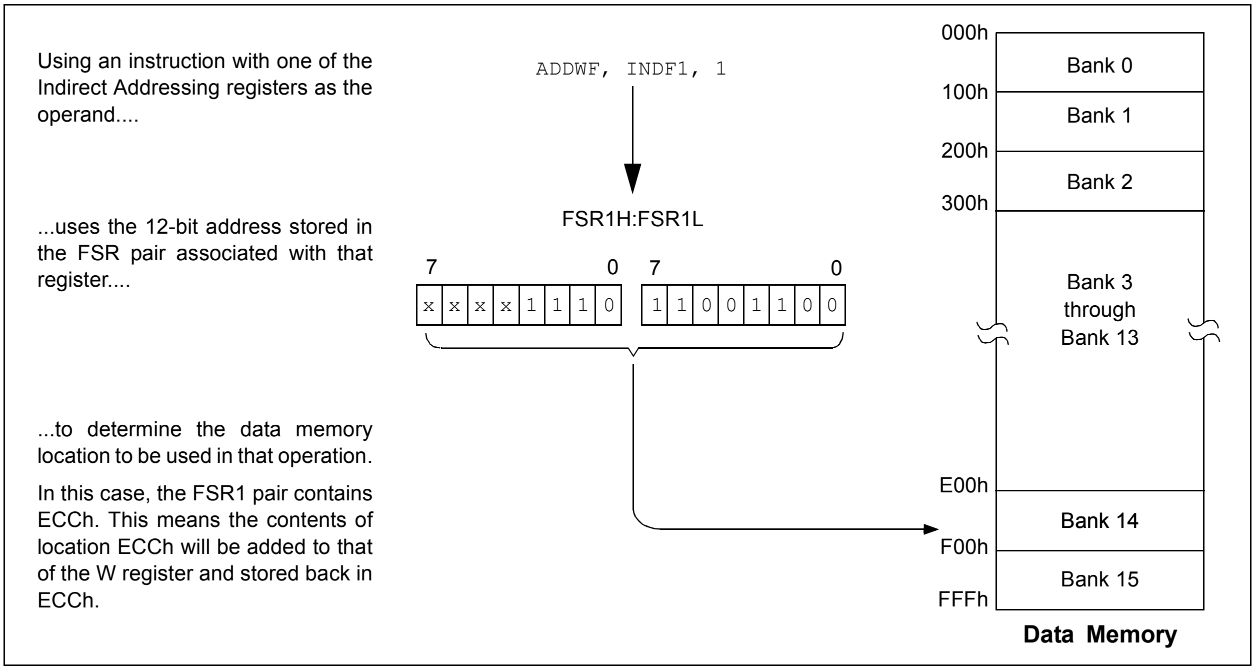
6.4.3.1 FSR Registers and the INDF Operand

At the core of Indirect Addressing are three sets of registers: FSR0, FSR1 and FSR2. Each represents a pair of 8-bit registers, FSRnH and FSRnL. The four upper bits of the FSRnH register are not used, so each FSR pair holds a 12-bit value. This represents a value that can address the entire range of the data memory in a linear fashion. The FSR register pairs, then, serve as pointers to data memory locations.

Indirect Addressing is accomplished with a set of Indirect File Operands, INDF0 through INDF2. These can be thought of as “virtual” registers: they are mapped in the SFR space, but are not physically implemented. Reading or writing to a particular INDF register actually accesses its corresponding FSR register pair. A read from INDF1, for example, reads the data at the address indicated by FSR1H:FSR1L. Instructions that use the INDF registers as operands actually use the contents of their corresponding FSR as a pointer to the instruction’s target. The INDF operand is just a convenient way of using the pointer.

Because Indirect Addressing uses a full 12-bit address, data RAM banking is not necessary. Thus, the current contents of the BSR and the Access RAM bit have no effect on determining the target address.

FIGURE 6-8: INDIRECT ADDRESSING



REGISTER 10-3: INTCON3: INTERRUPT CONTROL REGISTER 3

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

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

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

bit 6 **INT1IP:** 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: Interrupt flag bits are set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global interrupt enable bit. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt. This feature allows for software polling.

TABLE 11-5: PORTC I/O SUMMARY

Pin Name	Function	I/O	TRIS	Buffer	Description
RC0/T1OSO/ T13CKI	RC0	OUT	0	DIG	LATC<0> data output.
		IN	1	ST	PORTC<0> data input.
	T1OSO	OUT	x	ANA	Timer1 oscillator output – overrides the TRIS<0> control when enabled.
	T13CKI	IN	1	ST	Timer1/Timer3 clock input.
RC1/T1OSI	RC1	OUT	0	DIG	LATC<1> data output.
		IN	1	ST	PORTC<1> data input.
	T1OSI	IN	x	ANA	Timer1 oscillator input – overrides the TRIS<1> control when enabled.
RC2/CCP1	RC2	OUT	0	DIG	LATC<2> data output.
		IN	1	ST	PORTC<2> data input.
	CCP1	OUT	0	DIG	CCP1 compare output.
		IN	1	ST	CCP1 capture input.
RC3/SCK/SCL	RC3	OUT	0	DIG	LATC<3> data output.
		IN	1	ST	PORTC<3> data input.
	SCK	OUT	0	DIG	SPI clock output (MSSP module) – must have TRIS set to '1' to allow MSSP module to control the bidirectional communication.
		IN	1	ST	SPI clock input (MSSP module).
	SCL	OUT	0	DIG	I^{2}C™/SM bus clock output (MSSP module) – must have TRIS set to '1' to allow MSSP module to control the bidirectional communication.
		IN	1	I^{2}C™/SMB	I^{2}C/SM bus clock input.
RC4/SDI/SDA	RC4	OUT	0	DIG	LATC<4> data output.
		IN	1	ST	PORTC<4> data input.
	SDI	IN	1	ST	SPI data input (MSSP module).
	SDA	OUT	1	DIG	I^{2}C/SM bus data output (MSSP module) – must have TRIS set to '1' to allow MSSP module to control the bidirectional communication.
		IN	1	I^{2}C/SMB	I^{2}C/SM bus data input (MSSP module) – must have TRIS set to '1' to allow MSSP module to control the bidirectional communication.
RC5/SDO	RC5	OUT	0	DIG	LATC<5> data output.
		IN	1	ST	PORTC<5> data input.
	SDO	OUT	0	DIG	SPI data output (MSSP module).
RC6/TX/CK	RC6	OUT	0	DIG	LATC<6> data output.
		IN	1	ST	PORTC<6> data input.
	TX	OUT	0	DIG	EUSART data output.
	CK	OUT	1	DIG	EUSART synchronous clock output – must have TRIS set to '1' to enable EUSART to control the bidirectional communication.
		IN	1	ST	EUSART synchronous clock input.
RC7/RX/DT	RC7	OUT	0	DIG	LATC<7> data output.
		IN	1	ST	PORTC<7> data input.
	RX	IN	1	ST	EUSART asynchronous data input.
	DT	OUT	1	DIG	EUSART synchronous data output – must have TRIS set to '1' to enable EUSART to control the bidirectional communication.
		IN	1	ST	EUSART synchronous data input.

Legend: OUT = Output, IN = Input, ANA = Analog Signal, DIG = Digital Output, ST = Schmitt Buffer Input, TTL = TTL Buffer Input

TABLE 11-6: SUMMARY OF REGISTERS ASSOCIATED WITH PORTC

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset Values on Page:
PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	58
LATC	LATC Output Latch Register								58
TRISC	PORTC Data Direction Register								58

REGISTER 11-1: TRISE REGISTER (PIC18F4X80 DEVICES ONLY)

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

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

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

13.0 TIMER1 MODULE

The Timer1 timer/counter module incorporates these features:

- Software-selectable operation as a 16-bit timer or counter
- Readable and writable 8-bit registers (TMR1H and TMR1L)
- Selectable clock source (internal or external) with device clock or Timer1 oscillator internal options
- Interrupt-on-overflow
- Module Reset on CCP Special Event Trigger
- Device clock status flag (T1RUN)

A simplified block diagram of the Timer1 module is shown in Figure 13-1. A block diagram of the module's operation in Read/Write mode is shown in Figure 13-2.

The module incorporates its own low-power oscillator to provide an additional clocking option. The Timer1 oscillator can also be used as a low-power clock source for the microcontroller in power-managed operation.

Timer1 can also be used to provide Real-Time Clock (RTC) functionality to applications with only a minimal addition of external components and code overhead.

Timer1 is controlled through the T1CON Control register (Register 13-1). It also contains the Timer1 Oscillator Enable bit (T1OSCEN). Timer1 can be enabled or disabled by setting or clearing control bit, TMR1ON (T1CON<0>).

REGISTER 13-1: T1CON: TIMER1 CONTROL REGISTER

R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
RD16	T1RUN	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 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

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 **T1RUN:** Timer1 System Clock Status bit
1 = Device clock is derived from Timer1 oscillator
0 = Device clock is derived from another source

bit 5-4 **T1CKPS<1:0>:** Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Enable bit
1 = Timer1 oscillator is enabled
0 = Timer1 oscillator is shut off
The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **$\overline{\text{T1SYNC}}$:** Timer1 External Clock Input Synchronization Select bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T13CKI (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

NOTES:

24.0 ECAN MODULE

PIC18F2480/2580/4480/4580 devices contain an Enhanced Controller Area Network (ECAN) module. The ECAN module is fully backward compatible with the CAN module available in PIC18CXX8 and PIC18FXX8 devices.

The Controller Area Network (CAN) module is a serial interface which is useful for communicating with other peripherals or microcontroller devices. This interface, or protocol, was designed to allow communications within noisy environments.

The ECAN module is a communication controller, implementing the CAN 2.0A or B protocol as defined in the BOSCH specification. The module will support CAN 1.2, CAN 2.0A, CAN 2.0B Passive and CAN 2.0B Active versions of the protocol. The module implementation is a full CAN system; however, the CAN specification is not covered within this data sheet. Refer to the BOSCH CAN specification for further details.

The module features are as follows:

- Implementation of the CAN protocol, CAN 1.2, CAN 2.0A and CAN 2.0B
- DeviceNet™ data bytes filter support
- Standard and extended data frames
- 0-8 bytes data length
- Programmable bit rate up to 1 Mbit/sec
- Fully backward compatible with the PIC18XXX8 CAN module
- Three modes of operation:
  - Mode 0 – Legacy mode
  - Mode 1 – Enhanced Legacy mode with DeviceNet support
  - Mode 2 – FIFO mode with DeviceNet support
- Support for remote frames with automated handling
- Double-buffered receiver with two prioritized received message storage buffers
- Six buffers programmable as RX and TX message buffers
- 16 full (standard/extended identifier) acceptance filters that can be linked to one of four masks
- Two full acceptance filter masks that can be assigned to any filter
- One full acceptance filter that can be used as either an acceptance filter or acceptance filter mask
- Three dedicated transmit buffers with application specified prioritization and abort capability
- Programmable wake-up functionality with integrated low-pass filter
- Programmable Loopback mode supports self-test operation
- Signaling via interrupt capabilities for all CAN receiver and transmitter error states
- Programmable clock source
- Programmable link to timer module for time-stamping and network synchronization
- Low-power Sleep mode

24.1 Module Overview

The CAN bus module consists of a protocol engine and message buffering and control. The CAN protocol engine automatically handles all functions for receiving and transmitting messages on the CAN bus. Messages are transmitted by first loading the appropriate data registers. Status and errors can be checked by reading the appropriate registers. Any message detected on the CAN bus is checked for errors and then matched against filters to see if it should be received and stored in one of the two receive registers.

The CAN module supports the following frame types:

- Standard Data Frame
- Extended Data Frame
- Remote Frame
- Error Frame
- Overload Frame Reception

The CAN module uses the RB2/CANTX and RB3/CANRX pins to interface with the CAN bus. In normal mode, the CAN module automatically overrides TRISB<2>. The user must ensure that TRISB<3> is set.

24.1.1 MODULE FUNCTIONALITY

The CAN bus module consists of a protocol engine, message buffering and control (see Figure 24-1). The protocol engine can best be understood by defining the types of data frames to be transmitted and received by the module.

The following sequence illustrates the necessary initialization steps before the ECAN module can be used to transmit or receive a message. Steps can be added or removed depending on the requirements of the application.

1. Initial LAT and TRIS bits for RX and TX CAN.
2. Ensure that the ECAN module is in Configuration mode.
3. Select ECAN Operational mode.
4. Set up the Baud Rate registers.
5. Set up the Filter and Mask registers.
6. Set the ECAN module to normal mode or any other mode required by the application logic.

TABLE 24-1: CAN CONTROLLER REGISTER MAP (CONTINUED)

Address(1)	Name	Address	Name	Address	Name	Address	Name
EFFh	—(4)	EDFh	—(4)	EBFh	—(4)	E9Fh	—(4)
EFEh	—(4)	EDEh	—(4)	EBEh	—(4)	E9Eh	—(4)
EFDh	—(4)	EDDh	—(4)	EBDh	—(4)	E9Dh	—(4)
EFCh	—(4)	EDCh	—(4)	EBCh	—(4)	E9Ch	—(4)
EFBh	—(4)	EDBh	—(4)	EBBh	—(4)	E9Bh	—(4)
EFAh	—(4)	EDAh	—(4)	EBAh	—(4)	E9Ah	—(4)
EF9h	—(4)	ED9h	—(4)	EB9h	—(4)	E99h	—(4)
EF8h	—(4)	ED8h	—(4)	EB8h	—(4)	E98h	—(4)
EF7h	—(4)	ED7h	—(4)	EB7h	—(4)	E97h	—(4)
EF6h	—(4)	ED6h	—(4)	EB6h	—(4)	E96h	—(4)
EF5h	—(4)	ED5h	—(4)	EB5h	—(4)	E95h	—(4)
EF4h	—(4)	ED4h	—(4)	EB4h	—(4)	E94h	—(4)
EF3h	—(4)	ED3h	—(4)	EB3h	—(4)	E93h	—(4)
EF2h	—(4)	ED2h	—(4)	EB2h	—(4)	E92h	—(4)
EF1h	—(4)	ED1h	—(4)	EB1h	—(4)	E91h	—(4)
EF0h	—(4)	ED0h	—(4)	EB0h	—(4)	E90h	—(4)
EEFh	—(4)	ECFh	—(4)	EAFh	—(4)	E8Fh	—(4)
EEEh	—(4)	ECEh	—(4)	EAEh	—(4)	E8Eh	—(4)
EEDh	—(4)	ECDh	—(4)	EADh	—(4)	E8Dh	—(4)
EECh	—(4)	ECCh	—(4)	EACh	—(4)	E8Ch	—(4)
EEBh	—(4)	ECBh	—(4)	EABh	—(4)	E8Bh	—(4)
EEAh	—(4)	ECAh	—(4)	EAAh	—(4)	E8Ah	—(4)
EE9h	—(4)	EC9h	—(4)	EA9h	—(4)	E89h	—(4)
EE8h	—(4)	EC8h	—(4)	EA8h	—(4)	E88h	—(4)
EE7h	—(4)	EC7h	—(4)	EA7h	—(4)	E87h	—(4)
EE6h	—(4)	EC6h	—(4)	EA6h	—(4)	E86h	—(4)
EE5h	—(4)	EC5h	—(4)	EA5h	—(4)	E85h	—(4)
EE4h	—(4)	EC4h	—(4)	EA4h	—(4)	E84h	—(4)
EE3h	—(4)	EC3h	—(4)	EA3h	—(4)	E83h	—(4)
EE2h	—(4)	EC2h	—(4)	EA2h	—(4)	E82h	—(4)
EE1h	—(4)	EC1h	—(4)	EA1h	—(4)	E81h	—(4)
EE0h	—(4)	EC0h	—(4)	EA0h	—(4)	E80h	—(4)

Note 1: Shaded registers are available in Access Bank low area, while the rest are available in Bank 15.
2: CANSTAT register is repeated in these locations to simplify application firmware. Unique names are given for each instance of the controller register due to the Microchip header file requirement.
3: These registers are not CAN registers.
4: Unimplemented registers are read as '0'.

CLRF — Clear f

Syntax:	CLRF f {,a}
Operands:	$0 \leq f \leq 255$ $a \in [0,1]$
Operation:	$000h \rightarrow f$, $1 \rightarrow Z$
Status Affected:	Z
Encoding:	`0110` `101a` `ffff` `ffff`
Description:	Clears the contents of the specified register. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank. If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \leq 95$ (5Fh). See **Section 26.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.
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 = 5Ah
After Instruction
FLAG_REG = 00h

CLRWDT — Clear Watchdog Timer

Syntax:	CLRWDT
Operands:	None
Operation:	$000h \rightarrow$ WDT, $000h \rightarrow$ WDT postscaler, $1 \rightarrow \overline{TO}$, $1 \rightarrow \overline{PD}$
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 = 00h
WDT Postscaler = 0
$\overline{TO}$ = 1
$\overline{PD}$ = 1

INCFSZ — Increment f, Skip if 0

Syntax:	INCFSZ f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(f) + 1 $\rightarrow$ 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'. 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 is selected. If 'a' is '1', the BSR is used to select the GPR bank. If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 26.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.
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:	INFSNZ f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(f) + 1 $\rightarrow$ dest, skip if result $\neq$ 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'. 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 is selected. If 'a' is '1', the BSR is used to select the GPR bank. If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 26.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.
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)

POP — Pop Top of Return Stack

Syntax: POP

Operands: None

Operation: (TOS) → 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: PUSH

Operands: None

Operation: (PC + 2) → 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 implementing a software stack by modifying TOS and then pushing 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 = 345Ah
- PC = 0124h

After Instruction
- PC = 0126h
- TOS = 0126h
- Stack (1 level down) = 345Ah

SUBWFB — Subtract W from f with Borrow

Syntax:	SUBWFB f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
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'. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank. If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 26.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.
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  =  19h    (0001 1001)
W    =  0Dh    (0000 1101)
C    =  1
```

After Instruction

```
REG  =  0Ch    (0000 1011)
W    =  0Dh    (0000 1101)
C    =  1
Z    =  0
N    =  0      ; result is positive
```

Example 2: `SUBWFB REG, 0, 0`

Before Instruction

```
REG  =  1Bh    (0001 1011)
W    =  1Ah    (0001 1010)
C    =  0
```

After Instruction

```
REG  =  1Bh    (0001 1011)
W    =  00h
C    =  1
Z    =  1      ; result is zero
N    =  0
```

Example 3: `SUBWFB REG, 1, 0`

Before Instruction

```
REG  =  03h    (0000 0011)
W    =  0Eh    (0000 1101)
C    =  1
```

After Instruction

```
REG  =  F5h    (1111 0100)
               ; [2's comp]
W    =  0Eh    (0000 1101)
C    =  0
Z    =  0
N    =  1      ; result is negative
```

SWAPF — Swap f

Syntax:	SWAPF f {,d {,a}}
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	(f<3:0>) → dest<7:4>, (f<7:4>) → 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'. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank. If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever $f \le 95$ (5Fh). See **Section 26.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.
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  =  53h
```

After Instruction

```
REG  =  35h
```

26.2 Extended Instruction Set

In addition to the standard 75 instructions of the PIC18 instruction set, PIC18F2480/2580/4480/4580 devices also provide an optional extension to the core CPU functionality. The added features include eight additional instructions that augment indirect and indexed addressing operations and the implementation of Indexed Literal Offset Addressing mode for many of the standard PIC18 instructions.

The additional features are disabled by default. To enable them, users must set the XINST Configuration bit.

The instructions in the extended set can all be classified as literal operations, which either manipulate the File Select Registers or use them for indexed addressing. Two of the instructions, `ADDFSR` and `SUBFSR`, each have an additional special instantiation for using FSR2. These versions (`ADDULNK` and `SUBULNK`) allow for automatic return after execution.

The extended instructions are specifically implemented to optimize re-entrant program code (that is, code that is recursive or that uses a software stack) written in high-level languages, particularly C. Among other things, they allow users working in high-level languages to perform certain operations on data structures more efficiently. These include:

- dynamic allocation and de-allocation of software stack space when entering and leaving subroutines
- function pointer invocation
- software Stack Pointer manipulation
- manipulation of variables located in a software stack

A summary of the instructions in the extended instruction set is provided in Table 26-3. Detailed descriptions are provided in **Section 26.2.2 "Extended Instruction Set"**. The opcode field descriptions in Table 26-1 apply to both the standard and extended PIC18 instruction sets.

Note:	The instruction set extension and the Indexed Literal Offset Addressing mode were designed for optimizing applications written in C; the user may likely never use these instructions directly in assembler. The syntax for these commands is provided as a reference for users who may be reviewing code that has been generated by a compiler.

26.2.1 EXTENDED INSTRUCTION SYNTAX

Most of the extended instructions use indexed arguments, using one of the File Select Registers and some offset to specify a source or destination register. When an argument for an instruction serves as part of indexed addressing, it is enclosed in square brackets ("[ ]"). This is done to indicate that the argument is used as an index or offset. MPASM™ Assembler will flag an error if it determines that an index or offset value is not bracketed.

When the extended instruction set is enabled, brackets are also used to indicate index arguments in byte-oriented and bit-oriented instructions. This is in addition to other changes in their syntax. For more details, see **Section 26.2.3.1 "Extended Instruction Syntax with Standard PIC18 Commands"**.

Note:	In the past, square brackets have been used to denote optional arguments in the PIC18 and earlier instruction sets. In this text and going forward, optional arguments are denoted by braces ("{ }").

TABLE 26-3: EXTENSIONS TO THE PIC18 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected
ADDFSR	f, k	Add Literal to FSR	1	1110	1000	ffkk	kkkk	None
ADDULNK	k	Add Literal to FSR2 and Return	2	1110	1000	11kk	kkkk	None
CALLW		Call Subroutine using WREG	2	0000	0000	0001	0100	None
MOVSF	z_s, f_d	Move z_s (source) to 1st word	2	1110	1011	0zzz	zzzz	None
		f_d (destination) 2nd word		1111	ffff	ffff	ffff	
MOVSS	z_s, z_d	Move z_s (source) to 1st word	2	1110	1011	1zzz	zzzz	None
		z_d (destination) 2nd word		1111	xxxx	xzzz	zzzz	
PUSHL	k	Store Literal at FSR2, Decrement FSR2	1	1110	1010	kkkk	kkkk	None
SUBFSR	f, k	Subtract Literal from FSR	1	1110	1001	ffkk	kkkk	None
SUBULNK	k	Subtract Literal from FSR2 and Return	2	1110	1001	11kk	kkkk	None

26.2.3 BYTE-ORIENTED AND BIT-ORIENTED INSTRUCTIONS IN INDEXED LITERAL OFFSET MODE

Note:	Enabling the PIC18 instruction set extension may cause legacy applications to behave erratically or fail entirely.

In addition to eight new commands in the extended set, enabling the extended instruction set also enables Indexed Literal Offset Addressing mode (**Section 6.6.1 "Indexed Addressing with Literal Offset"**). This has a significant impact on the way that many commands of the standard PIC18 instruction set are interpreted.

When the extended set is disabled, addresses embedded in opcodes are treated as literal memory locations: either as a location in the Access Bank (a = `0`), or in a GPR bank designated by the BSR (a = `1`). When the extended instruction set is enabled and a = `0`, however, a file register argument of 5Fh or less is interpreted as an offset from the pointer value in FSR2 and not as a literal address. For practical purposes, this means that all instructions that use the Access RAM bit as an argument – that is, all byte-oriented and bit-oriented instructions, or almost half of the core PIC18 instructions – may behave differently when the extended instruction set is enabled.

When the content of FSR2 is 00h, the boundaries of the Access RAM are essentially remapped to their original values. This may be useful in creating backward compatible code. If this technique is used, it may be necessary to save the value of FSR2 and restore it when moving back and forth between 'C' and assembly routines in order to preserve the Stack Pointer. Users must also keep in mind the syntax requirements of the extended instruction set (see **Section 26.2.3.1 "Extended Instruction Syntax with Standard PIC18 Commands"**).

Although the Indexed Literal Offset Addressing mode can be very useful for dynamic stack and pointer manipulation, it can also be very annoying if a simple arithmetic operation is carried out on the wrong register. Users who are accustomed to the PIC18 programming must keep in mind that, when the extended instruction set is enabled, register addresses of 5Fh or less are used for Indexed Literal Offset Addressing.

Representative examples of typical byte-oriented and bit-oriented instructions in the Indexed Literal Offset Addressing mode are provided on the following page to show how execution is affected. The operand conditions shown in the examples are applicable to all instructions of these types.

26.2.3.1 Extended Instruction Syntax with Standard PIC18 Commands

When the extended instruction set is enabled, the file register argument, 'f', in the standard byte-oriented and bit-oriented commands is replaced with the literal offset value, 'k'. As already noted, this occurs only when 'f' is less than or equal to 5Fh. When an offset value is used, it must be indicated by square brackets ("[ ]"). As with the extended instructions, the use of brackets indicates to the compiler that the value is to be interpreted as an index or an offset. Omitting the brackets, or using a value greater than 5Fh within brackets, will generate an error in the MPASM™ Assembler.

If the index argument is properly bracketed for Indexed Literal Offset Addressing, the Access RAM argument is never specified; it will automatically be assumed to be '`0`'. This is in contrast to standard operation (extended instruction set disabled) when 'a' is set on the basis of the target address. Declaring the Access RAM bit in this mode will also generate an error in the MPASM Assembler.

The destination argument, 'd', functions as before.

In the latest versions of the MPASM assembler, language support for the extended instruction set must be explicitly invoked. This is done with either the command line option, `/y`, or the PE directive in the source listing.

26.2.4 CONSIDERATIONS WHEN ENABLING THE EXTENDED INSTRUCTION SET

It is important to note that the extensions to the instruction set may not be beneficial to all users. In particular, users who are not writing code that uses a software stack may not benefit from using the extensions to the instruction set.

Additionally, the Indexed Literal Offset Addressing mode may create issues with legacy applications written to the PIC18 assembler. This is because instructions in the legacy code may attempt to address registers in the Access Bank below 5Fh. Since these addresses are interpreted as literal offsets to FSR2 when the instruction set extension is enabled, the application may read or write to the wrong data addresses.

When porting an application to the PIC18F2480/2580/4480/4580, it is very important to consider the type of code. A large, re-entrant application that is written in 'C' and would benefit from efficient compilation will do well when using the instruction set extensions. Legacy applications that heavily use the Access Bank will most likely not benefit from using the extended instruction set.

FIGURE 28-3: HIGH/LOW-VOLTAGE DETECT CHARACTERISTICS

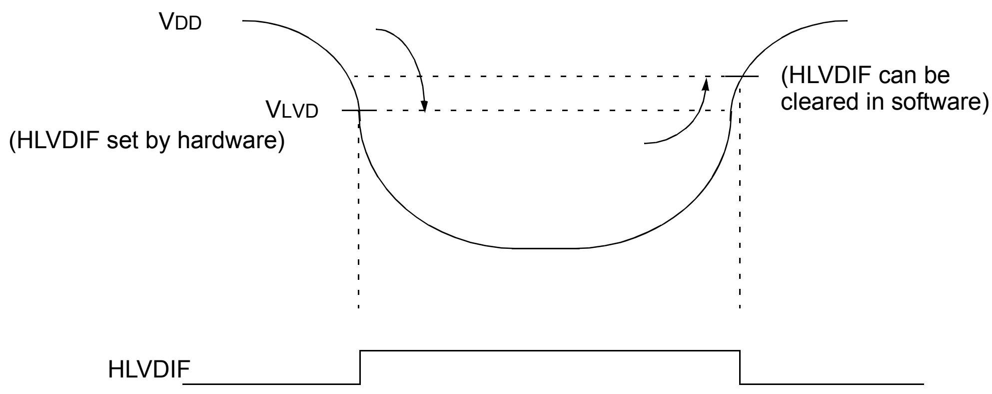
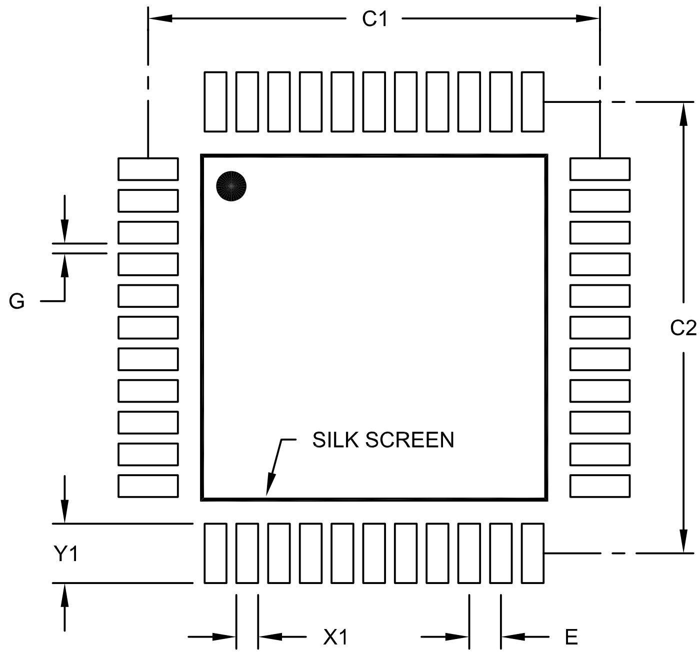


TABLE 28-4: HIGH/LOW-VOLTAGE DETECT CHARACTERISTICS

				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		HLVD Voltage on VDD Transition High-to-Low	LVV = 0000	2.12	2.17	2.22	V	
			LVV = 0001	2.18	2.23	2.28	V	
			LVV = 0010	2.31	2.36	2.42	V	
			LVV = 0011	2.38	2.44	2.49	V	
			LVV = 0100	2.54	2.60	2.66	V	
			LVV = 0101	2.72	2.79	2.85	V	
			LVV = 0110	2.82	2.89	2.95	V	
			LVV = 0111	3.05	3.12	3.19	V	
			LVV = 1000	3.31	3.39	3.47	V	
			LVV = 1001	3.46	3.55	3.63	V	
			LVV = 1010	3.63	3.71	3.80	V	
			LVV = 1011	3.81	3.90	3.99	V	
			LVV = 1100	4.01	4.11	4.20	V	
			LVV = 1101	4.23	4.33	4.43	V	
			LVV = 1110	4.48	4.59	4.69	V	
			LVV = 1111	1.14	1.2	1.26	V	

44-Lead Plastic Thin Quad Flatpack (PT) – 10x10x1 mm Body, 2.00 mm [TQFP]

Note:	For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



RECOMMENDED LAND PATTERN

Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Contact Pitch	E	0.80 BSC		
Contact Pad Spacing	C1		11.40	
Contact Pad Spacing	C2		11.40	
Contact Pad Width (X44)	X1			0.55
Contact Pad Length (X44)	Y1			1.50
Distance Between Pads	G	0.25		

Notes:

1. Dimensioning and tolerancing per ASME Y14.5M

   BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing No. C04-2076A

NOTES: