Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

SX48BD

Configurable Communications Controllers with EE/Flash Program Memory, In-System Programming Capability and On-Chip Debug

1.0 PRODUCT OVERVIEW

1.1. Introduction

The Parallax SX48BD is a member of the SX family of configurable communications controllers fabricated in an advanced CMOS process technology. The advanced process, combined with a RISC-like architecture, allows high-speed computation, flexible I/O control, and efficient data manipulation. Throughput is enhanced by operating the device at frequencies up to 75 MHz and by optimizing the instruction set to include mostly single-cycle instructions. In addition, the SX architecture is deterministic and fully reprogrammable. The unique combination of these characteristics enable the device to serve in multitasking applications, high-speed communications, and virtually every application where speed and accuracy are essential.

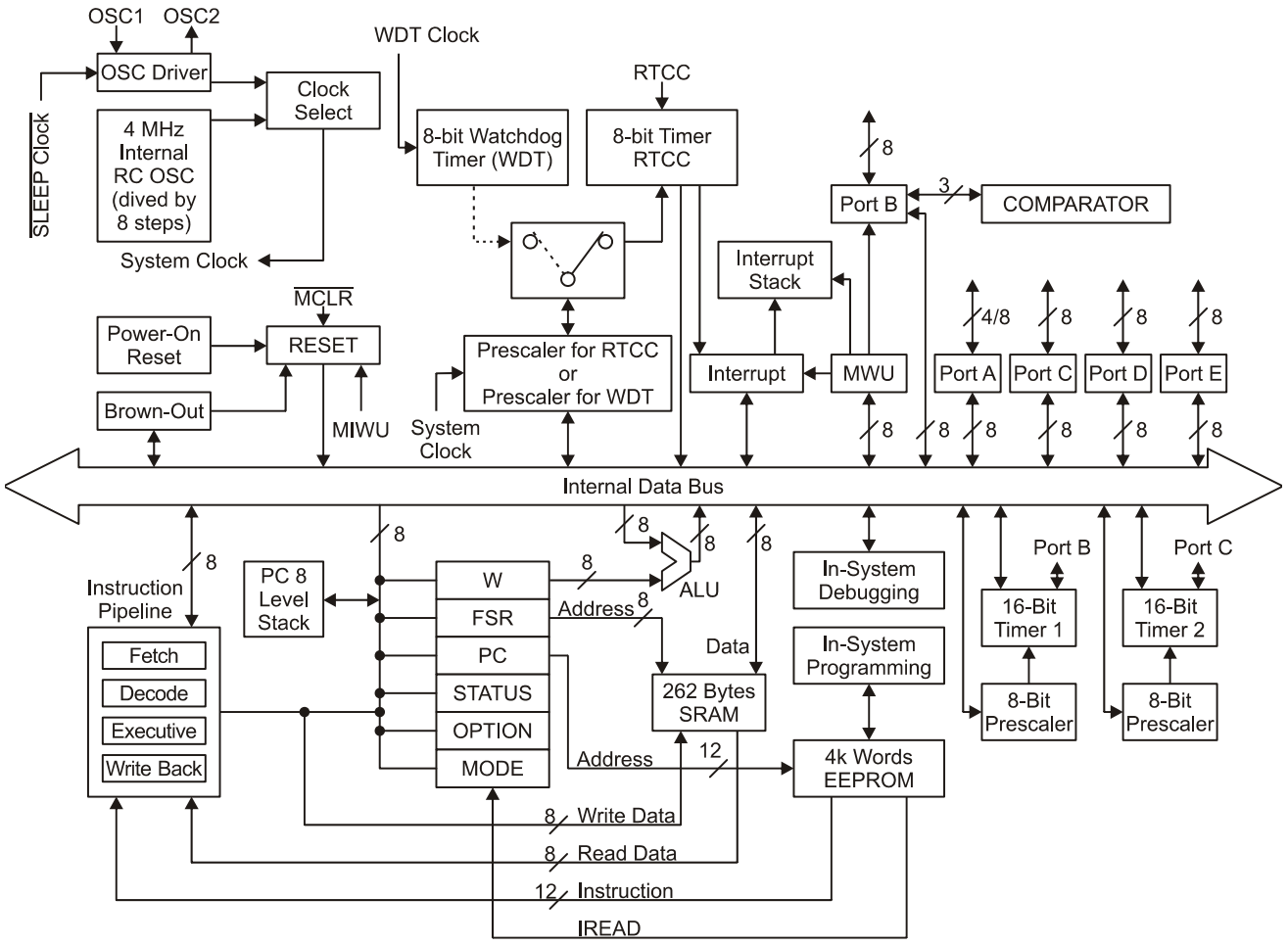


Figure 1-1: Block Diagram

Table of Contents

1.2. Key Features

75 MIPS Performance

- DC - 75 MHz operation
- 13.3 ns instruction cycle, 39.9 ns internal interrupt response at 75 MHz
- 1 instruction per clock for most instructions (skips require 2 clocks, branches require 3 clocks, IREAD requires 4)

EE/FLASH Program Memory and SRAM Data Memory

- Access time of < 13.3 ns provides single cycle access
- EE/Flash rated for > 10,000 rewrite cycles
- 4096 Words of EE/Flash program memory
- 262x8 bits SRAM data memory

CPU Features

- Compact, RISC-like instruction set
- All non-branch instructions are single cycle
- Eight-level push/pop hardware stack for subroutine linkage
- Fast table lookup capability through run-time readable code (IREAD instruction)
- Predictable program execution rate for hard real-time applications

Fast and Deterministic Interrupt

- Jitter-free 3-cycle internal interrupt response
- Hardware context save/restore of key resources such as PC, W, STATUS, and FSR within the 3-cycle interrupt response time
- External wakeup/interrupt capability on Port B (8 pins)

Flexible I/O

- All port pins individually programmable as I/O
- Inputs are TTL or CMOS level selectable
- All pins have selectable internal pull-ups
- Selectable Schmitt Trigger inputs on Ports B, C, D, and E
- All output pins capable of sourcing/sinking 30 mA
- Port A outputs have symmetrical drive
- Analog comparator support on Port B (RB0 OUT, RB1 IN-, RB2 IN+)
- Selectable I/O operation synchronous to the oscillator clock

Hardware Peripheral Features

- Two 16-bit timers with 8-bit prescalers supporting:
- Software Timer mode
- PWM mode
- Simultaneous PWM/Capture mode
- External Event mode
- One 8-bit Real Time Clock/Counter (RTCC) with programmable 8-bit prescaler
- Watchdog Timer (shares the RTCC prescaler)
- Analog comparator
- Brown-out detector
- Multi-Input Wakeup logic on 8 pins
- Internal RC oscillator with configurable rate from 31.25 kHz to 4 MHz
- Power-On-Reset

Package

- 48-pin Tiny PQFP

Programming and Debugging Support

- On- chip in-system programming support with serial or parallel interface
- In-system serial programming via oscillator pins
- On-chip in-System debugging support logic
- Real-time emulation, full program debug, and integrated development environment offered by third party tool vendors

Software Support

- Native assembly instruction set
- Expanded assembly instruction set available in the SASM assembler of the Parallax SX-Key IDE
- Parallax SX/B compiler (BASIC)
- Several “C” compliers available from third-party vendors

1.3. Architecture

The SX devices use a modified Harvard architecture. This architecture uses two separate memories with separate address buses, one for the program and one for data, while allowing transfer of data from program memory to SRAM. This ability allows accessing data tables from program memory. The advantage of this architecture is that instruction fetch and memory transfers can be overlapped with a multi-stage pipeline, which means the next instruction can be fetched from program memory while the current instruction is being executed using data from the data memory.

The SX uses a revolutionary RISC-like architecture and memory design technique that is 15 times faster than conventional MCUs, deterministic, jitter free, and fully reprogrammable. The SX family implements a four-stage pipeline (fetch, decode, execute, and write back), which results in execution of one instruction per clock cycle. At the maximum operating frequency of 75 MHz, instructions are executed at the rate of one per 13.3-ns clock cycle.

1.4. Programming Benefits in Assembly and High-Level Languages

The SX's high speed enables a "software system on a chip" approach. Programming in assembly language provides a particularly high-level of access to the interrupt service routine, the stack and registers to take the highest advantage of the SX's deterministic timing. The primary technical resources for programming the SX in assembly language include the following:

- The SX48BD datasheet
- *SX-Key Development System User's Manual* by Parallax, Inc.
- *Programming the SX Microcontroller – A Complete Guide* by Guenther Daubach

Customers with a high-level programming language background may prefer the use of a C or BASIC compiler.

1.4.1. Parallax SX/B Basic Compiler

Parallax's SX/B is a free BASIC language compiler for the SX microcontroller (SX20, SX28, and SX48). The compiler speeds the programming of the SX microcontrollers by providing a simple, yet robust high-level language familiar to Parallax customers. SX/B includes the following features and commands:

- ASM directive to support in-line assembly language
- Program structure commands including BRANCH, DO..LOOP, GOTO, GOSUB, IF..THEN..ELSE
- Numeric formatters
- WORD variable support
- Frequency generation with FREQOUT
- Synchronous serial communication for I2C, 1-Wire, SPI
- Asynchronous serial communication with SERIN and SEROUT
- Table data storage and retrieval with LOOKUP, LOOKDOWN
- I/O pin control with HIGH, LOW, TOGGLE, REVERSE
- Timing and delay with PAUSE, SLEEP
- PULSIN and PULSOUT
- Resistor/capacitor A/D with RCTIME
- RANDOM for pseudo-random number generation
- Non-volatile EEPROM memory access with DATA, READ
- Low-current SLEEP command

The complete SX/B command reference and examples are installed with the SX-Key IDE.

1.5. Programming and Debugging Support

The SX devices are supported by Parallax's programming and debugging tools. The Parallax SX-Blitz is a programming tool. The SX-Key supports programming and source-level debugging. On-chip in-system debug capabilities allow the Parallax tool to be an all-in-one integrated development environment with editor, macro assembler, debugger, and programmer. Unobtrusive in-system programming is provided through the OSC pins. Visit www.parallax.com for the SX-Key development tools, the IDE and support forum information.

The in-system programming specification is available to other 3rd party tool vendors upon request.

1.6. Applications

The SX may be used as a solution for process controllers, electronic appliances/tools, security/monitoring systems, sound and signal generation, GPS interface, robotic control, motor control, sensor interfacing and personal communication devices. Applications such as interactive toys, magnetic-stripe readers, infrared decoders, and other timing-sensitive projects are also common with the SX. Examples of customer applications may be seen on the Parallax web site.

1.7. Support

Parallax and our distributors provide support for the SX microcontroller. Support is available free of charge via phone (888) 512-1024 in the U.S. Also be sure to participate in the SX discussion forum at http://forums.parallax.com/forums/. The on-line SX support community is actively involved in customer support 24 hours a day.

1.8. Part Numbering

Table 20-1: Ordering Information							
Device Part#	**Pins**	**I/O**	**EE/Flash (Words)**	**RAM (Bytes)**	**Voltage Range (V)**	**Operating Temp @ 3.0 – 5.5 V, 50 MHz***	**Operating Temp @ 4.5 – 5.5 V, 75 MHz***
SX48BD/TQ	48	36	4 K	262	3.0 – 5.5	-40 °C to +85 °C	0 °C to +70 °C
SX48BD/TQ-G	48	36	4 K	262	3.0 – 5.5	-40 °C to +85 °C	0 °C to +70 °C

***Ratings are preliminary**

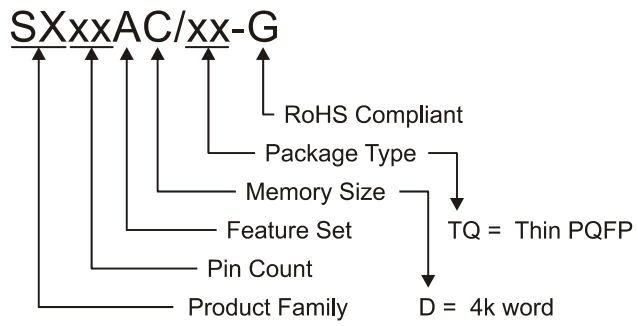
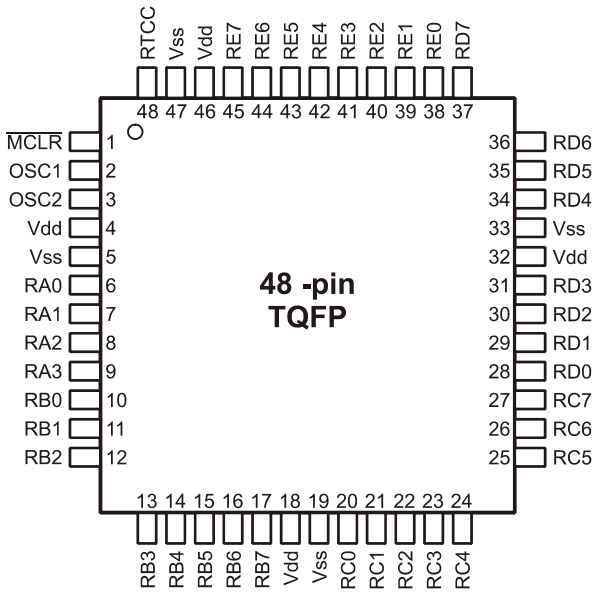


Figure 1-1

Part Number Reference Guide

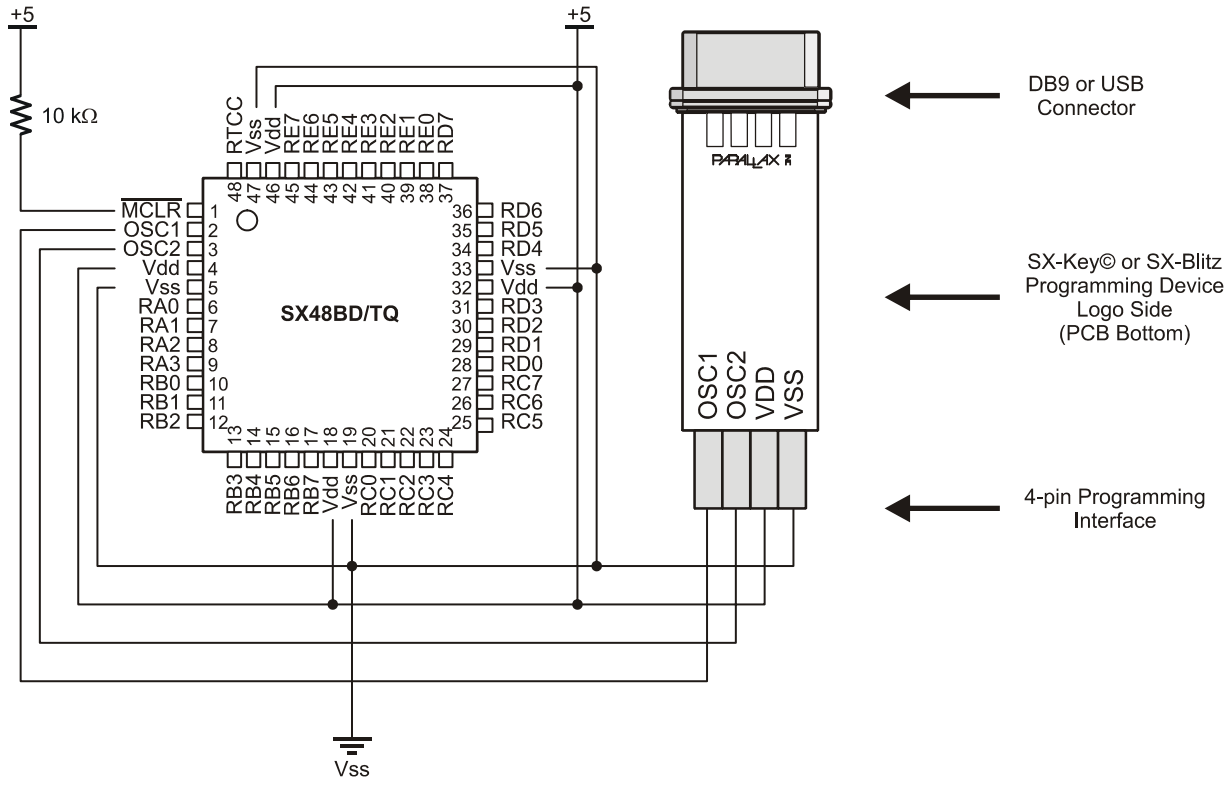
2.0 CONNECTION DIAGRAMS

2.1. Pin Assignments



Top View

2.2. Typical Connection Diagram



2.3. Pin Descriptions

Table 2-1: Pin Descriptions

Name	Pin Type	Input Levels	Description
RA0	I/O	TTL/CMOS	Bidirectional I/O Pin; symmetrical source / sink capability
RA1	I/O	TTL/CMOS	Bidirectional I/O Pin; symmetrical source / sink capability
RA2	I/O	TTL/CMOS	Bidirectional I/O Pin; symmetrical source / sink capability
RA3	I/O	TTL/CMOS	Bidirectional I/O Pin; symmetrical source / sink capability
RB0	I/O	TTL/CMOS/ST	Bidirectional I/O Pin; comparator output; MIWU/Interrupt input
RB1	I/O	TTL/CMOS/ST	Bidirectional I/O Pin; comparator negative input; MIWU/Interrupt input
RB2	I/O	TTL/CMOS/ST	Bidirectional I/O Pin; comparator positive input; MIWU/Interrupt input
RB3	I/O	TTL/CMOS/ST	Bidirectional I/O Pin; MIWU/Interrupt input,
RB4	I/O	TTL/CMOS/ST	Bidirectional I/O Pin; MIWU/Interrupt input, Timer T1 Capture Input 1
RB5	I/O	TTL/CMOS/ST	Bidirectional I/O Pin; MIWU/Interrupt input, Timer T1 Capture Input 2
RB6	I/O	TTL/CMOS/ST	Bidirectional I/O Pin; MIWU/Interrupt input, Timer T1 PWM/Compare Output
RB7	I/O	TTL/CMOS/ST	Bidirectional I/O Pin; MIWU/Interrupt input, Timer T1 External Event Counter Input
RC0	I/O	TTL/CMOS/ST	Bidirectional I/O Pin, Timer T2 Capture Input 1
RC1	I/O	TTL/CMOS/ST	Bidirectional I/O Pin, Timer T2 Capture Input 2
RC2	I/O	TTL/CMOS/ST	Bidirectional I/O Pin, Timer T2 PWM/compare Output
RC3	I/O	TTL/CMOS/ST	Bidirectional I/O Pin; Timer T2 External Event Counter Input
RC4	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RC5	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RC6	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RC7	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RD0	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RD1	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RD2	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RD3	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RD4	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RD5	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RD6	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RD7	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RE0	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RE1	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RE2	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RE3	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RE4	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RE5	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RE6	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RE7	I/O	TTL/CMOS/ST	Bidirectional I/O Pin
RTCC	I	ST	Input to Real-time Clock/Counter
$\overline{\text{MCLR}}$	I	ST	Master Clear reset input – active low. When not controlled externally, this pin must be pulled high with a 10 kΩ resistor.
OSC1/In/Vpp	I	ST	Crystal oscillator input – External Clock source input
OSC2/Out	O	CMOS	Crystal oscillator output; in R/C mode, internally pulled to V_{dd} through weak pull-up
V_{dd}	P	-	Positive supply pins (a total of 4, one on each side of the device)
V_{ss}	P	-	Ground pins (a total of 4, one on each side of the device)

Note: I = input, O = output, I/O = Input/Output, P = Power, TTL = TTL input, CMOS = CMOS input, ST = Schmitt Trigger input, MIWU = Multi-Input Wakeup input.

3.0 PORT DESCRIPTIONS

The device contains one 4-bit port (Port A) and four 8-bit I/O ports (Port B through Port E). Port A provides symmetrical drive capability. Each port has four associated 8-bit registers (Direction, Data, TTL/CMOS Select, and Pull-Up Enable) to configure each port pin as Hi-Z input or output, to select TTL or CMOS voltage levels, and to enable/disable the weak pull-up resistor. The least significant bit of the registers corresponds to the least significant port pin. To access these configuration registers, an appropriate value must be written into the MODE register. Upon power-up, all bits in these registers are initialized to "1".

The associated registers allow for each port bit to be individually configured under software control as shown in Table 3-2.

Ports B, C, D, and E have additional associated registers (Schmitt-Trigger Enable Registers ST_B and ST_C) to enable or disable the Schmitt Trigger function on each individual port pin as indicated in Table 3-1.

Table 3-1: Schmitt Trigger Select

Schmitt Trigger Enable Registers :ST_B, ST_C, ST_D, ST_E	
0	1
Enable	Disable

Port B also supports the on-chip differential comparator. Ports RB1 and RB2 are the comparator negative and positive inputs, respectively, while Port RB0 is the comparator output pin. Port B also supports the Multi-Input Wakeup feature on all eight pins.

Port B and Port C also support the multi-function timers T1 and T2. RB4 and RB5 are the T1 capture inputs, RB6 is the T1 PWM output, and RB7 is the T1 external event counter input. Similarly, RC0 and RC1 are the T2 capture inputs, RC2 is the T2 PWM output, and RC3 is the T2 external event counter input. Figure 3-1 shows the internal hardware structure and configuration registers for each pin of Port A. Figure 3-2 shows the same for each pin of Port B, C, D, or E.

Table 3-2: Port Configuration

Data Direction Registers: RA, RB, RC		TTL/CMOS Selected Registers: LVL_A, LVL_B, LVL_C, LVL_D, LVL_E		Pullup Enable Registers: PLP_A, PLP_B, PLP_C, PLP_D, PLP_E	
0	1	0	1	0	1
Output	Hi-Z Input	CMOS	TTL	Enable	Disable

3.1. Reading and Writing the Ports

The five ports are memory-mapped into the data memory address space. To the CPU, the five ports are available as the RA, RB, RC, RD, and RE file registers at data memory addresses 05h through 09h, respectively. Writing to a port data register sets the voltage levels of the corresponding port pins that have been configured to operate as outputs to a corresponding level, 1 = 5 V, 0 = 0 V. Reading from a data register reads either the voltage levels of the corresponding port pins or the data contained in the port data register depending on the status PORTRD bit contained in the T2CNTB register.

For example, suppose all four Port A pins are configured as outputs. To make RA0 and RA1 high and the remaining Port A pins low, you could use the following code:

```
mov     W,#$03  ;load W with the value 03h
                ;(bits 0 and 1 high)
mov     $05,W   ;write 03h to Port A data
                ;register
```

The second “mov” instruction in this example writes the Port A data register (RA), which controls the output levels of the Port A pins, RA0 through RA7. When a write is performed to a port bit position that has been configured as an input, a write to the port data register is still performed, but it has no immediate effect on the pin. If later that pin is configured to operate as an output, it will reflect the value that has been written to the data register.

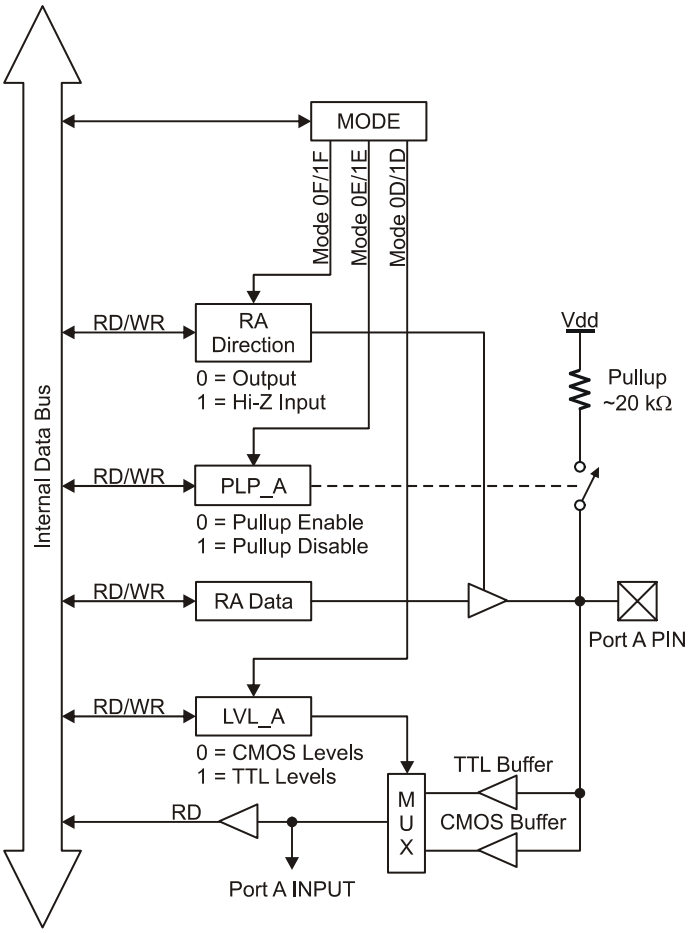


Figure 3-1

Port A Configuration

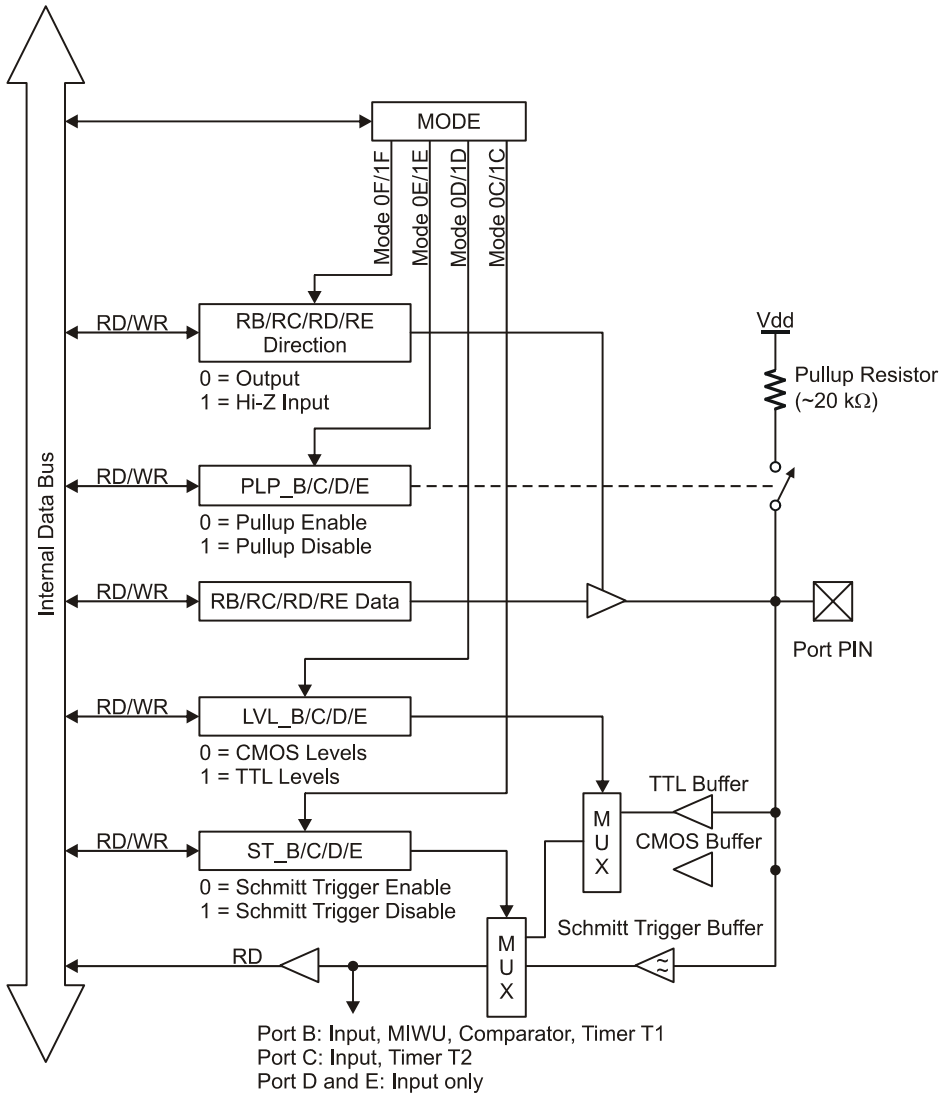


Figure 3-2: Port B, Port C, Port D, Port E Configuration

In the default device configuration, when a read is performed from a port bit position, the operation is actually reading the voltage level on the pin itself, and not the bit value stored in the port data register. This is true whether the pin is configured to operate as an input or an output. Therefore, with the pin configured to operate as an input, the data register contents have no effect on the value that you read. With the pin configured to operate as an output, what is read generally matches what has been written to the register. PORTRD of the T2CNT2 register determines how the device reads data from its I/O ports (Port A through Port E). Clear this bit to 0 to have the device read data from the port I/O pins directly. Set this bit to 1 to have the device read data from the port data registers. Under normal conditions, it should not matter which method you use to read the port data. However, if a port pin is configured as an output and an external circuit forces the pin to the opposite value, the value read from the port will depend on the reading mode used. Note that this control bit is not related to multi-function timers T1 and T2.

3.2. Read-Modify-Write Considerations

When two successive instructions are used on the same I/O port (except "mov Rx,W") with a very high clock rate, the "write" part of one instruction might not occur soon enough before the "read" part of the very next instruction, resulting in getting "old" data for the second instruction. To ensure predictable results, avoid using two successive read-modify-write instructions that access the same port data register if the clock rate is high or, insert 3 NOP instructions between the successive read-modify-write instructions (if SYNC bit in the FUSE register is enabled, 5 NOP instructions are required). For operating frequencies of 50 MHz or lower, if bit 7 of the T2CNTB (PORTRD) is set, the port reads data from the data register instead of port pins. In this case, the NOP instructions are not required.

3.3. Port Configuration

Each port pin offers the following configuration options:

- data direction
- input voltage levels (TTL or CMOS)
- pullup type (enable or disable)
- Schmitt trigger input (except for Port A)

Port B offers the additional option to use the port pins for the Multi-Input Wakeup/Interrupt function, the analog comparator function, or Timer T1 I/O. Port C offers the additional option to use the port pins for Timer T2 I/O. Port configuration is performed by writing to a set of control registers associated with the port. A special-purpose instruction is used to write these control registers:

```
mov !RA,W       ;move W to/from Port A
                ;control register
mov !RB,W       ;move W to/from Port B
                ;control register
mov !RC,W       ;move W to/from Port C
                ;control register
mov !RD,W       ;move W to/from Port D
                ;control register
mov !RE,W       ;move W to/from Port E
                ;control register
```

Each one of these instructions reads or writes a port control register for Port A, B, C, D, or E. There are multiple control registers for each port. To specify which one you want to access, you use another register called the MODE register.

3.3.1. MODE Register

The MODE register controls access to the port configuration registers and Timer T1/T2 control registers. Because the MODE register is not memory-mapped, it is accessed by the following special-purpose instructions:

```
mov M, #lit     ;move literal to lower 4-bits
                ;of MODE register
mov M,W         ;move W to lower 5-bits
                ;of MODE register
mov W,M         ;move MODE register to W
```

The value contained in the MODE register determines which port control register is accessed by the "mov !rx,W" instruction as indicated in Table 3-3. (The table also shows the timer control registers accessed according to the MODE register setting.) MODE register values not defined in the table are reserved for future expansion and should not be used. Upon power-up, the MODE register is initialized to 1Fh, which enables write access to the port direction control registers.

When bit 4 of the MODE register is 0 (the top half of Table 3-3), a "mov !rx,W" instruction moves the contents of the applicable control register into W. When bit 4 of the MODE register is 1 (the bottom half of Table 3-3), a "mov !rx,W" instruction moves the contents of W into the applicable control register. However, there are some exceptions to this. For the CMP_B and WKPND_B registers, the CPU does an exchange of data between W and the control register, regardless of the state of bit 4 in the MODE register. For the WKED_B and WKEN_B registers, the CPU moves the data from W to the control register, regardless of the state of bit 4 in the MODE register.

After a value is written to the MODE register, that setting remains in effect until it is changed by writing to the MODE register again. For example, you can write the value 1Eh to the MODE register just once, and then write to each of the five pullup configuration registers using the five "mov !rx,W" instructions.

Table 3-3: Mode Register Settings					
MODE Reg.	**Mov !RA,W**	**Mov !RB,W**	**Mov !RC,W**	**Mov !RD,W**	**Mov !RE,W**
00h		Read T1CPL	Read T2CPL		
01h		Read T1CPH	Read T2CPH		
02h		Read T1R2CML	Read T2R2CML		
03h		Read T1R2CMH	Read T2R2CMH		
04h		Read T1R1CML	Read T2R1CML		
05h		Read T1R1CMH	Read T2R1CMH		
06h		Read T1CNTB	Read T2CNTB		
07h		Read T1CNTA	Read T2CNTA		
08h		Exchange CMP_B with W			
09h		Exchange WKPND_B with W			
0Ah		Write WKED_B			
0Bh		Write $\overline{\text{WKEN\_B}}$			
0Ch		Read ST_B	Read ST_C	Read ST_D	Read ST_E
0Dh	Read LVL_A	Read LVL_B	Read LVL_C	Read LVL_D	Read LVL_E
0Eh	Read PLP_A	Read PLP_B	Read PLP_C	Read PLP_D	Read PLP_E
0Fh	Read RA Direction	Read RB Direction	Read RC Direction	Read RD Direction	Read RE Direction
10h		Clear Timer T1			
11h					
12h		Write T1R2CML	Write T2R2CML		
13h		Write T1R2CMH	Write T2R2CMH		
14h		Write T1R1CML	Write T2R1CML		
15h		Write T1R1CMH	Write T2R1CMH		
16h		Write T1CNTB	Write T2CNTB		
17h		Write T1CNTA	Write T2CNTA		
18h		Exchange CMP_B with W			
19h		Exchange WKPND_B with W			
1Ah		Write WKED_B			
1Bh		Write $\overline{\text{WKEN\_B}}$			
1Ch		Write ST_B	Write ST_C	Write ST_D	Write ST_E
1Dh	Write LVL_A	Write LVL_B	Write LVL_C	Write LVL_D	Write LVL_E
1Eh	Write PLP_A	Write PLP_B	Write PLP_C	Write PLP_D	Write PLP_E
1Fh	Write RA Direction	Write RB Direction	Write RC Direction	Write RD Direction	Write RE Direction

The following code example shows how to program the pullup control registers.

```
mov     W,#$1E  ;MODE=1Eh to write port pullup
mov     M, W    ;registers

mov     W,#$03  ;W = 0000 0011
mov     !RA,W   ;disable pullups for RA0 and RA1

mov     W,#$FF  ;W = 1111 1111
mov     !RB,W   ;disable all pullups for RB0-RB7

mov     W,#$00  ;W = 0000 0000
mov     !RC,W   ;enable all pullups for RC0-RC7
```

First the MODE register is loaded with 1Eh to select write access to the pullup control registers (PLP_A, PLP_B, and so on). Then the MOV !rx,W instructions are used to specify which port pins are to be connected to the internal pullup resistors. Setting a bit to 1 disconnects the corresponding pullup resistor, and clearing a bit to 0 connects the corresponding pullup resistor.

3.3.2. Port Configuration Registers

The port configuration registers that you control with the MOV !rx,W instruction operate as described below.

RA through RE Data Direction Registers (MODE=1Fh)

Each register bit sets the data direction for one port pin. Set the bit to 1 to make the pin operate as a high-impedance input. Clear the bit to 0 to make the pin operate as an output. Upon reset, the bit is set to 1.

PLP_A through PLP_E: Pullup Enable Registers (MODE=1Eh)

Each register bit determines whether an internal pullup resistor is connected to the pin. Set the bit to 1 to disconnect the pullup resistor or clear the bit to 0 to connect the pullup resistor. Upon reset, the bit is set to 1.

LVL_A through LVL_E: Input Level Registers (MODE=1Dh)

Each register bit determines the voltage levels sensed on the input port, either TTL or CMOS, when the Schmitt trigger option is disabled. Program each bit according to the type of device that is driving the port input pin. Set the bit to 1 for TTL or clear the bit to 0 for CMOS. Upon reset, the bit is set to 1. If $\overline{\text{SYNC}}$ is enabled in the FUSE register, port data must be read more than 2 cycles after a change to the input level mode or Schmitt Trigger mode (see Figure 3-2).

ST_B through ST_E: Schmitt Trigger Enable Registers (MODE=1Ch)

Each register bit determines whether the port input pin operates with a Schmitt trigger. Set the bit to 1 to disable Schmitt trigger operation and sense either TTL or CMOS voltage levels; or clear the bit to 0 to enable Schmitt trigger operation. Upon reset, the bit is set to 1. If $\overline{\text{SYNC}}$ is enabled in the FUSE register, port data must be read more than 2 cycles after a change to the input level mode or Schmitt Trigger mode (see Figure 3-2).

$\overline{\text{WKEN}}$_B: Wakeup Enable Register (MODE=1Bh)

Each register bit enables or disables the Multi-Input Wakeup/Interrupt (MIWU) function for the corresponding Port B input pin. Clear the bit to 0 to enable MIWU operation or set the bit to 1 to disable MIWU operation. Upon reset, the bit is set to 1. For more information on using the Multi-Input Wakeup/Interrupt function, see Section 8.0.

WKED_B: Wakeup Edge Register (MODE=1Ah)

Each register bit selects the edge sensitivity of the Port B input pin for MIWU operation. Clear the bit to 0 to sense rising (low-to-high) edges. Set the bit to 1 to sense falling (high-to-low) edges. Upon reset, the bit is set to 1.

WKPND_B: Wakeup Pending Flag Register (MODE=19h)

When you access the WKPND_B register using "mov !RB,W", the CPU exchanges the contents of W and WKPND_B. This feature lets you read the WKPND_B register contents while clearing the Wakeup Pending bits simultaneously. Each bit read from the WKPND_B register indicates the status of the corresponding MIWU pin. A bit set to 1 indicates that a valid edge has occurred on the corresponding MIWU pin, and has triggered a wakeup or interrupt. A bit cleared to 0 indicates that no valid edge has occurred on the MIWU pin.

CMP_B: Comparator Register (MODE=08h)

When you access the CMP_B register using MOV !RB,W, the CPU exchanges the contents of W and CMP_B. This feature lets you read the CMP_B register contents while writing a new value to the register. Clear bit 7 to enable operation of the comparator. Clear bit 6 to place the comparator result on the RB0 pin. Bit 0 is a result flag that is set to 1 when the voltage on RB2 (positive input) is greater than RB1 (negative input), or cleared to 0 otherwise. (For more information on using the comparator, see Section 12.0.)

3.3.3. Port Configuration Upon Power-Up

Upon power-up, all the port control registers are initialized to FFh. Thus, each port pin is configured to operate as a high-impedance input that senses TTL voltage levels, with no internal pullup resistor connected. The MODE register is initialized to 1Fh, which allows immediate write access to the data direction registers using the "MOV !rx,W" instruction.

4.0 SPECIAL-FUNCTION REGISTERS

The CPU uses a set of special-function registers to control operation of the device.

The CPU registers include an 8-bit working register (W), which serves as a pseudo accumulator. It holds the second operand of an instruction, receives the literal in immediate type instructions, and also can be program selected as the destination register.

A set of 31 file registers serves as the primary accumulator. One of these registers holds the first operand of an instruction and another can be program-selected as the destination register. The first 10 file registers include the Real-Time Clock/Counter register (RTCC), the lower eight bits of the 12-bit Program Counter (PC), the 8-bit STATUS register, five port control registers for Ports A through E, the 8-bit File Select Register (FSR), and INDF (used for indirect addressing).

The five low-order bits of the FSR register select one of the 31 file registers in the indirect addressing mode. Calling for the file register located at address 00h (INDF) in any of the file-oriented instructions selects indirect addressing, which uses the FSR register. It should be noted that the file register at address 00h is not a physically implemented register. The CPU also contains an 8 level, 12-bit hardware push/pop stack for subroutine linkage.

Table 4-1: Special-Function Register

Address	Name	Function
00h	INDF	Used for indirect addressing
01h	RTCC	Real Time Clock/Counter
02h	PC	Program Counter (low byte)
03h	STATUS	Holds status bits of ALU
04h	FSR	File Select Register
05h	RA	Port RA data register
06h	RB	Port RB data register
07h	RC	Port RC data register
08h	RD	Port RD data register
09h	RE	Port RE data register

4.1. PC Register (02h)

The PC register holds the lower eight bits of the program counter. It is accessible at run time to perform branch operations. The upper three bits are located in the STATUS register (PA2:0), bit 8 is not accessible.

4.2. STATUS Register (03h)

The STATUS register holds the arithmetic status of the ALU, the page select bits, and the reset state. The STATUS register is accessible during run time, except that bits PD and TO are read-only. It is recommended that only SETB and CLRB instructions be used on this register. Care should be exercised when writing to the STATUS register as the ALU status bits are updated upon completion of the write operation, possibly leaving the STATUS register with a result that is different than intended.

PA2	PA1	PA0	TO	PD	Z	DC	C
Bit 7							**Bit 0**

Bit 7-5: Program memory page select bits PA2:PA0

000 = Page 0 (000h - 1FFh)

001 = Page 1 (200h - 3FFh)

...

111 = Page 7 (E00h - FFFh)

Bit 4: Time Out bit, TO (Read Only)

1 = Set to 1 after power up and upon execution of CLRWDT or SLEEP instructions

0 = A watchdog time-out occurred

Bit 3: Power Down bit, PD (Read Only)

1= Set to a 1 after power up and upon execution of the CLR !WDT instruction

0 = Cleared to a '0' upon execution of SLEEP instruction

Bit 2: Zero bit, Z (affected by most logical, arithmetic, and data movement instructions

1 = Result of math operation is zero

0 = Result of math operation is non-zero Bit 1: Digit Carry bit, DC

After Addition:

1 = A carry from bit 3 occurred

0 = No carry from bit 3 occurred

After Subtraction:

1 = No borrow from bit 3 occurred

0 = A borrow from bit 3 occurred

Bit 0: Carry bit, C

After Addition:

1 = A carry from bit 7 of the result occurred

0 = No carry from bit 7 of the result occurred.

After Subtraction:

1 = No borrow from bit 7 of the result occurred

0 = A borrow from bit 7 of the result occurred

Rotate (RR or RL) Instructions:

The carry bit is loaded with the low or high order bit, respectively

When CF bit of the FUSEX register is cleared to 0, Carry bit works as input for ADD and SUB instructions.

4.3. OPTION Register

RTW	RTW_IE	RTS	RTE_ES	PSA	PS2	PS1	PS0
Bit 7							**Bit 0**

Bit 7: RTW RTCC/W register selection:
0 = Register 01h addresses W
1 = Register 01h addresses RTCC

Bit 6: RTE_IE RTCC interrupt enable:
0 = RTCC roll-over interrupt is enabled
1 = RTCC roll-over interrupt is disabled

Bit 5: RTS RTCC increment select:
0 = RTCC increments on internal instruction cycle
1 = RTCC increments upon transition on RTCC pin

Bit 4: RTE_ES RTCC edge select:
0 = RTCC increments on low-to-high transitions
1 = RTCC increments on high-to-low transitions

Bit 3: PSA Prescaler Assignment:
0 = Prescaler is assigned to RTCC, with divide rate determined by PS0 PS2 bits
1 = Prescaler is assigned to WDT, and divide rate on RTCC is 1:1

Bits 2-0: PS2-PS0 Prescaler divider (see Table 4-2)

Upon reset, all bits in the OPTION register are set to 1.

Table 4-2: Prescaler Divider Ratios

PS2, PS1, PS0	RTCC Divide Rate	Watchdog Timer Divide Rate
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

5.0 DEVICE CONFIGURATION AND ID REGISTERS

The SX device has two registers (FUSE, FUSEX) that control functions such as clock oscillator configuration. These registers are not programmable "on the fly" during normal device operation. Instead, the FUSE and FUSEX registers can only be accessed when the SX device is being programmed. The DEVICE ID register is a read only, hard-wired register, defined during the manufacturing process. Locations 1000h to 100Fh are allocated for user code ID.

5.1. FUSE Word (Read/program via Programming Command)

Unused	$\overline{\text{SYNC}}$	Unused	Unused	$\overline{\text{IRC}}$	DIV1/$\overline{\text{IFBD}}$	DIV0/FOSC2	XTLBUF_EN	$\overline{\text{CP}}$	WDTE	FOSC1	FOSC0
11	10	9	8	7	6	5	4	3	2	1	0

$\overline{\text{SYNC}}$ Synchronous input enable (this bit synchronizes the signal presented at the input pins to the internalclock through two internal flip-flops). Required to be enabled unless the transition on the input pin isnot close to the clock edge. If enabled, port data must be read more than 2 cycles after a change tothe input level mode or Schmitt Trigger mode (see Figure 3-2). SYNC is always enabled on RTCC.

0 = enabled

1 = disabled

$\overline{\text{IRC}}$ Internal RC oscillator enable

0 = enabled - OSC1 is pulled low by weak pulldown, OSC2 is pulled high by weak pullup

1 = disabled - OSC1 and OSC2 behave according to FOSC2:FOSC0

DIV1:DIV0 Internal RC oscillator divider (if $\overline{\text{IRC}}$ = 0)

00b = 4 MHz

01b = 1 MHz

10b = 128 KHz

11b = 32 KHz

$\overline{\text{IFBD}}$ Internal crystal/resonator oscillator feedback resistor (10MΩ)

0 = Internal feedback resistor disable (external feedback required for crystal/resonator oscillator)

1 = Internal feedback resistor enabled (valid only when $\overline{\text{IRC}}$ = 1, disabled when $\overline{\text{IRC}}$= 0)

XTLBUF_EN Crystal Buffer enable (disable when not using a crystal to reduce Idd)

0 = Crystal Buffer disabled (required if not using crystal/resonator oscillator)

1 = Crystal Buffer enabled

$\overline{\text{CP}}$ Code protect enable

0 = enabled (FUSE, code, and ID memories read back as scrambled data, programming disabled)

1 = disabled (FUSE, code, and ID memories can be read normally)

WDTE Watchdog timer enable

0 = disabled

1 = enabled

FOSC2:FOSC0 External oscillator configuration (valid when $\overline{\text{IRC}}$ = 1, lower settings are recommended for lower power consumption):

000b = LP1 - low power crystal (32KHz)

001b = LP2 - low power crystal/resonator (32KHz - 1MHz)

010b = XT1 - normal crystal/resonator (32KHz - 1MHz)

011b = XT2 - normal crystal/resonator (1MHz - 8MHz)

100b = HS1 - high speed crystal/resonator (1MHz - 20MHz)

101b = HS2 - high speed crystal/resonator (1MHz - 50MHz)

110b = HS3 - high speed crystal/resonator (1MHz - 75MHz)

111b = External RC network - OSC2 is pulled high by a weak pullup (no CLKOUT output)

5.2. FUSEX Word (Read/program via Programming Command)

IRCTRIM2	$\overline{\text{SLEEPCLK}}$	IRCTRIM1:IRCTRIM0	Unused	$\overline{\text{CF}}$	BOR1:BOR0	BORTR1:BORTR0	DRT1:DRT0
11	**10**	**9 8**	**7**	**6**	**5 4**	**3 2**	**1 0**

IRCTRIM2: IRCTRIM0 — Internal RC Oscillator Trim. This 3-bit field adjusts the operation of the internal RC oscillator to make it operate within the target frequency range of typically 4.0 MHz. Parts are shipped from the factory untrimmed. The device relies on the programming tool to provide trimming.

100b = maximum frequency

111b = typical

011b = minimum frequency

$\overline{\text{SLEEPCLK}}$ — Sleep Clock Disable.

0 = enable operation of the crystal/resonator clock during power down mode (to allow fast start up).

1 = disable crystal/resonator clock operation during power down mode (to reduce power consumption).

$\overline{\text{CF}}$ — Carry Flag ADD/SUB enable

0 = carry bit input to ADD and SUB instructions.

1 = ADD and SUB without carry

BOR1: BOR0 — Sets the Brown Out Reset threshold voltage

00b = 4.2 V

01b = 2.6 V

10b = 2.2 V

11b = BOR disabled

BORTR1: BORTR0 — Brown-Out trim bits (parts are shipped out of factory untrimmed).

01b = minimum threshold voltage

00b = LOW

11b = HIGH

10b = maximum threshold voltage

DRT1:DRT0 — Delay Reset Timer (DRT) timeout period. Specifies the time from de-assertion of reset to start code execution.

10b = 0.60 µsec

11b = 18 msec

00b = 60 msec

01b = 960 msec

5.3. DEVICE ID Word

(Hard-Wired Read-Only Via Programming Command) - Part ID Code

0	0	0	0	0	0	0	0	0	0	1	0
11	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**

5.4. User Code ID

Locations 1000h to 100Fh are allocated for user code ID.

6.0 MEMORY ORGANIZATION

6.1. Program Memory

The program memory is organized as 4K, 12-bit wide words. The program memory words are addressed sequentially by a binary program counter. Upon reset, the program counter is initialized with 0FFFh. If there is no branch operation, it will increment to the maximum value possible for the device and roll over and begin again. Internally, the program memory has a semi-transparent page structure. A page is composed of 512 contiguous program memory words. The lower nine bits of the program counter are zeros at the first address of a page and ones at the last address of a page. This page structure has no effect on the program counter. The program counter will freely increment through the page boundaries.

6.1.1. Program Counter

The program counter contains the 12-bit address of the instruction to be executed. The lower eight bits of the program counter are contained in the PC register (02h), and the three upper bits are specified by the STATUS register (PA0, PA1, PA2). Bit 8 is not accessible. Changing the STATUS bits is necessary to cause jumps and subroutine calls across program memory page boundaries. Prior to the execution of a branch operation, the user program must initialize the upper bits of the STATUS register to cause a branch to the desired page. An alternative method is to use the PAGE instruction, which automatically causes subsequent branch instructions to vector to the desired page, based on the value specified in the operand field.

6.1.2. Subroutine Stack

The subroutine stack consists of eight 12-bit save registers. A physical transfer of register contents from the program counter to the stack or vice versa, and within the stack, occurs on all operations affecting the stack, primarily calls and returns. The stack is physically and logically separate from data RAM. The program cannot read or write the stack.

6.2. Data Memory

The data memory is a RAM-based register set consisting of 262 general-purpose registers and nine special-purpose registers. All of these registers are eight bits wide. The data memory is organized into 16 banks, designated Bank 0 through Bank F, each containing 16 registers, plus an additional bank of 16 “global” registers. Because the registers are organized into banks or “files,” these memory-mapped registers are called “file registers.”

6.2.1. Addressing Modes/FSR

Each SX instruction that accesses a data memory register contains a 5-bit field in the instruction opcode that specifies the register to be accessed. The abbreviation “fr” (file register) represents the 5-bit register address designator. For example, the instruction description “mov fr,W” means that a 5-bit value or label must be substituted for “fr” in the instruction, such as “mov $0F,W” (to move the contents of the working register W into file reg ister 0Fh).

There are three different addressing modes, called the indirect, direct, and semi-direct modes. The addressing mode used for register access depends on the 5-bit “fr” value used in the instruction:

- indirect mode: fr = 00h
- direct mode (fr bit 4 = 0): fr = 01h through 0Fh
- semi-direct mode (fr bit 4 = 1): fr = 10h through 1Fh

Figure 6-1 illustrates the data memory addressing scheme. For indirect addressing (fr=00), the File Select Register (FSR) specifies the register to be accessed. FSR is an 8bit, memory-mapped register (at address 04h) which serves as an 8-bit pointer into data memory for indirect addressing. In this mode, the global register bank and Bank 1 through Bank F are accessible. Bank 0 is not accessible.

For direct addressing (fr=01-0F), the value of “fr” itself specifies the register to be accessed, and the FSR register is ignored. For this addressing mode, only the global register bank is accessible. To gain access to any other bank, you must use either indirect or semi-direct addressing.

For semi-direct addressing (fr=10-1F), the bank number is selected by the four high-order bits of FSR, and the register within that bank is selected by the four low-order bits of “fr.” In other words, the register address is obtained by combining the four high-order bits of FSR with the four low-order bits of “fr”. In this addressing mode, the low-order bits of FSR are ignored. Bank 0 through Bank F are accessible, but the global register bank is not accessible.

Figure 6-1 shows how register addressing works in the indirect, direct, and semi-direct modes. The 16 global registers are always accessible by direct addressing, regardless of what is contained in the FSR register. The global registers are also accessible with indirect addressing, but they are not accessible with semi-direct addressing. Of the 16 global registers, nine are special-purpose registers (RTCC, PC, STATUS, and so on), and six are general-purpose registers. Location 00 is used for indirect addressing (INDF). All of the registers in Bank 0 though Bank F are general-purpose registers. To change the contents of the FSR register, the program can either write an eight-bit value to the FSR register or use the “bank” instruction. The “bank” instruction writes bits 4, 5, and 6 in the FSR register. Bit 7 of FSR is used to select the upper or lower “bank” of memory banks. Thus, to change from one upper bank to another, only a single “bank” instruction is required. To change from one upper bank to a lower bank, the “bank” instruction must be followed by “setb FSR.7”.

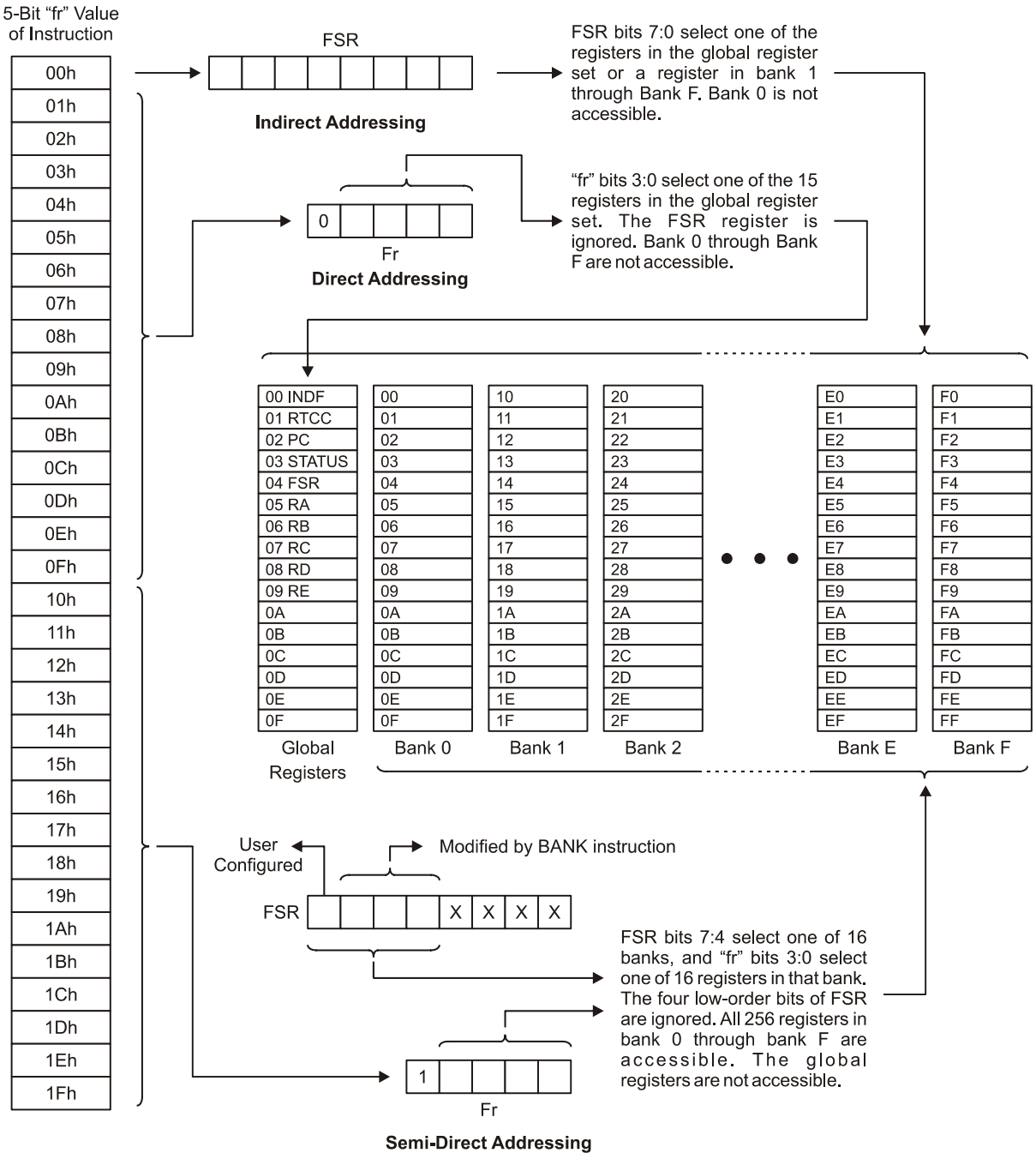


Figure 6-1: Register Access Modes

6.2.2. Register Access Examples

Here is an example of an instruction that uses direct addressing:

```
inc     $0F    ;increment file register 0Fh
```

This instruction increments the contents of file register 0Fh in the global register bank. It does not matter what is contained in the FSR register.

To gain access to any register outside of the global register bank, it is necessary to use semi-direct or indirect addressing. In that case, you need to make sure that the FSR register contains the correct value for accessing the desired bank. Here are 2 examples that use semi-direct addressing:

```
mov     W,#$F0  ;load W with F0h
mov     FSR,W   ;load W into FSR (Bank F)
inc     $1F     ;increment file register FFh
```

Or, to access bank 0,

```
mov     W,#$00  ;load W with 00h
mov     FSR,W   ;load W into FSR (Bank 0)
inc     $1F     ;increment file register 0Fh
```

In these examples, "FSR" is a label that represents the value 04h, which is the address of the FSR register in the global register bank. Note that the FSR register is itself a memory-mapped global register, which is always accessible using direct addressing.

The "banked" data memory is divided into upper and lower blocks, each consisting of 8 banks of data memory. The range for the lower block is from $00 to $7F, while the range for the upper block is from $80 to $FF. Bit 7 of the FSR is used to select the upper or lower block. The BANK instruction is used to select the bank within that block.

To use the "bank" instruction, in the syntax of the assembly language, you specify an 8-bit value that corresponds to the desired bank number. The assembler encodes bits 4, 5, and 6 of the specified value into the instruction opcode and ignores bit 7 and the low-order bits. For example, if another lower bank was being used to increment file register 2Fh, you could use the following instructions:

```
bank    $20     ;select Bank 2 in FSR
inc     $1F     ;increment register 2F
```

Note that the "bank" instruction only modifies bits 4, 5, and 6 the FSR register. Therefore, to change from a lower block to an upper block bank, the "bank" instruction will not work. Instead, you need to write the whole FSR register using code such as the following:

```
mov     W,#$80  ;load W with 80h
mov     FSR,W   ;select Bank 8 in FSR
```

Another approach is to set bit 7 of the FSR register individually after the "bank" instruction to address an upper block bank.

```
bank    $80     ;set bits in 4, 5, and 6 FSR
setb    FSR.7   ;select Bank 8 in FSR
```

To change from an upper block to a lower block bank, bit 7 of FSR must be cleared.

With indirect addressing, you specify the full 8-bit address of the register using FSR as a pointer. This addressing mode provides the flexibility to access different registers or multiple registers using the same instruction in the program.

You invoke indirect addressing by using fr=00h. For example:

```
mov     W,#$F5  ;load W with F5h
mov     $04,W   ;move value F5h into FSR
mov     W,#$01  ;load W with 01h
mov     $00,W   ;move value 01h into register F5h
```

In the second "mov" instruction, FSR is loaded with the desired 8-bit register address. In the fourth "mov" instruction, fr = 00, so the device looks at FSR and moves the result to the register addressed by FSR, which is the register at F5h (Bank F, register number 5).

A practical example that uses indirect addressing is the following program, which clears the upper eight registers in the global register bank and the upper 8 registers in all banks from Bank 1 through Bank F:

```
clr     FSR     ;clear FSR to 00h (at 04h)
:loop   setb    FSR.3   ;set FSR bit 3
clr     $00     ;clear register pointed to by FSR
incsz   FSR     ;increment FSR and test
                ;skip jmp if 00h
jmp     :loop   ;jump back and clear next reg.
```

This program initially clears FSR to 00h. At the beginning of the loop, it sets bit 3 of FSR so that it starts at 08h. The "clr $00" instruction clears the register pointed to by FSR (initially, the file register at 08h in the global register bank). Then the program increments FSR and clears consecutive file registers, always in the upper half of each bank: (08h, 09h, 0Ah... 0Fh, 18h, 19h... FFh). The loop ends when FSR wraps back to 00h.

For addresses from 01h through 0Fh, the global register bank is accessed. For higher addresses, Bank 1 through Bank F are accessed. This program does not affect Bank 0, which is not accessible in the indirect addressing mode.

7.0 POWER DOWN MODE

The power down mode is entered by executing the SLEEP instruction.

In power down mode, only the Watchdog Timer (WDT) and SLEEPCLOCK are active, if enabled. The operation clock can be enabled or disabled during this mode, by using the SLEEPCLK bit of the FUSEX register. If the Watchdog Timer is enabled, upon execution of the SLEEP instruction, the Watchdog Timer is cleared, the TO (time out) bit is set in the STATUS register, and the PD (power down) bit is cleared in the STATUS register.

There are three different ways to exit from the power down mode:

1. A timer overflow signal from the Watchdog Timer (WDT).
2. A valid transition on any of the Multi-Input Wakeup pins (Port B pins).
3. An external reset input on the MCLR pin.

The states of registers (upon wakeup) are described in Section 15.0.

To achieve the lowest possible power consumption, the Watchdog Timer should be disabled (the sleep clock should be disabled) and the device should exit the power down mode through the (Multi-Input Wakeup) MIWU pins or an external reset. In addition, the SLEEPCLOCK should be disabled during the power down mode.

Bit 11 of the FUSEX can be used to enable (clear bit to 0) the clock operation during the power down mode (to allow fast clock start-up upon exiting the power down mode).

7.1. Multi-Input Wakeup

Multi-Input Wakeup is one way of causing the device to exit the power down mode. Port B is used to support this feature. The WKEN_B register (Wakeup Enable Register) allows any Port B pin or combination of pins to cause the wakeup. Clearing a bit in the WKEN_B register enables the wakeup on the corresponding Port B pin. If multi-input wakeup is selected to cause a wakeup, the trigger condition on the selected pin can be either rising edge (low to high) or falling edge (high to low). The WKED_B register (Wakeup Edge Select) selects the desired transition edge. Setting a bit in the WKED_B register selects the falling edge on the corresponding Port B. Resetting the bit selects the rising edge. The WKEN_B and WKED_B registers are set to FFh upon reset.

Once a valid transition occurs on the selected pin, the WKPND_B register (Wakeup Pending Register) latches the transition in the corresponding bit position. A logic ‘1’ indicates the occurrence of the selected trigger edge on the corresponding Port B pin. The WKPND_B comes up with undefined value upon reset. The user program must clear the WKPND_B register prior to enabling the interrupt.

Upon exiting the power down mode, the Multi-Input Wakeup logic causes program counter to branch to the maximum program memory address (same as reset).

Figure 7-1 shows the Multi-Input Wakeup block diagram.

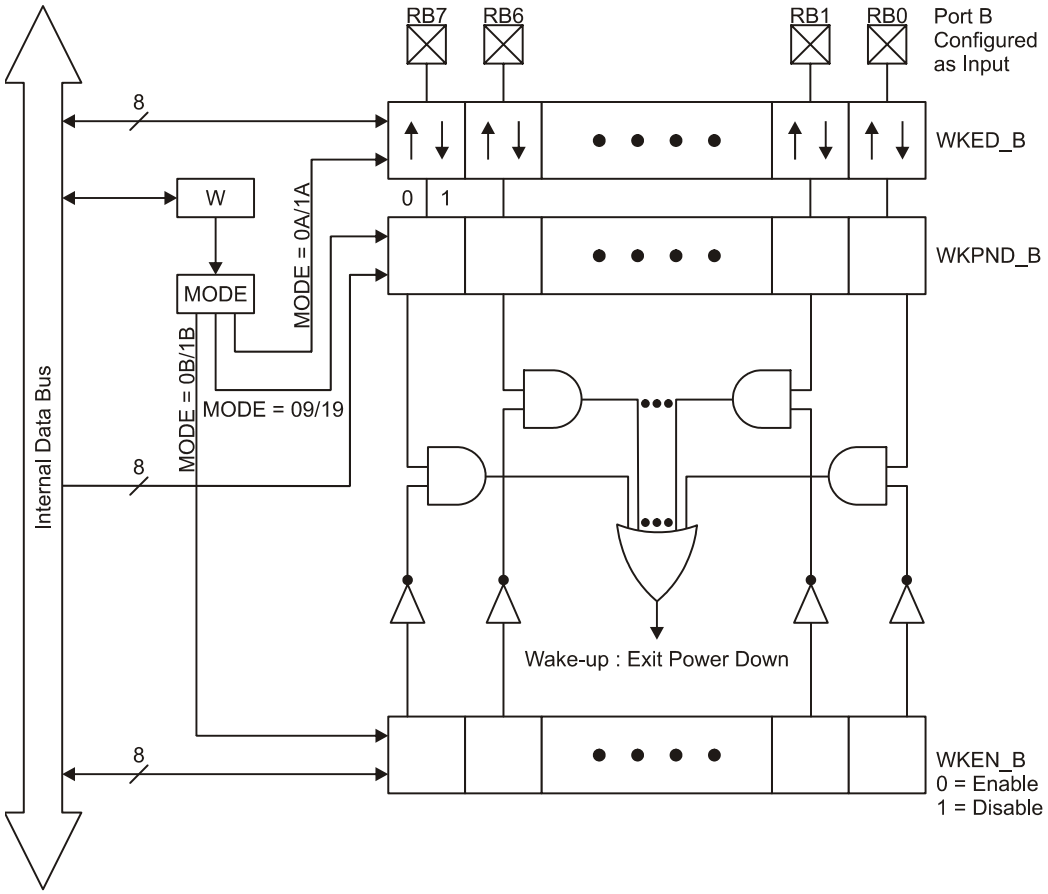


Figure 7-1: Multi-Input Wakeup Block Diagram

7.2. Port B MIWU/Interrupt Configuration

The WKPND_B register comes up with an unknown value upon reset. The user program must clear the register prior to enabling the wake-up condition or interrupts. The proper initialization sequence is:

Select the desired edge (through WKED_B register).

Clear the WKPND_B register.

Enable the Wakeup condition (through WKEN_B register).

Below is an example of how to read the WKPND_B register to determine which Port B pin caused the wakeup or interrupt, and to clear the WKPND_B register:

```
mov     W, #$19 ;prepare to exchange WKPND_B
                ;with W (can also use $09)
        mov M,W
        clr W
mov     !RB,W   ;W contains WKPND_B
                ;contents of W exchanged
                ;with contents of WKPND_B
```

The final "mov" instruction in this example performs an exchange of data between the working register (W) and the WKPND_B register. This exchange occurs only with accesses to the WKPND_B and CMP_B registers. Otherwise, the "mov" instruction does not perform an exchange, but only moves data from the source to the destination. Here is an example of a program segment that configures the RB0, RB1, and RB2 pins to operate as Multi-Input Wakeup/Interrupt pins, sensitive to falling edges:

```
mov     W,#$1F  ;prepare to write port data
mov     M,W     ;direction registers
mov     W,#$07  ;load W with the value 07h
mov     RB,W    ;configure RB0-RB2 to be inputs

mov     W,#$1A  ;prepare to write WKED_B
mov     M,W     ;(edge) register
mov     W,#$07  ;load W with the value 07h
mov     RB,W    ;configure RB0-RB2 to sense
                ;falling edges

mov     W,#$19  ;prepare to access WKPND_B
mov     M,W     ;(pending) register
mov     W,#$00  ;clear W
mov     !RB,W   ;clear all wakeup pending flags

mov     W,#$1B  ;prepare to write WKEN_B (enable)
mov     M,W     ;register
mov     W,#$F8  ;load W with the value F8h
mov     !RB,W   ;enable RB0-RB2 to operate as
                ;wakeup inputs
```

To prevent false interrupts, the enabling step (clearing bits in WKEN_B) should be done as the last step in a sequence of Port B configuration steps.

After this program segment is executed, the device can receive interrupts on the RB0, RB1, and RB2 pins. If the device is put into the power down mode (by executing a SLEEP instruction), the device can then receive wakeup signals on those same pins.

8.0 INTERRUPT SUPPORT

The device supports both internal and external maskable interrupts. The internal interrupt is generated as a result of the RTCC rolling over from FFh to 00h. This interrupt source has an associated enable bit located in the OPTION register and pending flag bit in the Timer T1 Control B register. In addition, timers T1 and T2 each have three interrupt sources associated with counter overflow, compare match, and input capture.

Port B provides the source for eight external software selectable, edge sensitive interrupts, when the device is not in the power down mode. These interrupt sources share logic with the Multi-Input Wakeup circuitry. The WKEN_B register allows interrupt from Port B to be individually enabled or disabled.

Clearing a bit in the WKEN_B register enables the interrupt on the corresponding Port B pin. The WKED_B selects the transition edge to be either positive or negative. The WKEN_B and WKED_B registers are set to FFh upon reset. Setting a bit in the WKED_B register selects the falling edge while clearing the bit selects the rising edge on the corresponding Port B pin.

The WKPND_B register serves as the external interrupt pending register.

The WKPND_B register comes up with a random value upon reset. The user program must clear the WKPND_B register prior to enabling the interrupt.

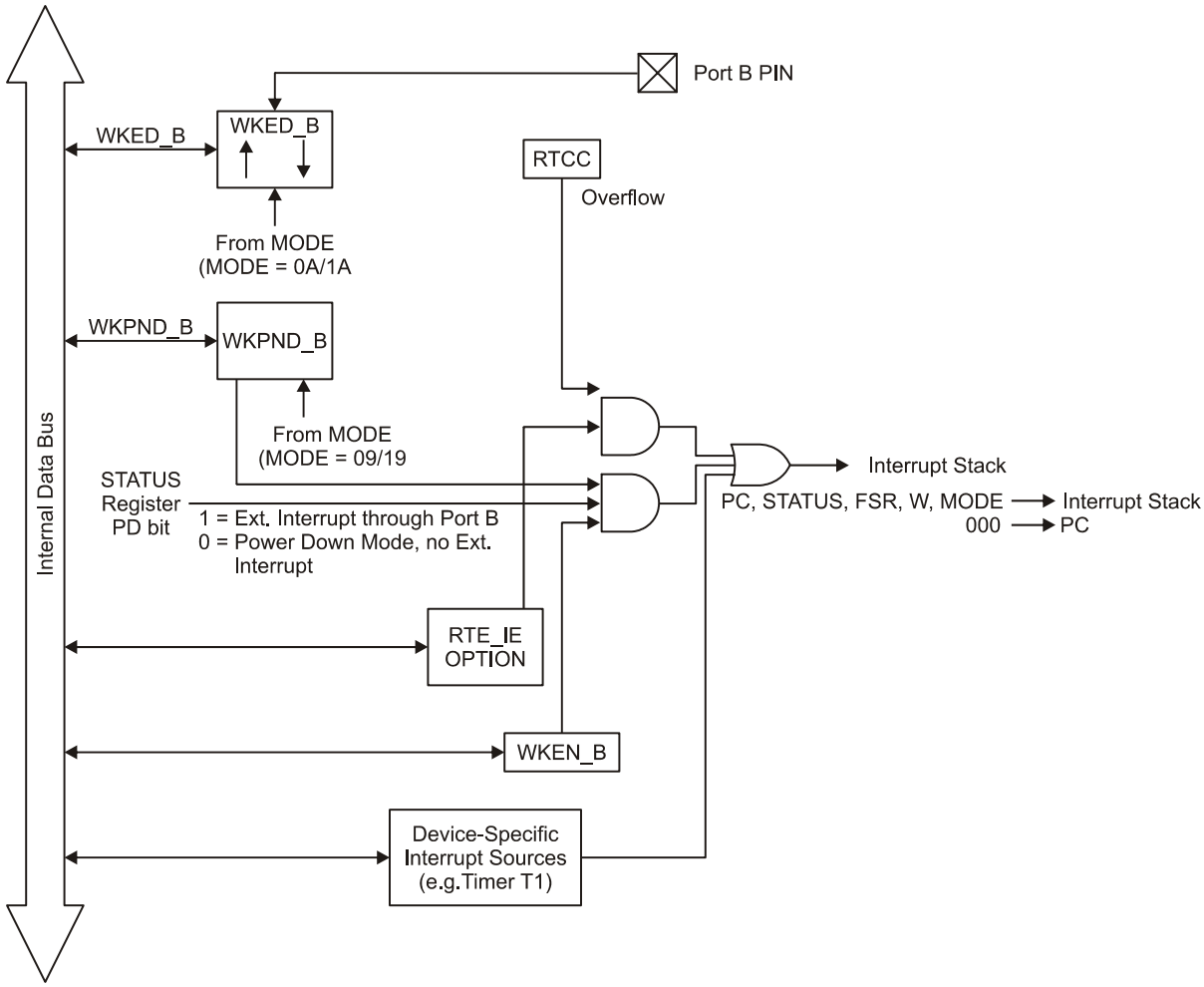


Figure 8-1: Interrupt Structure

All interrupts are global in nature; that is, no interrupt has priority over another. Interrupts are handled sequentially. Figure 8-2 shows the interrupt processing sequence. Once an interrupt is acknowledged, all subsequent interrupts are disabled until return from servicing the current interrupt. The PC is pushed onto the single level interrupt stack, and the contents of the FSR, STATUS, MODE, and W registers are saved in their corresponding shadow registers. The status bits PA2, PA1, and PA0 are cleared after STATUS has been saved in its shadow register. The interrupt logic has its own single-level stack and is not part of the CALL subroutine stack. The vector for the interrupt service routine is address 0.

Once in the interrupt service routine, the user program must poll all interrupt pending bits to determine the source of the interrupt. The interrupt service routine should clear the corresponding interrupt pending flag. Normally it is a requirement for the user program to process every interrupt without missing any. To ensure this, the longest path through the interrupt routine must take less time than the shortest possible delay between interrupts.

Using more than one interrupt, such as multiple external interrupts or both RTCC and external interrupts, can result in missed or, at best, jittery interrupt handling should one occur during the processing of another. When handling external interrupts, the interrupt routine should clear at least one pending register bit. The bit that is cleared should represent the interrupt being handled in order for the next interrupt to trigger.

Upon return from the interrupt service routine, the contents of PC, FSR, STATUS, MODE, and W registers are restored from their corresponding shadow registers. The interrupt service routine should end with instructions such as RETI or RETIW. RETI pops the interrupt stack and the special shadow registers used for storing W, STATUS, MODE, and FSR (preserved during interrupt handling). RETIW behaves like RETI but also adds W to RTCC. The interrupt return instruction enables the global interrupts.

If a MIWU interrupt occurs during a pre-existing interrupt service routine, the MIWU interrupt flag is set immediately, and the MIWU interrupt is serviced upon completion of the pre-existing interrupt service routine.

Timer interrupt will occur only if not in the ISR when the interrupt occurs.

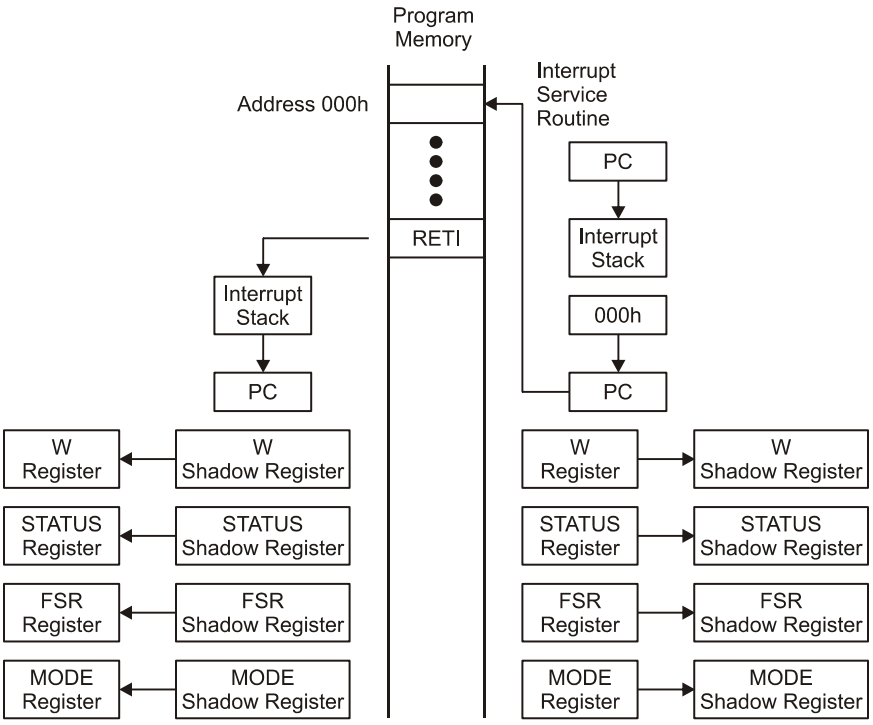


Figure 8-2: Interrupt Processing

Note: The interrupt logic has its own single-level stack and is not part of the CALL subroutine stack.