MICROCHIP

Section 9. I/O Ports

HIGHLIGHTS

This section of the manual contains the following major topics:

9.1 Introduction

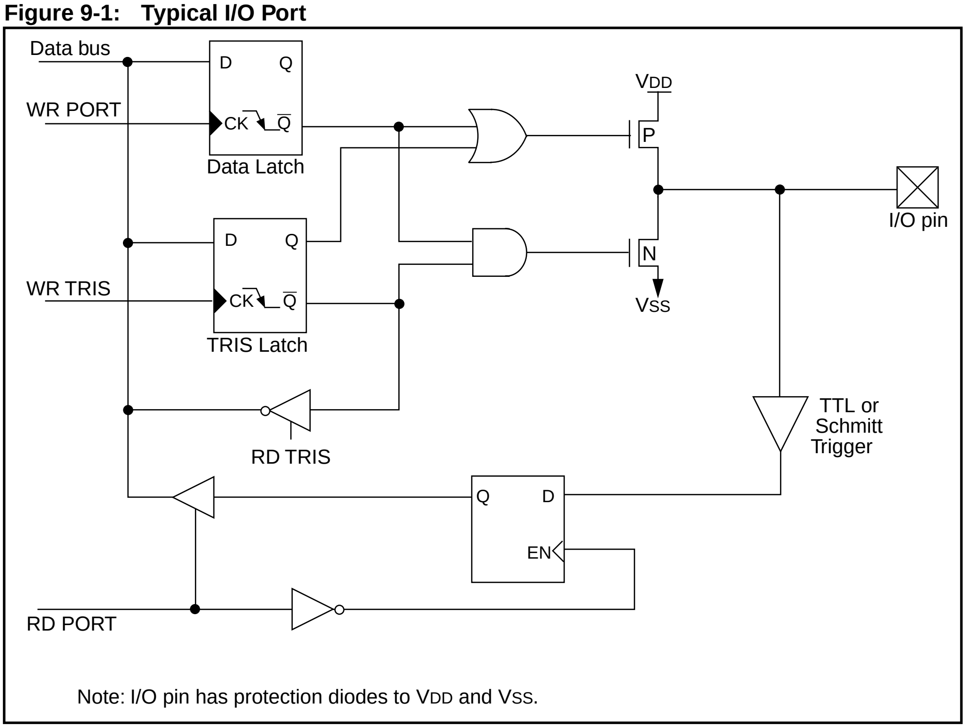
General purpose I/O pins can be considered the simplest of peripherals. They allow the PICmicro™ to monitor and control other devices. To add flexibility and functionality to a device, some pins are multiplexed with an alternate function(s). These functions depend on which peripheral features are on the device. In general, when a peripheral is functioning, that pin may not be used as a general purpose I/O pin.

For most ports, the I/O pin's direction (input or output) is controlled by the data direction register, called the TRIS register. TRIS<x> controls the direction of PORT<x>. A '1' in the TRIS bit corresponds to that pin being an input, while a '0' corresponds to that pin being an output. An easy way to remember is that a '1' looks like an I (input) and a '0' looks like an O (output).

The PORT register is the latch for the data to be output. When the PORT is read, the device reads the levels present on the I/O pins (not the latch). This means that care should be taken with read-modify-write commands on the ports and changing the direction of a pin from an input to an output.

Figure 9-1 shows a typical I/O port. This does not take into account peripheral functions that may be multiplexed onto the I/O pin. Reading the PORT register reads the status of the pins whereas writing to it will write to the port latch. All write operations (such as BSF and BCF instructions) are read-modify-write operations. Therefore a write to a port implies that the port pins are read, this value is modified, and then written to the port data latch.

Figure 9-1: Typical I/O Port



When peripheral functions are multiplexed onto general I/O pins, the functionality of the I/O pins may change to accommodate the requirements of the peripheral module. Examples of this are the Analog-to-Digital (A/D) converter and LCD driver modules, which force the I/O pin to the peripheral function when the device is reset. In the case of the A/D, this prevents the device from consuming excess current if any analog levels were on the A/D pins after a reset occurred.

With some peripherals, the TRIS bit is overridden while the peripheral is enabled. Therefore, read-modify-write instructions (`BSF, BCF, XORWF`) with TRIS as destination should be avoided. The user should refer to the corresponding peripheral section for the correct TRIS bit settings.

PORT pins may be multiplexed with analog inputs and analog V_{REF} input. The operation of each of these pins is selected, to be an analog input or digital I/O, by clearing/setting the control bits in the ADCON1 register (A/D Control Register1). When selected as an analog input, these pins will read as ‘0’s.

The TRIS registers control the direction of the port pins, even when they are being used as analog inputs. The user must ensure the TRIS bits are maintained set when using the pins as analog inputs.

Note 1: If pins are multiplexed with Analog inputs, then on a Power-on Reset these pins are configured as analog inputs, as controlled by the ADCON1 register. Reading port pins configured as analog inputs read a ‘0’.

Note 2: If pins are multiplexed with comparator inputs, then on a Power-on Reset these pins are configured as analog inputs, as controlled by the CMCON register. Reading port pins configured as analog inputs read a ‘0’.

Note 3: If pins are multiplexed with LCD driver segments, then on a Power-on Reset these pins are configured as LCD driver segments, as controlled by the LCDSE register. To configure the pins as a digital port, the corresponding bits in the LCDSE register must be cleared. Any bit set in the LCDSE register overrides any bit settings in the corresponding TRIS register.

Note 4: Pins may be multiplexed with the Parallel Slave Port (PSP). For the PSP to function, the I/O pins must be configured as digital inputs and the PSPMODE bit must be set.

Note 5: At present the Parallel Slave Port (PSP) is only multiplexed onto PORTD and PORTE. The microprocessor port becomes enabled when the PSPMODE bit is set. In this mode, the user must make sure that the TRISE bits are set (pins are configured as digital inputs) and that PORTE is configured for digital I/O. PORTD will override the values in the TRISD register. In this mode the PORTD and PORTE input buffers are TTL. The control bits for the PSP operation are located in TRISE.

9.2 PORTA and the TRISA Register

The RA4 pin is a Schmitt Trigger input and an open drain output. All other RA port pins have TTL input levels and full CMOS output drivers. All pins have data direction bits (TRIS registers) which can configure these pins as output or input.

Setting a TRISA register bit puts the corresponding output driver in a hi-impedance mode. Clearing a bit in the TRISA register puts the contents of the output latch on the selected pin(s).

Example 9-1: Initializing PORTA

```
CLRF   STATUS       ; Bank0
CLRF   PORTA        ; Initialize PORTA by clearing output
                    ;    data latches
BSF    STATUS, RP0  ; Select Bank1
MOVLW  0xCF         ; Value used to initialize data direction
MOVWF  TRISA        ; PORTA<3:0> = inputs PORTA<5:4> = outputs
                    ;    TRISA<7:6> always read as '0'
```

Figure 9-2: Block Diagram of RA3:RA0 and RA5 Pins

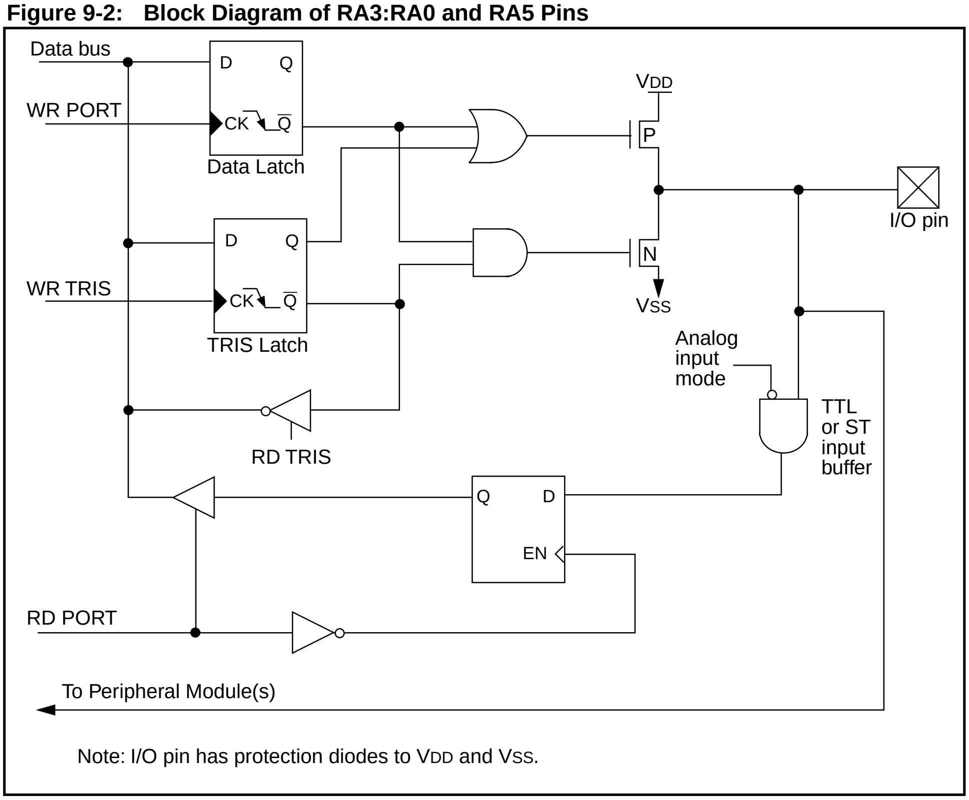
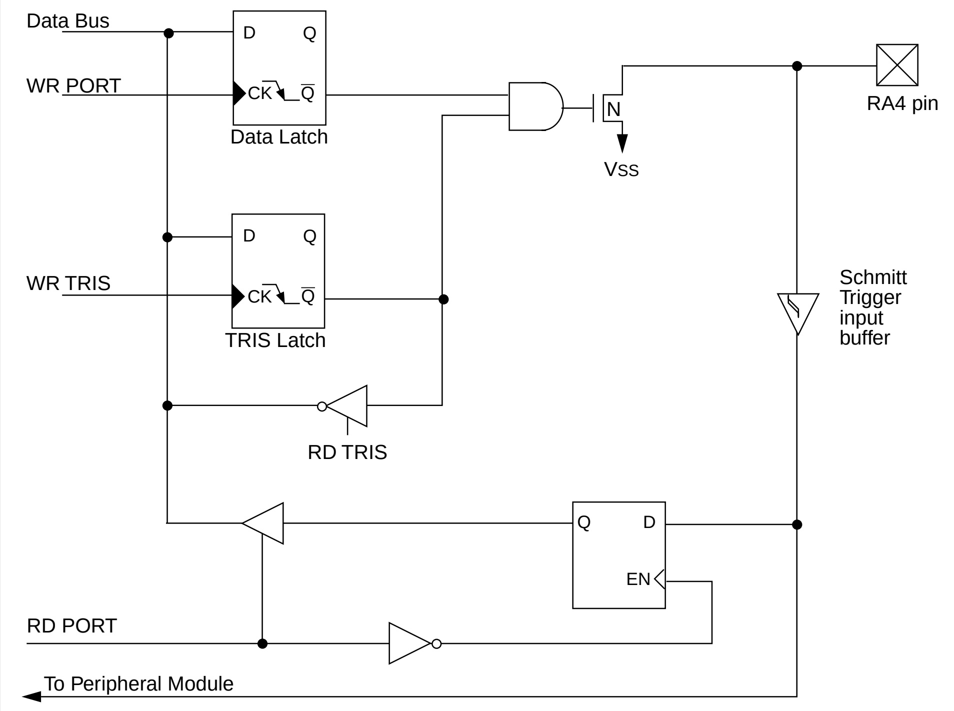


Figure 9-3: Block Diagram of RA4 Pin



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

9.3 PORTB and the TRISB Register

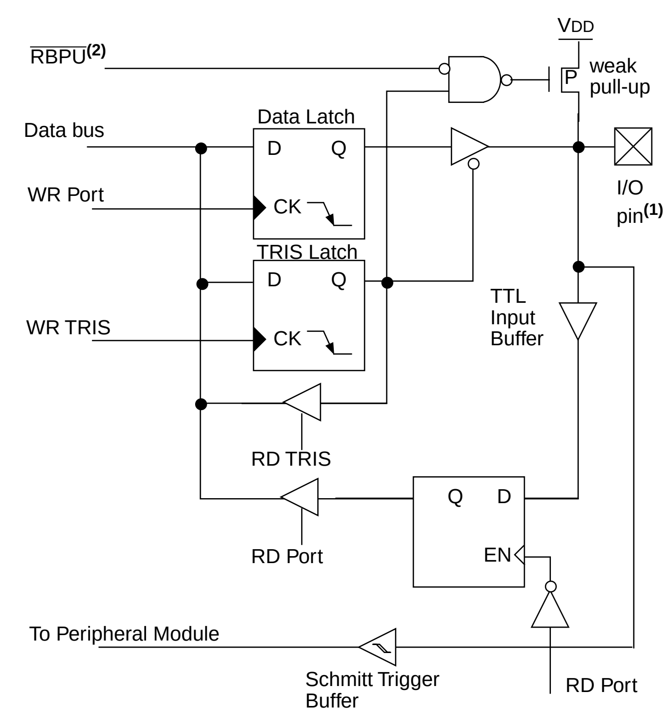
PORTB is an 8-bit wide bi-directional port. The corresponding data direction register is TRISB. Setting a bit in the TRISB register puts the corresponding output driver in a high-impedance input mode. Clearing a bit in the TRISB register puts the contents of the output latch on the selected pin(s).

Example 9-2: Initializing PORTB

```
CLRF    STATUS        ; Bank0
CLRF    PORTB         ; Initialize PORTB by clearing output
                      ;    data latches
BSF     STATUS, RP0   ; Select Bank1
MOVLW   0xCF          ; Value used to initialize data direction
MOVWF   TRISB         ; PORTB<3:0> = inputs, PORTB<5:4> = outputs
                      ;    PORTB<7:6> = inputs
```

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

Figure 9-4: Block Diagram of RB3:RB0 Pins



Note 1: I/O pins have diode protection to V_{DD} and V_{SS}.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the $\overline{\text{RBPU}}$ bit (OPTION<7>).

Four of PORTB's pins, RB7:RB4, have an interrupt on change feature. Only pins configured as inputs can cause this interrupt to occur (i.e. any RB7:RB4 pin configured as an output is excluded from the interrupt on change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB. The “mismatch” outputs of RB7:RB4 are OR'ed together to generate the RB Port Change Interrupt with flag bit RBIF (INTCON<0>).

This interrupt can wake the device from SLEEP. The user, in the interrupt service routine, can clear the interrupt in the following manner:

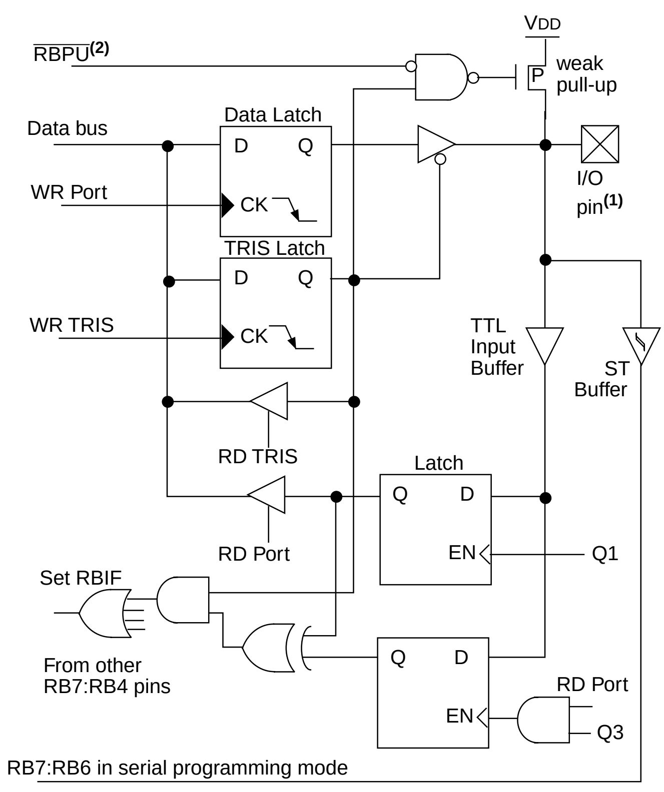
a) Any read or write of PORTB. This will end the mismatch condition.
b) Clear flag bit RBIF.

A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition, and allow flag bit RBIF to be cleared.

This interrupt on mismatch feature, together with software configurable pull-ups on these four pins allow easy interface to a keypad and make it possible for wake-up on key-depression.

The interrupt on change feature is recommended for wake-up on key depression and operations where PORTB is only used for the interrupt on change feature. Polling of PORTB is not recommended while using the interrupt on change feature.

Figure 9-5: Block Diagram of RB7:RB4 Pins



Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the $\overline{\text{RBPU}}$ bit (OPTION<7>).
3: In sleep mode the device is in Q1 state.

9

I/O Ports

9.4 PORTC and the TRISC Register

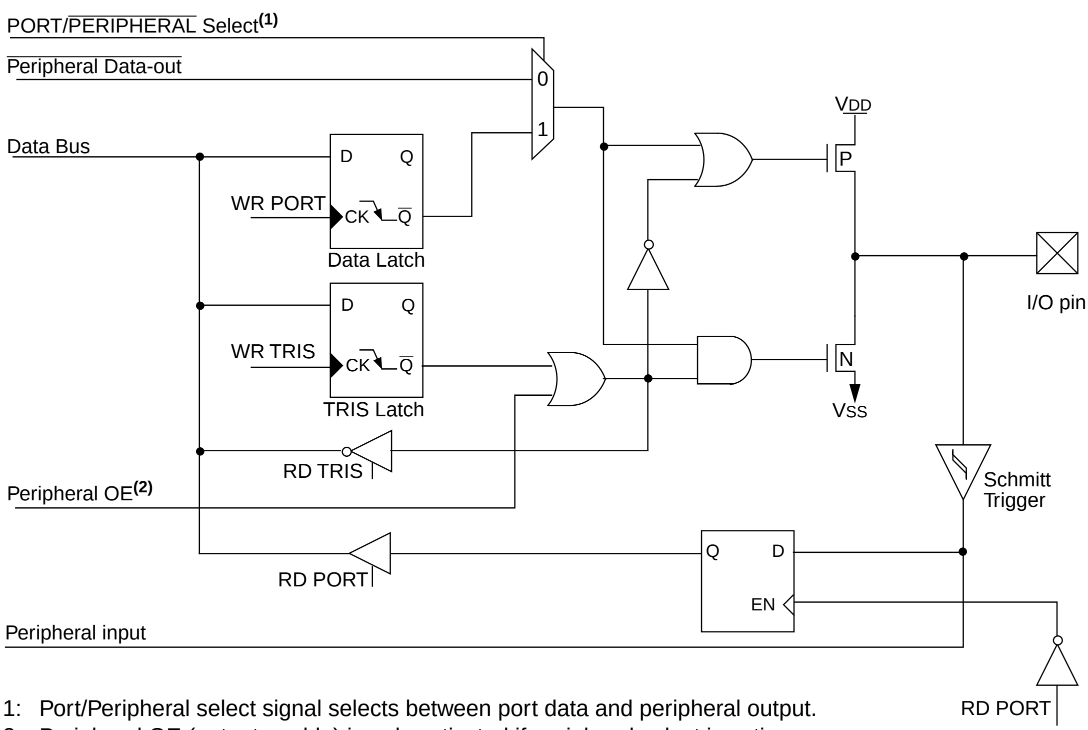
PORTC is an 8-bit bi-directional port. Each pin is individually configurable as an input or output through the TRISC register. PORTC pins have Schmitt Trigger input buffers.

When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin. Some peripherals override the TRIS bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input.

Example 9-3: Initializing PORTC

```
CLRF   STATUS       ; Bank0
CLRF   PORTC        ; Initialize PORTC by clearing output
                    ;    data latches
BSF    STATUS, RP0  ; Select Bank1
MOVLW  0xCF         ; Value used to initialize data direction
MOVWF  TRISC        ; PORTC<3:0> = inputs, PORTC<5:4> = outputs
                    ;    PORTC<7:6> = inputs
```

Figure 9-6: PORTC Block Diagram (Peripheral Output Override)



Note 1: Port/Peripheral select signal selects between port data and peripheral output.
2: Peripheral OE (output enable) is only activated if peripheral select is active.
3: I/O pins have diode protection to V_{DD} and V_{SS}.

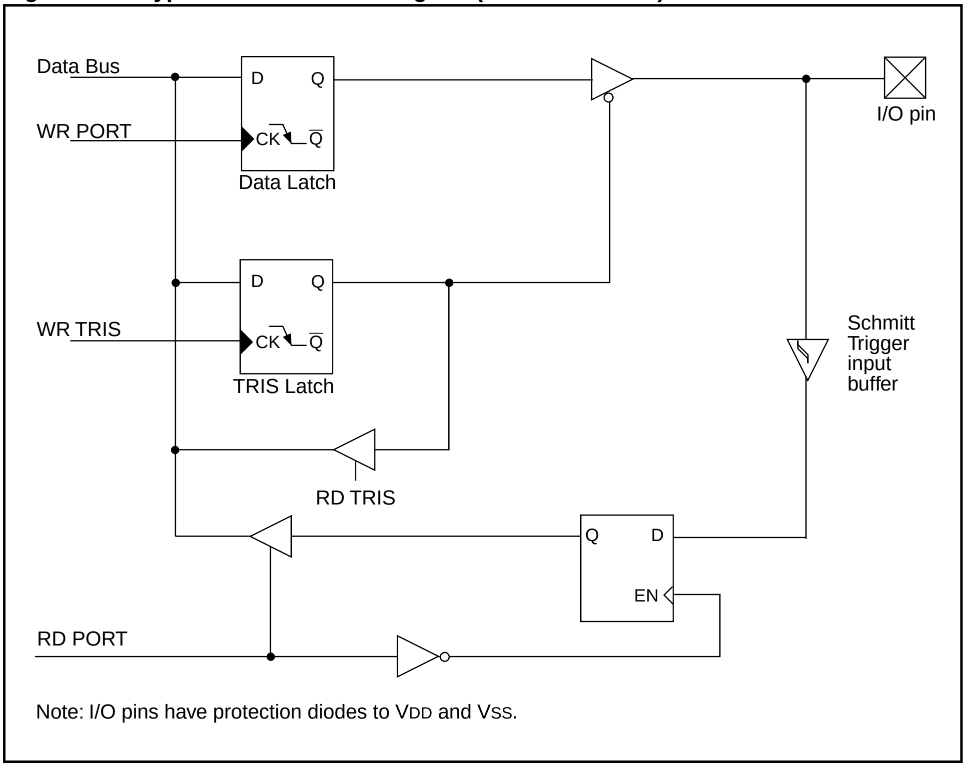
9.5 PORTD and the TRISD Register

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

Example 9-4: Initializing PORTD

```
CLRF   STATUS       ; Bank0
CLRF   PORTD        ; Initialize PORTD by clearing output
                    ;    data latches
BSF    STATUS, RP0  ; Select Bank1
MOVLW  0xCF         ; Value used to initialize data direction
MOVWF  TRISD        ; PORTD<3:0> = inputs, PORTD<5:4> = outputs
                    ;    PORTD<7:6> = inputs
```

Figure 9-7: Typical PORTD Block Diagram (in I/O Port Mode)



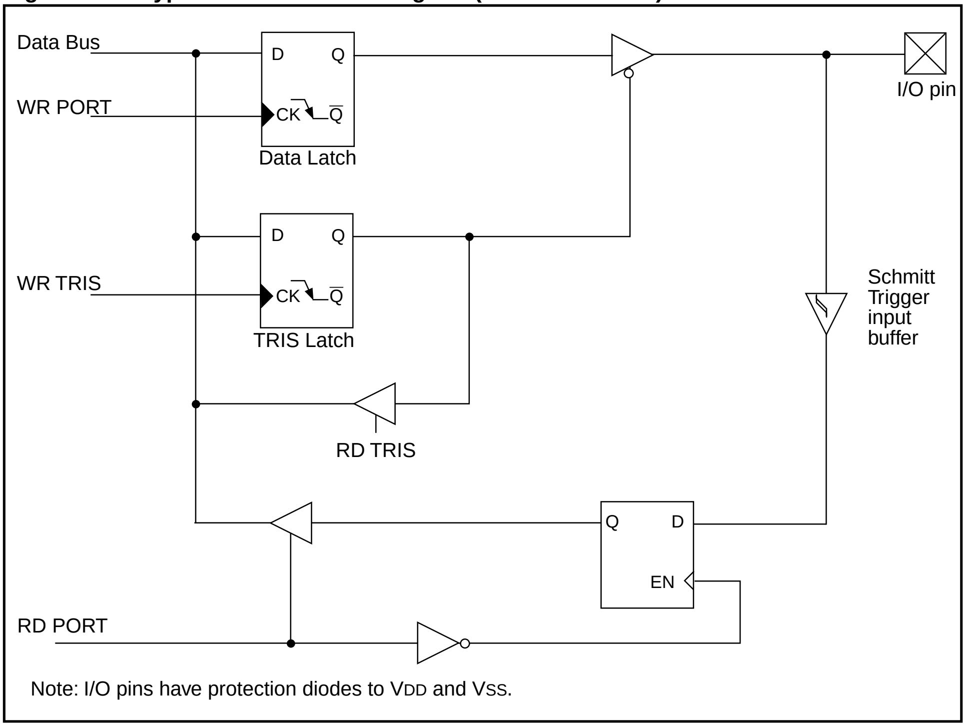
9.6 PORTE and the TRISE Register

PORTE can be up to an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output.

Example 9-5: Initializing PORTE

```
CLRF   STATUS       ; Bank0
CLRF   PORTE        ; Initialize PORTE by clearing output
                    ;    data latches
BSF    STATUS, RP0  ; Select Bank1
MOVLW  0x03         ; Value used to initialize data direction
MOVWF  TRISE        ; PORTE<1:0> = inputs, PORTE<7:2> = outputs
```

Figure 9-8: Typical PORTE Block Diagram (in I/O Port Mode)



Note:	On some devices with PORTE, the upper bits of the TRISE register are used for the Parallel Slave Port control and status bits.

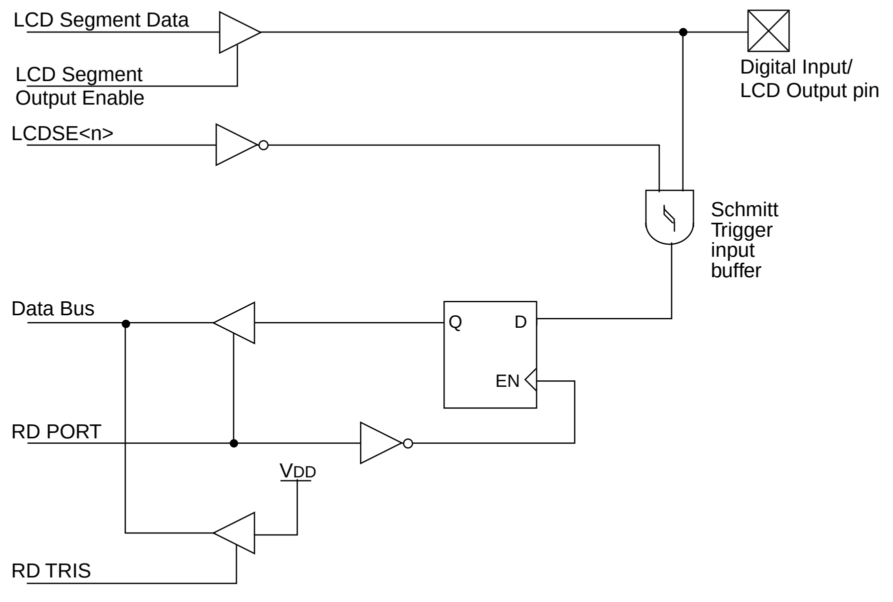
9.7 PORTF and the TRISF Register

PORTF is a digital input only port. Each pin is multiplexed with an LCD segment driver. These pins have Schmitt Trigger input buffers.

Example 9-6: Initializing PORTF

```
BCF   STATUS, RP0    ; Select Bank2
BSF   STATUS, RP1    ;
BCF   LCDSE, SE16    ; Make all PORTF
BCF   LCDSE, SE12    ;    digital inputs
```

Figure 9-9: PORTF LCD Block Diagram



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

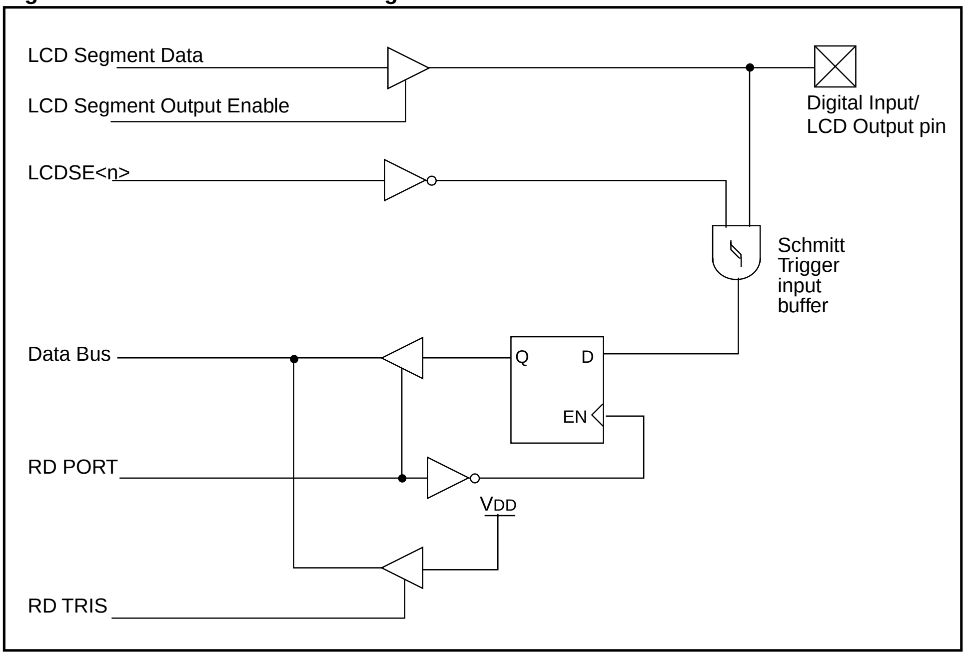
9.8 PORTG and the TRISG Register

PORTG is a digital input only port. Each pin is multiplexed with an LCD segment driver. These pins have Schmitt Trigger input buffers.

Example 9-7: Initializing PORTG

```
    BCF   STATUS, RP0    ; Select Bank2
    BSF   STATUS, RP1    ;
    BCF   LCDSE, SE27    ; Make all PORTG
    BCF   LCDSE, SE20    ;    and PORTE<7> digital inputs
```

Figure 9-10: PORTG LCD Block Diagram



9.9 GPIO and the TRISGP Register

GPIO is an 8-bit I/O register. Only the low order six bits are implemented (GP5:GP0). Bits 7 and 6 are unimplemented and read as ‘0’s. Any GPIO pin (except GP3) can be programmed individually as input or output. The GP3 pin is an input only pin.

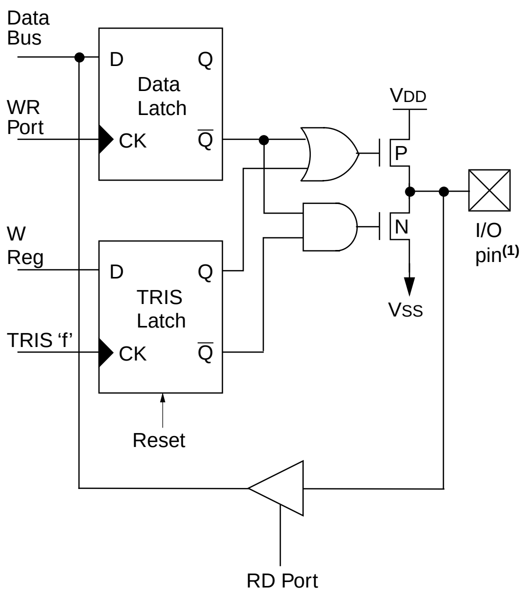
The TRISGP register controls the data direction for GPIO pins. A ‘1’ in a TRISGP register bit puts the corresponding output driver in a hi-impedance mode. A ‘0’ puts the contents of the output data latch on the selected pins, enabling the output buffer. The exceptions are GP3 which is input only and its TRIS bit will always read as '1'. Upon reset, the TRISGP register is all ‘1’s, making all pins inputs.

A read of the GPIO port, reads the pins not the output data latches. Any input must be present until read by an input instruction (e.g., `MOVF GPIO,W`). The outputs are latched and remain unchanged until the output latch is rewritten.

Example 9-8: Initializing GPIO

```
    CLRF   STATUS       ; Bank0
    CLRF   GPIO         ; Initialize GPIO by clearing output
                        ;   data latches
    BSF    STATUS, RP0  ; Select Bank1
    MOVLW  0xCF         ; Value used to initialize data direction
    MOVWF  TRISGP       ; GP<3:0> = inputs GP<5:4> = outputs
                        ;   TRISGP<7:6> always read as '0'
```

Figure 9-11: Block Diagram of GP5:GP0 Pins



Note 1: I/O pins have protection diodes to V_{DD} and V_{SS}.
GP3 is input only with no data latch and no output drivers.

The configuration word can set several I/O’s to alternate functions. When acting as alternate functions the pins will read as ‘0’ during port read. The GP0, GP1, and GP3 pins can be configured with weak pull-ups and also with interrupt on change. The interrupt on change and weak pull-up functions are not pin selectable. Interrupt on change is enabled by setting INTCON<3>. If the device configuration bits select one of the external oscillator modes, the GP4 and GP5 pin’s GPIO functions are overridden and these pins are used for the oscillator.

9.10 I/O Programming Considerations

When using the ports (and GPIO) as I/O, design considerations need to be taken into account to ensure that the operation is as intended.

9.10.1 Bi-directional I/O Ports

Any instruction which performs a write operation actually does a read followed by a write operation. The BCF and BSF instructions, for example, read the register into the CPU, execute the bit operation, and write the result back to the register. Caution must be used when these instructions are applied to a port with both inputs and outputs defined. For example, a BSF operation on bit5 of PORTB will cause all eight bits of PORTB to be read into the CPU. Then the BSF operation takes place on bit5 and PORTB is written to the output latches. If another bit of PORTB is used as a bi-directional I/O pin (e.g., bit0) and it is defined as an input at this time, the input signal present on the pin itself would be read into the CPU and rewritten to the data latch of this particular pin, overwriting the previous content. As long as the pin stays in the input mode, no problem occurs. However, if bit0 is switched to an output, the content of the data latch may now be unknown.

Reading the port register, reads the values of the port pins. Writing to the port register writes the value to the port latch. When using read-modify-write instructions (ex. BCF, BSF, etc.) on a port, the value of the port pins is read, the desired operation is performed on this value, and the value is then written to the port latch.

Example 9-9 shows the effect of two sequential read-modify-write instructions on an I/O port.

Example 9-9: Read-Modify-Write Instructions on an I/O Port

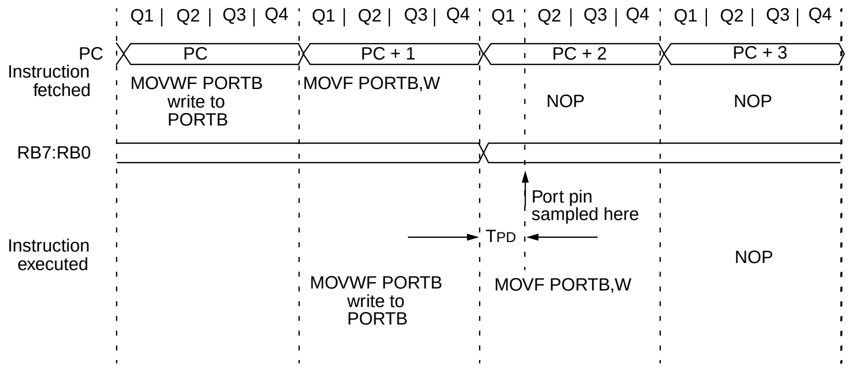
```
; Initial PORT settings: PORTB<7:4> Inputs
;                        PORTB<3:0> Outputs
; PORTB<7:6> have external pull-ups and are not connected to other circuitry
;
;                       PORT latch  PORT pins
;                       ----------  ---------
    BCF  PORTB, 7     ; 01pp pppp    11pp pppp
    BCF  PORTB, 6     ; 10pp pppp    11pp pppp
    BSF  STATUS, RP0  ;
    BCF  TRISB, 7     ; 10pp pppp    11pp pppp
    BCF  TRISB, 6     ; 10pp pppp    10pp pppp
;
; Note that the user may have expected the pin values to be 00pp ppp.
; The 2nd BCF caused RB7 to be latched as the pin value (high).
```

A pin configured as an output, actively driving a Low or High should not be driven from external devices at the same time in order to change the level on this pin (“wired-or,” “wired-and”). The resulting high output currents may damage the chip.

9.10.2 Successive Operations on an I/O Port

The actual write to an I/O port happens at the end of an instruction cycle, whereas for reading, the data must be valid at the beginning of the instruction cycle (Figure 9-12). Therefore, care must be exercised if a write followed by a read operation is carried out on the same I/O port. The sequence of instructions should be such to allow the pin voltage to stabilize (load dependent) before the next instruction which causes that file to be read into the CPU is executed. Otherwise, the previous state of that pin may be read into the CPU rather than the new state. When in doubt, it is better to separate these instructions with a `NOP` or another instruction not accessing this I/O port.

Figure 9-12: Successive I/O Operation



This example shows a write to PORTB followed by a read from PORTB.

Note: Data setup time = (0.25TCY - TPD)
whereTCY = instruction cycle
TPD = propagation delay

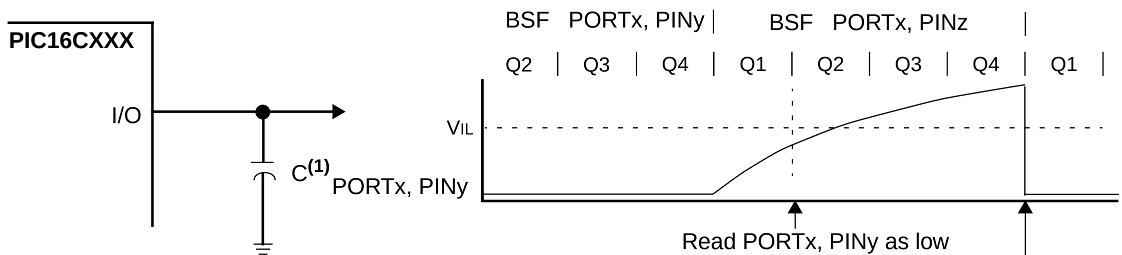
Therefore, at higher clock frequencies, a write followed by a read may be problematic due to external capacitance.

Figure 9-13 shows the I/O model which causes this situation. As the effective capacitance (C) becomes larger, the rise/fall time of the I/O pin increases. As the device frequency increases or the effective capacitance increases, the possibility of this subsequent PORTx read-modify-write instruction issue increases. This effective capacitance includes the effects of the board traces.

The best way to address this is to add an series resistor at the I/O pin. This resistor allows the I/O pin to get to the desired level before the next instruction.

The use of `NOP` instructions between the subsequent PORTx read-modify-write instructions, is a lower cost solution, but has the issue that the number of `NOP` instructions is dependent on the effective capacitance C and the frequency of the device.

Figure 9-13: I/O Connection Issues



Note: This is not a capacitor to ground, but the effective capacitive loading on the trace.

9.11 Initialization

See the section describing each port for examples of initialization of the ports.

> **Note:** It is recommended that when initializing the port, the data latch (PORT register) should be initialized first, and then the data direction (TRIS register). This will eliminate a possible pin glitch, since the PORT data latch values power up in a random state.

9.12 Design Tips

Question 1: ***Code will not toggle any I/O ports, but the oscillator is running. What can I be doing wrong?***

Answer 1:

1. Have the TRIS registers been initialized properly? These registers can be written to directly in the second bank (Bank1). In most cases the user is not switching to Bank1 (`BSF STATUS,RP0`) before writing zeros to the TRIS register.
2. If you are setting up the TRIS registers properly in Bank1 (RP0 = 1), you may not be returning to Bank0 before writing to the ports (`BCF STATUS,RP0`).
3. Is there a peripheral multiplexed onto those pins that are enabled?
4. Is the Watchdog Timer enabled (done at programming)? If it is enabled, is it being cleared properly with a `CLRWDT` instruction at least every 9 ms (or more if prescaled)?
5. Are you using the correct instructions to write to the port? More than one person has used the `MOVF` command when they should have used `MOVWF`.
6. For parts with interrupts, are the interrupts disabled? If not, try disabling them to verify they are not interfering.

Question 2: ***When my program reads a port, I get a different value than what I put in the port register. What can cause this?***

Answer 2:

1. When a port is read, it is always the pin that is read, regardless of its being set to input or output. So if a pin is set to an input, you will read the value on the pin regardless of the register value.
2. If a pin is set to output, say it has a one in the data latch; if it is shorted to ground you will still read a zero on the pin. This is very useful for building fault tolerant systems, or handling I^2C™ bus conflicts. (The I^2C bus is only driven low, and the pin is tristated for a one. If the pin is low and you are not driving it, some other device is trying to take the bus).
3. Mid-Range MCU devices all have at least one open drain (or open collector) pin. These pins can only drive a zero or tristate. For most Mid-Range devices this is pin RA4. Open drain pins must have a pull-up resistor to have a high state. This pin is useful for driving odd voltage loads. The pull-up can be connected to a voltage (typically less than V_{DD}) which becomes the high state.

Question 3: ***I have a PIC16CXXX with pin RB0 configured as an interrupt input, but am not getting interrupted. When I change my routine to poll the pin, it reads the high input and operates fine. What is the problem?***

Answer 3:

PORTB accepts TTL input levels (on most parts), so when you have an input of say 3V (with V_{DD} = 5V), you will read a one. However the buffer to the interrupt structure from pin RB0 is Schmitt Trigger, which requires a higher voltage (than TTL input) before the high input is registered. So it is possible to read a one, but not get the interrupt. The interrupt was given a Schmitt Trigger input with hysteresis to minimize noise problems. It is one thing to have short noise spikes on a pin that is a data input that can potentially cause bad data, but quite another to permit noise to cause an interrupt, hence the difference.

I^2C is a trademark of Philips Corporation.

Question 4: ***When I perform a BCF instruction, other pins get cleared in the port. Why?***

Answer 4:

1. Another case where a read-modify-write instruction may seem to change other pin values unexpectedly can be illustrated as follows: Suppose you make PORTC all outputs and drive the pins low. On each of the port pins is an LED connected to ground, such that a high output lights it. Across each LED is a 100 μF capacitor. Let's also suppose that the processor is running very fast, say 20 MHz. Now if you go down the port setting each pin in order; `BSF PORTC,0` then `BSF PORTC,1` then `BSF PORTC,2` and so on, you may see that only the last pin was set, and only the last LED actually turns on. This is because the capacitors take a while to charge. As each pin was set, the pin before it was not charged yet and so was read as a zero. This zero is written back out to the port latch (r-m-w, remember) which clears the bit you just tried to set the instruction before. This is usually only a concern at high speeds and for successive port operations, but it can happen, so take it into consideration.
2. If this is on a PIC16C7XX device, you have not configured the I/O pins properly in the ADCON1 register. If a pin is configured for analog input, any read of that pin will read a zero, regardless of the voltage on the pin. This is an exception to the normal rule that the pin state is always read. You can still configure an analog pin as an output in the TRIS register, and drive the pin high or low by writing to it, but you will always read a zero. Therefore if you execute a Read-Modify-Write instruction (see previous question) all analog pins are read as zero, and those not directly modified by the instruction will be written back to the port latch as zero. A pin configured as analog is expected to have values that may be neither high nor low to a digital pin, or floating. Floating inputs on digital pins are a no-no, and can lead to high current draw in the input buffer, so the input buffer is disabled.

9.13 Related Application Notes

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

Title	**Application Note #**
Improving the Susceptibility of an Application to ESD	AN595
Clock Design using Low Power/Cost Techniques	AN615
Implementing Wake-up on Keystroke	AN528
Interfacing to AC Power Lines	AN521
Multiplexing LED Drive and a 4 x 4 Keypad Sampling	AN529
Using PIC16C5X as an LCD Drivers	AN563
Serial Port Routines Without Using TMR0	AN593
Implementation of an Asynchronous Serial I/O	AN510
Using the PORTB Interrupt on Change Feature as an External Interrupt	AN566
Implementing Wake-up on Keystroke	AN522
Apple Desktop Bus	AN591
Software Implementation of Asynchronous Serial I/O	AN555
Communicating with the I^2C Bus using the PIC16C5X	AN515
Interfacing 93CX6 Serial EEPROMs to the PIC16C5X Microcontrollers	AN530
Logic Powered Serial EEPROMs	AN535
Interfacing 24LCXXB Serial EEPROMs to the PIC16C54	AN567
Using the 24XX65 and 24XX32 with Stand-alone PIC16C54 Code	AN558

9.14 Revision History

Revision A

This is the initial released revision of the I/O Ports description.