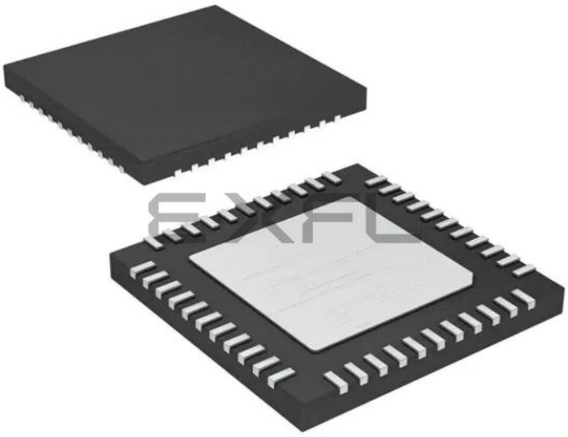


What is "Embedded - Microcontrollers"?

"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	16-Bit
Speed	32MHz
Connectivity	I²C, IrDA, LINbus, PMP, SPI, UART/USART, USB OTG
Peripherals	Brown-out Detect/Reset, DMA, LVD, POR, PWM, WDT
Number of I/O	34
Program Memory Size	128KB (43K x 24)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	8K x 8
Voltage - Supply (Vcc/Vdd)	2V ~ 3.6V
Data Converters	A/D 12x10b/12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 125°C (TA)
Mounting Type	Surface Mount
Package / Case	44-VQFN Exposed Pad
Supplier Device Package	44-QFN (8x8)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/pic24fj128gb204-e-ml

TABLE 1-1: DEVICE FEATURES FOR THE PIC24FJ128GB204 FAMILY: 44-PIN DEVICES

Features	PIC24FJ64GB204	PIC24FJ128GB204
Operating Frequency	DC – 32 MHz	
Program Memory (bytes)	64K	128K
Program Memory (instructions)	22,016	44,032
Data Memory (bytes)	8K	
Interrupt Sources (soft vectors/ NMI traps)	72 (68/4)	
I/O Ports	Ports A, B, C	
Total I/O Pins	34	
Remappable Pins	24 (23 I/Os, 1 Input only)	
Timers:		
Total Number (16-bit)	5(1)	
32-Bit (from paired 16-bit timers)	2	
Input Capture w/Timer Channels	6(1)	
Output Compare/PWM Channels	6(1)	
Input Change Notification Interrupts	34	
Serial Communications:		
UART	4(1)	
SPI (3-wire/4-wire)	3(1)	
I^2C™	2	
Digital Signal Modulator (DSM)	Yes	
Parallel Communications (EPMP/PSP)	Yes	
JTAG Boundary Scan	Yes	
12-Bit SAR Analog-to-Digital (A/D) Converter (input channels)	12	
Analog Comparators	3	
CTMU Interface	12 Channels	
Resets (and Delays)	Core POR, V_{DD} POR, V_{BAT} POR, BOR, `RESET` Instruction, $\overline{\text{MCLR}}$, WDT; Illegal Opcode, `REPEAT` Instruction, Hardware Traps, Configuration Word Mismatch (OST, PLL Lock)	
Instruction Set	76 Base Instructions, Multiple Addressing Mode Variations	
Packages	44-Pin TQFP and QFN	
Cryptographic Engine	Supports AES with 128, 192 and 256-Bit Key, DES and TDES, True Random and Pseudorandom Number Generator, On-Chip OTP Storage	
USB	USB Full-Speed and Low-Speed Compatible, On-The-Go (OTG) USB	
RTCC	Yes	

Note 1: Peripherals are accessible through remappable pins.

3.3 Arithmetic Logic Unit (ALU)

The PIC24F ALU is 16 bits wide and is capable of addition, subtraction, bit shifts and logic operations. Unless otherwise mentioned, arithmetic operations are 2's complement in nature. Depending on the operation, the ALU may affect the values of the Carry (C), Zero (Z), Negative (N), Overflow (OV) and Digit Carry (DC) Status bits in the SR register. The C and DC Status bits operate as $\overline{\text{Borrow}}$ and $\overline{\text{Digit Borrow}}$ bits, respectively, for subtraction operations.

The ALU can perform 8-bit or 16-bit operations, depending on the mode of the instruction that is used. Data for the ALU operation can come from the W register array, or data memory, depending on the addressing mode of the instruction. Likewise, output data from the ALU can be written to the W register array or a data memory location.

The PIC24F CPU incorporates hardware support for both multiplication and division. This includes a dedicated hardware multiplier and support hardware for 16-bit divisor division.

3.3.1 MULTIPLIER

The ALU contains a high-speed, 17-bit x 17-bit multiplier. It supports unsigned, signed or mixed sign operation in several multiplication modes:

- 16-bit x 16-bit signed
- 16-bit x 16-bit unsigned
- 16-bit signed x 5-bit (literal) unsigned
- 16-bit unsigned x 16-bit unsigned
- 16-bit unsigned x 5-bit (literal) unsigned
- 16-bit unsigned x 16-bit signed
- 8-bit unsigned x 8-bit unsigned

3.3.2 DIVIDER

The divide block supports 32-bit/16-bit and 16-bit/16-bit signed and unsigned integer divide operations with the following data sizes:

1. 32-bit signed/16-bit signed divide
2. 32-bit unsigned/16-bit unsigned divide
3. 16-bit signed/16-bit signed divide
4. 16-bit unsigned/16-bit unsigned divide

The quotient for all divide instructions ends up in W0 and the remainder in W1. The 16-bit signed and unsigned `DIV` instructions can specify any W register for both the 16-bit divisor (Wn), and any W register (aligned) pair (W(m + 1):Wm) for the 32-bit dividend. The divide algorithm takes one cycle per bit of divisor, so both 32-bit/16-bit and 16-bit/16-bit instructions take the same number of cycles to execute.

3.3.3 MULTI-BIT SHIFT SUPPORT

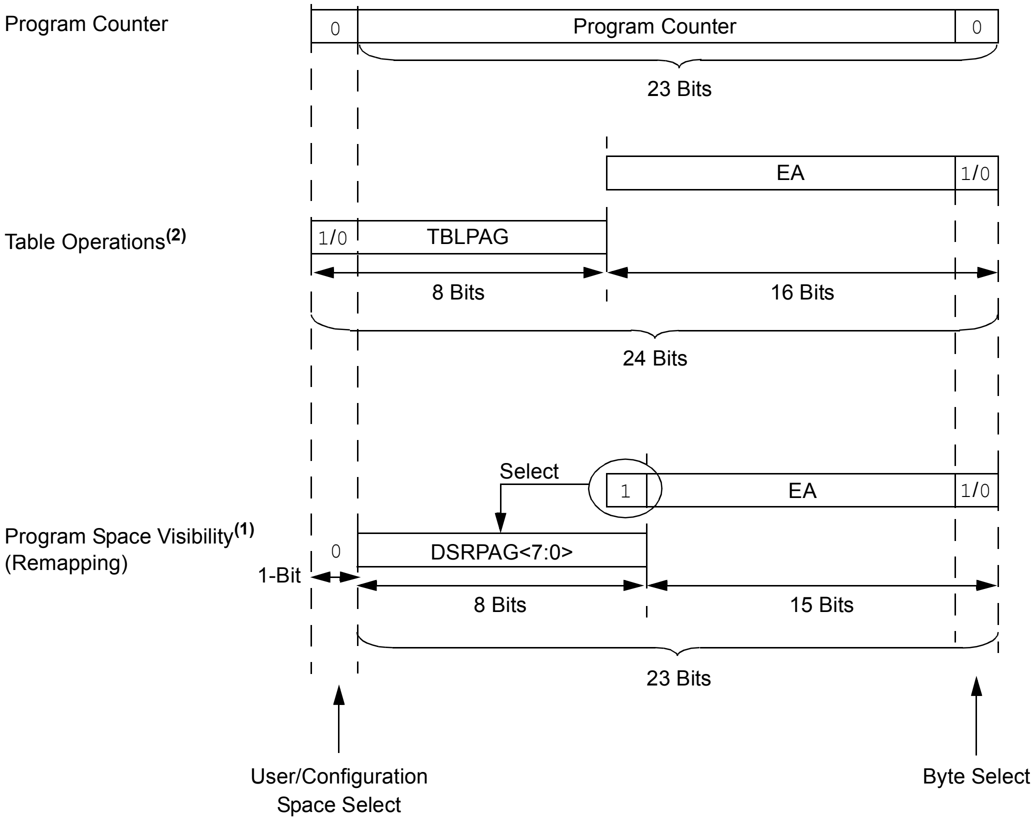
The PIC24F ALU supports both single bit and single-cycle, multi-bit arithmetic and logic shifts. Multi-bit shifts are implemented using a shifter block, capable of performing up to a 15-bit arithmetic right shift, or up to a 15-bit left shift, in a single cycle. All multi-bit shift instructions only support Register Direct Addressing for both the operand source and result destination.

A full summary of instructions that use the shift operation is provided in Table 3-2.

TABLE 3-2: INSTRUCTIONS THAT USE THE SINGLE BIT AND MULTI-BIT SHIFT OPERATION

Instruction	Description
`ASR`	Arithmetic Shift Right Source register by one or more bits.
`SL`	Shift Left Source register by one or more bits.
`LSR`	Logical Shift Right Source register by one or more bits.

FIGURE 4-8: DATA ACCESS FROM PROGRAM SPACE ADDRESS GENERATION



Note 1: DSRPAG<8> acts as word select. DSRPAG<9> should always be '`1`' to map program memory to data memory.

2: The instructions, `TBLRDH/TBLWTH/TBLRDL/TBLWTL`, decide if the higher or lower word of program memory is accessed. `TBLRDH/TBLWTH` instructions access the higher word and `TBLRDL/TBLWTL` instructions access the lower word. Table read operations are permitted in the configuration memory space.

5.0 DIRECT MEMORY ACCESS CONTROLLER (DMA)

Note:	This data sheet summarizes the features of this group of PIC24F devices. It is not intended to be a comprehensive reference source. For more information, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"Direct Memory Access Controller (DMA)"** (DS39742). The information in this data sheet supersedes the information in the FRM.

The Direct Memory Access Controller (DMA) is designed to service high data throughput peripherals operating on the SFR bus, allowing them to access data memory directly and alleviating the need for CPU-intensive management. By allowing these data-intensive peripherals to share their own data path, the main data bus is also deloaded, resulting in additional power savings.

The DMA Controller functions both as a peripheral and a direct extension of the CPU. It is located on the microcontroller data bus between the CPU and DMA-enabled peripherals, with direct access to SRAM. This partitions the SFR bus into two buses, allowing the DMA Controller access to the DMA capable peripherals located on the new DMA SFR bus. The controller serves as a master device on the DMA SFR bus, controlling data flow from DMA capable peripherals.

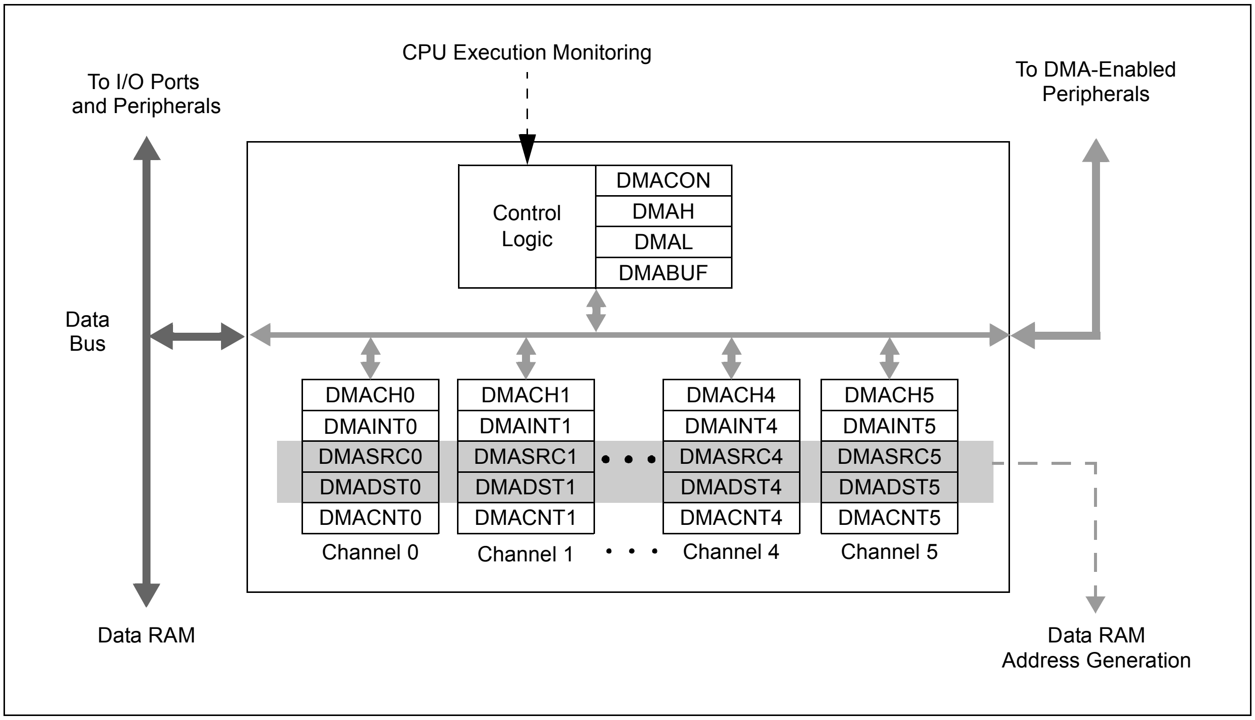
The controller also monitors CPU instruction processing directly, allowing it to be aware of when the CPU requires access to peripherals on the DMA bus and automatically relinquishing control to the CPU as needed. This increases the effective bandwidth for handling data without DMA operations causing a processor stall. This makes the controller essentially transparent to the user.

The DMA Controller has these features:

- Six multiple independent and independently programmable channels
- Concurrent operation with the CPU (no DMA caused Wait states)
- DMA bus arbitration
- Five Programmable Address modes
- Four Programmable Transfer modes
- Four Flexible Internal Data Transfer modes
- Byte or word support for data transfer
- 16-Bit Source and Destination Address register for each channel, dynamically updated and reloadable
- 16-Bit Transaction Count register, dynamically updated and reloadable
- Upper and Lower Address Limit registers
- Counter half-full level interrupt
- Software-triggered transfer
- Null Write mode for symmetric buffer operations

A simplified block diagram of the DMA Controller is shown in Figure 5-1.

FIGURE 5-1: DMA FUNCTIONAL BLOCK DIAGRAM



EXAMPLE 6-2: ERASING A PROGRAM MEMORY BLOCK (‘C’ LANGUAGE CODE)

```
// C example using MPLAB C30
    unsigned long progAddr = 0xXXXXXX;           // Address of row to write
    unsigned int offset;
//Set up pointer to the first memory location to be written
    TBLPAG = progAddr>>16;                       // Initialize PM Page Boundary SFR
    offset = progAddr & 0xFFFF;                  // Initialize lower word of address
    __builtin_tblwtl(offset, 0x0000);            // Set base address of erase block
                                                 // with dummy latch write
    NVMCON = 0x4042;                             // Initialize NVMCON
    asm("DISI #5");                              // Block all interrupts with priority <7
                                                 // for next 5 instructions
    __builtin_write_NVM();                       // check function to perform unlock
                                                 // sequence and set WR
```

EXAMPLE 6-3: LOADING THE WRITE BUFFERS

```
; Set up NVMCON for row programming operations
        MOV    #0x4001, W0                       ;
        MOV    W0, NVMCON                        ; Initialize NVMCON
; Set up a pointer to the first program memory location to be written
; program memory selected, and writes enabled
        MOV    #0x0000, W0                       ;
        MOV    W0, TBLPAG                        ; Initialize PM Page Boundary SFR
        MOV    #0x6000, W0                       ; An example program memory address
; Perform the TBLWT instructions to write the latches
; 0th_program_word
        MOV    #LOW_WORD_0, W2                   ;
        MOV    #HIGH_BYTE_0, W3                  ;
        TBLWTL W2, [W0]                          ; Write PM low word into program latch
        TBLWTH W3, [W0++]                        ; Write PM high byte into program latch
; 1st_program_word
        MOV    #LOW_WORD_1, W2                   ;
        MOV    #HIGH_BYTE_1, W3                  ;
        TBLWTL W2, [W0]                          ; Write PM low word into program latch
        TBLWTH W3, [W0++]                        ; Write PM high byte into program latch
;  2nd_program_word
        MOV    #LOW_WORD_2, W2                   ;
        MOV    #HIGH_BYTE_2, W3                  ;
        TBLWTL W2, [W0]                          ; Write PM low word into program latch
        TBLWTH W3, [W0++]                        ; Write PM high byte into program latch
        •
        •
        •
; 63rd_program_word
        MOV    #LOW_WORD_63, W2                  ;
        MOV    #HIGH_BYTE_63, W3                 ;
        TBLWTL W2, [W0]                          ; Write PM low word into program latch
        TBLWTH W3, [W0]                          ; Write PM high byte into program latch
```

EXAMPLE 6-4: INITIATING A PROGRAMMING SEQUENCE

```
        DISI   #5                         ; Block all interrupts with priority <7
                                          ; for next 5 instructions
        MOV.B  #0x55, W0
        MOV    W0, NVMKEY                 ; Write the 0x55 key
        MOV.B  #0xAA, W1                  ;
        MOV    W1, NVMKEY                 ; Write the 0xAA key
        BSET   NVMCON, #WR                ; Start the programming sequence
        NOP                               ; Required delays
        NOP
        BTSC   NVMCON, #15                ; and wait for it to be
        BRA    $-2                        ; completed
```

REGISTER 8-6: IFS1: INTERRUPT FLAG STATUS REGISTER 1 (CONTINUED)

bit 2	**CMIF:** Comparator Interrupt Flag Status bit `1` = Interrupt request has occurred `0` = Interrupt request has not occurred
bit 1	**MI2C1IF:** Master I2C1 Event Interrupt Flag Status bit `1` = Interrupt request has occurred `0` = Interrupt request has not occurred
bit 0	**SI2C1IF:** Slave I2C1 Event Interrupt Flag Status bit `1` = Interrupt request has occurred `0` = Interrupt request has not occurred

REGISTER 8-7: IFS2: INTERRUPT FLAG STATUS REGISTER 2 (CONTINUED)

bit 1	**SPI2TXIF:** SPI2 Transmit Interrupt Flag Status bit `1` = Interrupt request has occurred `0` = Interrupt request has not occurred
bit 0	**SPI2IF:** SPI2 General Interrupt Flag Status bit `1` = Interrupt request has occurred `0` = Interrupt request has not occurred

8.4 Interrupt Setup Procedures

8.4.1 INITIALIZATION

To configure an interrupt source:

1. Set the NSTDIS (INTCON1<15>) control bit if nested interrupts are not desired.
2. Select the user-assigned priority level for the interrupt source by writing the control bits in the appropriate IPCx register. The priority level will depend on the specific application and type of interrupt source. If multiple priority levels are not desired, the IPCx register control bits for all enabled interrupt sources may be programmed to the same non-zero value.

Note: At a device Reset, the IPCx registers are initialized, such that all user interrupt sources are assigned to Priority Level 4.

3. Clear the interrupt flag status bit associated with the peripheral in the associated IFSx register.
4. Enable the interrupt source by setting the interrupt enable control bit associated with the source in the appropriate IECx register.

8.4.2 INTERRUPT SERVICE ROUTINE (ISR)

The method that is used to declare an Interrupt Service Routine (ISR) and initialize the IVT with the correct vector address will depend on the programming language (i.e., ‘C’ or assembler) and the language development toolsuite that is used to develop the application. In general, the user must clear the interrupt flag in the appropriate IFSx register for the source of the interrupt that the ISR handles; otherwise, the ISR will be re-entered immediately after exiting the routine. If the ISR is coded in assembly language, it must be terminated using a `RETFIE` instruction to unstack the saved PC value, SRL value and old CPU priority level.

8.4.3 TRAP SERVICE ROUTINE (TSR)

A Trap Service Routine (TSR) is coded like an ISR, except that the appropriate trap status flag in the INTCON1 register must be cleared to avoid re-entry into the TSR.

8.4.4 INTERRUPT DISABLE

All user interrupts can be disabled using the following procedure:

1. Push the current SR value onto the software stack using the `PUSH` instruction.
2. Force the CPU to Priority Level 7 by inclusive ORing the value, 0Eh, with SRL.

To enable user interrupts, the `POP` instruction may be used to restore the previous SR value.

Note that only user interrupts with a priority level of 7 or less can be disabled. Trap sources (Levels 8-15) cannot be disabled.

The `DISI` instruction provides a convenient way to disable interrupts of Priority Levels 1-6 for a fixed period of time. Level 7 interrupt sources are not disabled by the `DISI` instruction.

10.5.2 WAKE-UP FROM VBAT MODES

When VDD is restored to a device in VBAT mode, it automatically wakes. Wake-up occurs with a POR, after which, the device starts executing code from the Reset vector. All SFRs, except the Deep Sleep Semaphores, are reset to their POR values. IF the RTCC was not configured to run during VBAT mode, it will remain disabled and RTCC will not run. Wake-up timing is similar to that for a normal POR.

To differentiate a wake-up from VBAT mode, from other POR states, check the VBAT status bit (RCON2<0>). If this bit is set while the device is starting to execute the code from the Reset vector, it indicates that there has been an exit from VBAT mode. The application must clear the VBAT bit to ensure that future VBAT wake-up events are captured.

If a POR occurs without a power source connected to the VBAT pin, the VBPOR bit (RCON2<1>) is set. If this bit is set on a Power-on Reset, it indicates that a battery needs to be connected to the VBAT pin.

In addition, if the VBAT power source falls below the level needed for Deep Sleep Semaphore operation while in VBAT mode (e.g., the battery has been drained), the VBPOR bit will be set. VBPOR is also set when the microcontroller is powered up the very first time, even if power is supplied to VBAT.

10.5.3 I/O PINS DURING VBAT MODES

All I/O pins switch to Input mode during VBAT mode. The only exceptions are the SOSCI and SOSCO pins, which maintain their states if the Secondary Oscillator is being used as the RTCC clock source. It is the user's responsibility to restore the I/O pins to their proper states using the TRISx and LATx bits once VDD has been restored.

10.5.4 SAVING CONTEXT DATA WITH THE DSGPRx REGISTERS

As with Deep Sleep mode (i.e., without the low-voltage/retention regulator), all SFRs are reset to their POR values after VDD has been restored. Only the Deep Sleep Semaphore registers are preserved. Applications which require critical data to be saved should save it in DSGPR0 and DSGPR1.

Note: If the VBAT mode is not used, it is recommended to connect the VBAT pin to VDD.

The POR should be enabled for the reliable operation of the VBAT.

10.6 Clock Frequency and Clock Switching

In Run and Idle modes, all PIC24FJ devices allow for a wide range of clock frequencies to be selected under application control. If the system clock configuration is not locked, users can choose low-power or high-precision oscillators by simply changing the NOSCx bits. The process of changing a system clock during operation, as well as limitations to the process, are discussed in more detail in **Section 9.0 “Oscillator Configuration”**.

10.7 Doze Mode

Generally, changing clock speed and invoking one of the power-saving modes are the preferred strategies for reducing power consumption. There may be circumstances, however, where this is not practical. For example, it may be necessary for an application to maintain uninterrupted synchronous communication, even while it is doing nothing else. Reducing system clock speed may introduce communication errors, while using a power-saving mode may stop communications completely.

Doze mode is a simple and effective alternative method to reduce power consumption while the device is still executing code. In this mode, the system clock continues to operate from the same source and at the same speed. Peripheral modules continue to be clocked at the same speed, while the CPU clock speed is reduced. Synchronization between the two clock domains is maintained, allowing the peripherals to access the SFRs while the CPU executes code at a slower rate.

Doze mode is enabled by setting the DOZEN bit (CLKDIV<11>). The ratio between peripheral and core clock speed is determined by the DOZE<2:0> bits (CLKDIV<14:12>). There are eight possible configurations, from 1:1 to 1:128, with 1:1 being the default.

It is also possible to use Doze mode to selectively reduce power consumption in event driven applications. This allows clock-sensitive functions, such as synchronous communications, to continue without interruption while the CPU Idles, waiting for something to invoke an interrupt routine. Enabling the automatic return to full-speed CPU operation on interrupts is enabled by setting the ROI bit (CLKDIV<15>). By default, interrupt events have no effect on Doze mode operation.

10.8 Selective Peripheral Module Control

Idle and Doze modes allow users to substantially reduce power consumption by slowing or stopping the CPU clock. Even so, peripheral modules still remain clocked, and thus, consume power. There may be cases where the application needs what these modes do not provide: the allocation of power resources to CPU processing, with minimal power consumption from the peripherals.

PIC24F devices address this requirement by allowing peripheral modules to be selectively disabled, reducing or eliminating their power consumption. This can be done with two control bits:

- The Peripheral Enable bit, generically named, “XXXEN”, located in the module’s main control SFR.
- The Peripheral Module Disable (PMD) bit, generically named, “XXXMD”, located in one of the PMDx Control registers (XXXMD bits are in the PMD1, PMD2, PMD3, PMD4, PMD6, PMD7, PMD8 registers).

Both bits have similar functions in enabling or disabling its associated module. Setting the PMD bit for a module disables all clock sources to that module, reducing its power consumption to an absolute minimum. In this state, the control and status registers associated with the peripheral will also be disabled, so writes to those registers will have no effect and read values will be invalid. Many peripheral modules have a corresponding PMD bit.

In contrast, disabling a module by clearing its XXXEN bit disables its functionality, but leaves its registers available to be read and written to. Power consumption is reduced, but not by as much as the use of the PMD bits. Most peripheral modules have an enable bit; exceptions include capture, compare and RTCC.

To achieve more selective power savings, peripheral modules can also be selectively disabled when the device enters Idle mode. This is done through the control bit of the generic name format, “XXXSIDL”. By default, all modules that can operate during Idle mode will do so. Using the disable on Idle feature disables the module while in Idle mode, allowing further reduction of power consumption during Idle mode, enhancing power savings for extremely critical power applications.

16.1 Standard Master Mode

To set up the SPIx module for the Standard Master mode of operation:

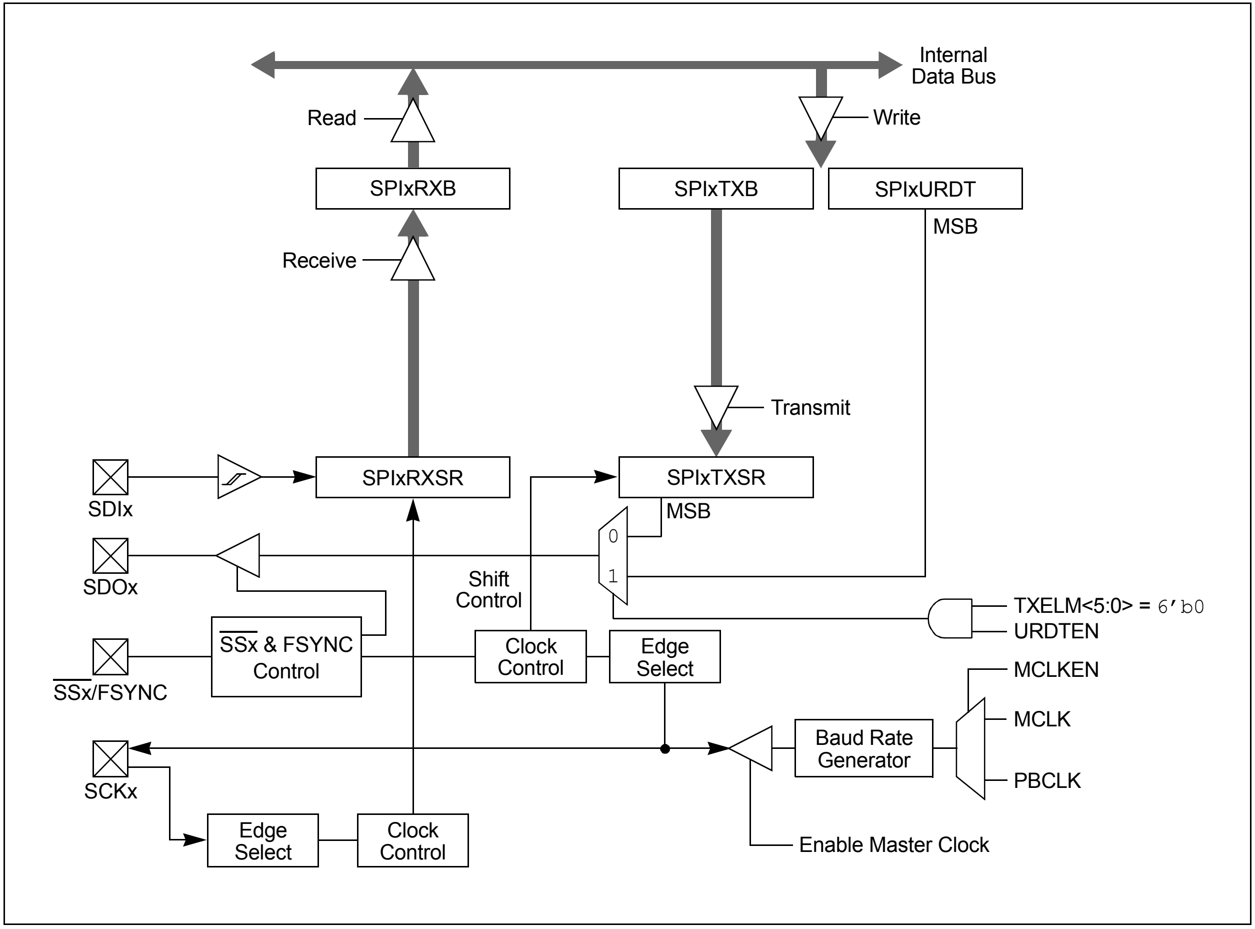
1. If using interrupts:
    a) Clear the interrupt flag bits in the respective IFSx register.
    b) Set the interrupt enable bits in the respective IECx register.
    c) Write the SPIxIP<2:0> bits in the respective IPCx register to set the interrupt priority.
2. Write the desired settings to the SPIxCON1L and SPIxCON1H registers with the MSTEN bit (SPIxCON1L<5>) = `1`.
3. Clear the SPIROV bit (SPIxSTATL<6>).
4. Enable SPIx operation by setting the SPIEN bit (SPIxCON1L<15>).
5. Write the data to be transmitted to the SPIxBUFL and SPIxBUFH registers. Transmission (and reception) will start as soon as data is written to the SPIxBUFL and SPIxBUFH registers.

16.2 Standard Slave Mode

To set up the SPIx module for the Standard Slave mode of operation:

1. Clear the SPIxBUF registers.
2. If using interrupts:
    a) Clear the SPIxBUFL and SPIxBUFH registers.
    b) Set the interrupt enable bits in the respective IECx register.
    c) Write the SPIxIP<2:0> bits in the respective IPCx register to set the interrupt priority.
3. Write the desired settings to the SPIxCON1L, SPIxCON1H and SPIxCON2L registers with the MSTEN bit (SPIxCON1L<5>) = `0`.
4. Clear the SMP bit.
5. If the CKE bit (SPIxCON1L<8>) is set, then the SSEN bit (SPIxCON1L<7>) must be set to enable the $\overline{\text{SSx}}$ pin.
6. Clear the SPIROV bit (SPIxSTATL<6>).
7. Enable SPIx operation by setting the SPIEN bit (SPIxCON1L<15>).

FIGURE 16-1: SPIx MODULE BLOCK DIAGRAM (STANDARD MODE)



REGISTER 19-18: U1IE: USB INTERRUPT ENABLE REGISTER (ALL USB MODES)

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

R/K-0, HS	R/K-0, HS	R/K-0, HS	R/K-0, HS	R/K-0, HS	R/K-0, HS	R/K-0, HS	R/K-0, HS
STALLIE	ATTACHIE[1]	RESUMEIE	IDLEIE	TRNIE	SOFIE	UERRIE	URSTIE DETACHIE[1]
bit 7							bit 0

Legend:	U = Unimplemented bit, read as ‘0’		
R = Readable bit	K = Write ‘1’ to Clear bit	HS = Hardware Settable bit	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 15-8 **Unimplemented:** Read as ‘0’

bit 7 **STALLIE:** STALL Handshake Interrupt Enable bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 6 **ATTACHIE:** Peripheral Attach Interrupt bit (Host mode only)[1]
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 5 **RESUMEIE:** Resume Interrupt bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 4 **IDLEIE:** Idle Detect Interrupt bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 3 **TRNIE:** Token Processing Complete Interrupt bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 2 **SOFIE:** Start-of-Frame Token Interrupt bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 1 **UERRIE:** USB Error Condition Interrupt bit
1 = Interrupt is enabled
0 = Interrupt is disabled

bit 0 **URSTIE or DETACHIE:** USB Reset Interrupt (Device mode) or
USB Detach Interrupt (Host mode) Enable bit[1]
1 = Interrupt is enabled
0 = Interrupt is disabled

Note 1: The ATTACHIE and DETACHIE bits are unimplemented in Device mode, read as ‘0’.

REGISTER 20-3: MDCAR: DATA SIGNAL MODULATOR CARRIER CONTROL REGISTER

R/W-x	R/W-x	R/W-x	U-0	R/W-x	R/W-x	R/W-x	R/W-x
CHODIS	CHPOL	CHSYNC	—	CH3(1)	CH2(1)	CH1(1)	CH0(1)
bit 15							bit 8

R/W-0	R/W-x	R/W-x	U-0	R/W-x	R/W-x	R/W-x	R/W-x
CLODIS	CLPOL	CLSYNC	—	CL3(1)	CL2(1)	CL1(1)	CL0(1)
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 15 **CHODIS:** DSM High Carrier Output Disable bit

`1` = Output signal driving the peripheral output pin (selected by CH<3:0>) is disabled
`0` = Output signal driving the peripheral output pin is enabled

bit 14 **CHPOL:** DSM High Carrier Polarity Select bit

`1` = Selected high carrier signal is inverted
`0` = Selected high carrier signal is not inverted

bit 13 **CHSYNC:** DSM High Carrier Synchronization Enable bit

`1` = Modulator waits for a falling edge on the high carrier before allowing a switch to the low carrier
`0` = Modulator output is not synchronized to the high time carrier signal(1)

bit 12 **Unimplemented:** Read as ‘`0`’

bit 11-8 **CH<3:0>** DSM Data High Carrier Selection bits(1)

`1111`
. . . = Reserved
`1010`
`1001` = Output Compare/PWM Module 6 output
`1000` = Output Compare/PWM Module 5 output
`0111` = Output Compare/PWM Module 4 output
`0110` = Output Compare/PWM Module 3 output
`0101` = Output Compare/PWM Module 2 output
`0100` = Output Compare/PWM Module 1 output
`0011` = Reference Clock Output (REFO)
`0010` = Input on MDCIN2 pin
`0001` = Input on MDCIN1 pin
`0000` = Vss

bit 7 **CLODIS:** DSM Low Carrier Output Disable bit

`1` = Output signal driving the peripheral output pin (selected by CL<3:0>) is disabled
`0` = Output signal driving the peripheral output pin is enabled

bit 6 **CLPOL:** DSM Low Carrier Polarity Select bit

`1` = Selected low carrier signal is inverted
`0` = Selected low carrier signal is not inverted

bit 5 **CLSYNC:** DSM Low Carrier Synchronization Enable bit

`1` = Modulator waits for a falling edge on the low carrier before allowing a switch to the high carrier
`0` = Modulator output is not synchronized to the low time carrier signal(1)

bit 4 **Unimplemented:** Read as ‘`0`’

bit 3-0 **CL<3:0>:** DSM Data Low Carrier Selection bits(1)

Bit settings are identical to those for CH<3:0>.

Note 1: Narrowed carrier pulse widths or spurs may occur in the signal stream if the carrier is not synchronized.

NOTES:

REGISTER 24-2: CRCCON2: CRC CONTROL 2 REGISTER

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	DWIDTH4	DWIDTH3	DWIDTH2	DWIDTH1	DWIDTH0
bit 15							bit 8

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	PLEN4	PLEN3	PLEN2	PLEN1	PLEN0
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 15-13	**Unimplemented:** Read as ‘0’
bit 12-8	**DWIDTH<4:0>:** Data Word Width Configuration bits Configures the width of the data word (Data Word Width – 1).
bit 7-5	**Unimplemented:** Read as ‘0’
bit 4-0	**PLEN<4:0>:** Polynomial Length Configuration bits Configures the length of the polynomial (Polynomial Length – 1).

REGISTER 24-3: CRCXORL: CRC XOR POLYNOMIAL REGISTER, LOW BYTE

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
X<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
X<7:1>							—
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 15-1	**X<15:1>:** XOR of Polynomial Term x^n Enable bits
bit 0	**Unimplemented:** Read as ‘0’

REGISTER 24-4: CRCXORH: CRC XOR POLYNOMIAL REGISTER, HIGH BYTE

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
X<31:24>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
X<23:16>							
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 15-0	**X<31:16>:** XOR of Polynomial Term x^n Enable bits

29.0 HIGH/LOW-VOLTAGE DETECT (HLVD)

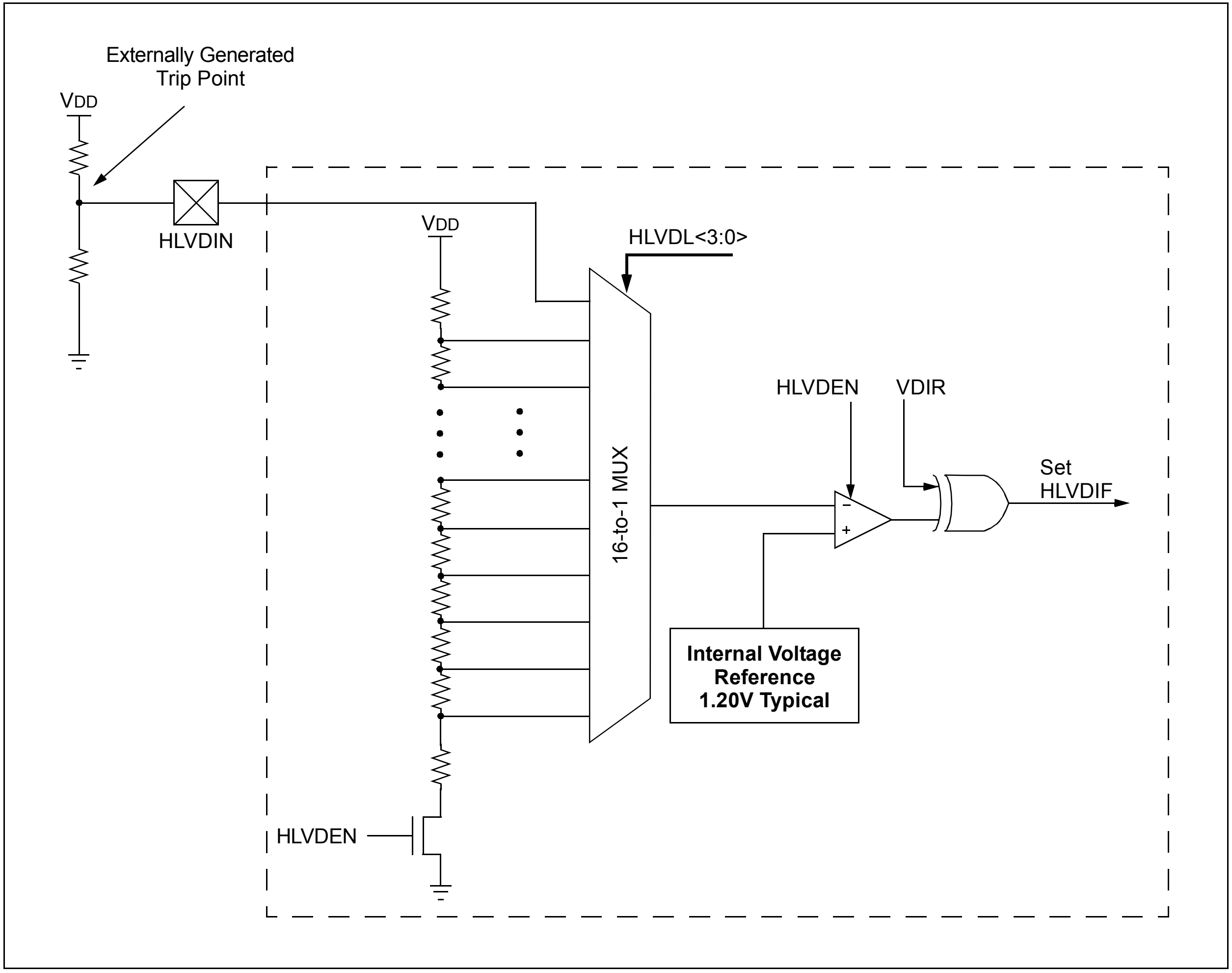
Note: This data sheet summarizes the features of this group of PIC24F devices. It is not intended to be a comprehensive reference source. For more information on the High/Low-Voltage Detect, refer to the *"dsPIC33/PIC24 Family Reference Manual"*, **"High-Level Integration with Programmable High/Low-Voltage Detect (HLVD)"** (DS39725).

The High/Low-Voltage Detect (HLVD) module is a programmable circuit that allows the user to specify both the device voltage trip point and the direction of change.

An interrupt flag is set if the device experiences an excursion past the trip point in the direction of change. If the interrupt is enabled, the program execution will branch to the interrupt vector address and the software can then respond to the interrupt.

The HLVD Control register (see Register 29-1) completely controls the operation of the HLVD module. This allows the circuitry to be "turned off" by the user under software control, which minimizes the current consumption for the device.

FIGURE 29-1: HIGH/LOW-VOLTAGE DETECT (HLVD) MODULE BLOCK DIAGRAM



REGISTER 30-2: CW2: FLASH CONFIGURATION WORD 2 (CONTINUED)

bit 5	**OSCIOFCN:** OSCO Pin Configuration bit If POSCMD<1:0> = `11` or `00`: `1` = OSCO/CLKO/RA3 functions as CLKO (FOSC/2) `0` = OSCO/CLKO/RA3 functions as port I/O (RA3) If POSCMD<1:0> = `10` or `01`: OSCIOFCN has no effect on OSCO/CLKO/RA3.
bit 4-3	**WDTCLK<1:0>:** WDT Clock Source Select bits When WDTCMX = `1`: `11` = LPRC `10` = Either the 31 kHz FRC source or LPRC, depending on device configuration(1) `01` = SOSC input `00` = System clock when active, LPRC while in Sleep mode When WDTCMX = `0`: LPRC is always the WDT clock source.
bit 2	**Reserved:** Configure as ‘`1`’
bit 1-0	**POSCMD<1:0>:** Primary Oscillator Configuration bits `11` = Primary Oscillator mode is disabled `10` = HS Oscillator mode is selected `01` = XT Oscillator mode is selected `00` = EC Oscillator mode is selected

Note 1: The 31 kHz FRC source is used when a Windowed WDT mode is selected and the LPRC is not being used as the system clock. The LPRC is used when the device is in Sleep mode and in all other cases.

2: When VBUS functionality is used, this Configuration bit must be programmed to ‘`1`’.

REGISTER 30-4: CW4: FLASH CONFIGURATION WORD 4

U-1	U-1	U-1	U-1	U-1	U-1	U-1	U-1
—	—	—	—	—	—	—	—
bit 23							bit 16

R/PO-1	R/PO-1	R/PO-1	R/PO-1	R/PO-1	R/PO-1	r-1	R/PO-1
IOL1WAY	I2C1SEL	PLLDIV3	PLLDIV2	PLLDIV1	PLLDIV0	—	DSSWEN
bit 15							bit 8

R/PO-1	R/PO-1	R/PO-1	R/PO-1	R/PO-1	R/PO-1	R/PO-1	R/PO-1
DSWDTEN	DSBOREN	DSWDTOSC	DSWDTPS4	DSWDTPS3	DSWDTPS2	DSWDTPS1	DSWDTPS0
bit 7							bit 0

Legend:	r = Reserved bit	PO = Program Once bit	
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 23-16 **Unimplemented:** Read as ‘1’

bit 15 **IOL1WAY:** IOLOCK One-Way Set Enable bit

1 = The IOLOCK bit (OSCCON<6>) can be set once, provided the unlock sequence has been completed; once set, the Peripheral Pin Select registers cannot be written to a second time
0 = The IOLOCK bit can be set and cleared as needed, provided the unlock sequence has been completed

bit 14 **I2C1SEL:** Alternate I2C1 Location Select bit

1 = I2C1 uses the SCL1 and SDA1 pins
0 = I2C1 uses the ASCL1 and ASDA1 pins

bit 13-10 **PLLDIV<3:0>:** USB 96 MHz PLL Prescaler Select bits

1111 = PLL is disabled
1110 = 8x PLL is selected
1101 = 6x PLL is selected
1100 = 4x PLL is selected
1011
. . . . = Reserved, do not use
1000
0111 = Oscillator input divided by 12 (48 MHz input)
0110 = Oscillator input divided by 8 (32 MHz input)
0101 = Oscillator input divided by 6 (24 MHz input)
0100 = Oscillator input divided by 5 (20 MHz input)
0011 = Oscillator input divided by 4 (16 MHz input)
0010 = Oscillator input divided by 3 (12 MHz input)
0001 = Oscillator input divided by 2 (8 MHz input)
0000 = Oscillator input used directly (4 MHz input)

bit 9 **Reserved:** Always maintain as ‘1’

bit 8 **DSSWEN:** Deep Sleep Software Control Select bit

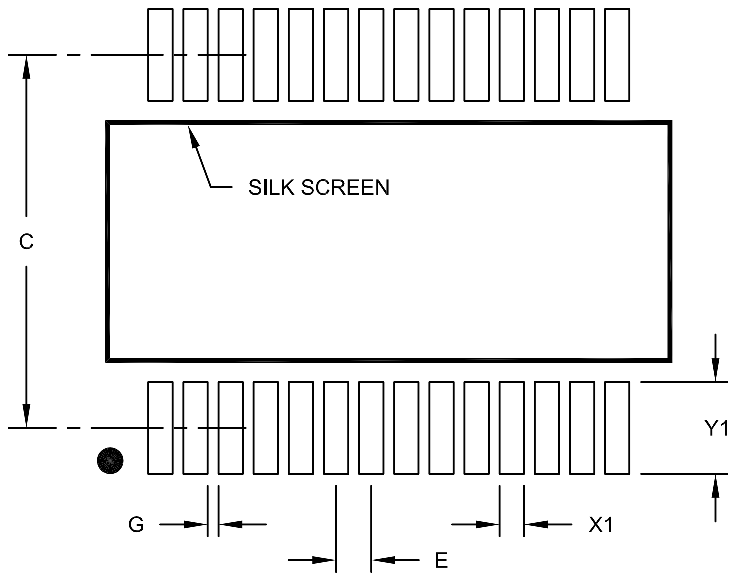
1 = Deep Sleep operation is enabled and controlled by the DSEN bit
0 = Deep Sleep operation is disabled

bit 7 **DSWDTEN:** Deep Sleep Watchdog Timer Enable bit

1 = Deep Sleep WDT is enabled
0 = Deep Sleep WDT is disabled

28-Lead Plastic Shrink Small Outline (SS) - 5.30 mm Body [SSOP]

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	0.65 BSC		
Contact Pad Spacing	C		7.20	
Contact Pad Width (X28)	X1			0.45
Contact Pad Length (X28)	Y1			1.75
Distance Between Pads	G	0.20		

Notes:

1. Dimensioning and tolerancing per ASME Y14.5M

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

Microchip Technology Drawing No. C04-2073A