Parallel Port I/O

The simplest type of I/O via the PIC18F242 external pins is **parallel port** I/O.

The 18F242 has three parallel ports:

PORTA – 7 bits, bidirectional
PORTB – 8 bits, bidirectional
PORTC – 8 bits, bidirectional

We will use PORTB pins most of the time because the PORTA, PORTC pins will be used for other functions beside parallel I/O.

Each pin on these ports can either be an input or output – the data direction is controlled by the corresponding bit in the TRISA, TRISB, TRISC registers ('1' = input, '0' = output).

LATA, LATB, LATC are latches that hold the last value written to ports A, B, C.

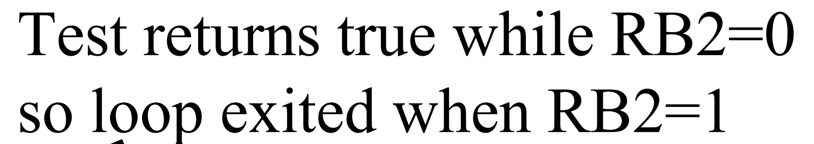
PORTB Example

Set the upper four bits of PORTB to outputs, lower four bits to be inputs:

```
TRISB = 0x0f;
```

Drive RB4, RB5 high; RB6, RB7 low:

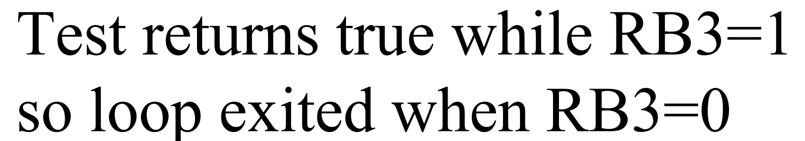
```
PORTB = 0x30;
```



Wait until input RB2 is high:

```
while (!bittst(PORTB,2)) ;
```

Wait until input RB3 is low:




```
while (bittst(PORTB,3)) ;
```

PORTB Example (cont.)

Individual PORT bits are named as RB0, RB1, ..RA0, etc. so this can be used in C code instead of using the *bittst* macros.

Wait until input RB2 is high:

```
while (!RB2) ;
```

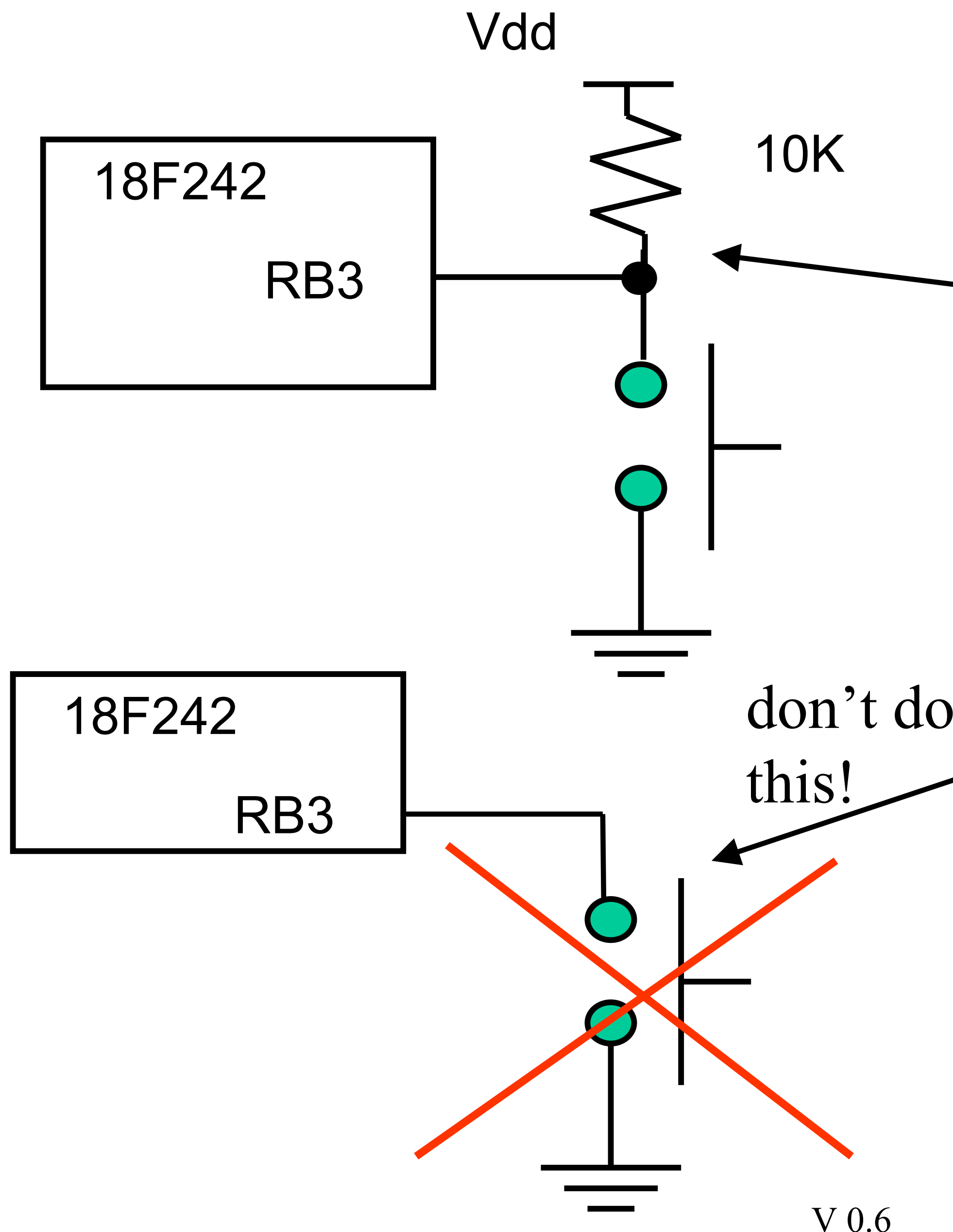
Test returns true while RB2=0 so loop exited when RB2=1

Wait until input RB3 is low:

```
while (RB3) ;
```

Test returns true while RB3=1 so loop exited when RB3=0

Switch Input



External pullup

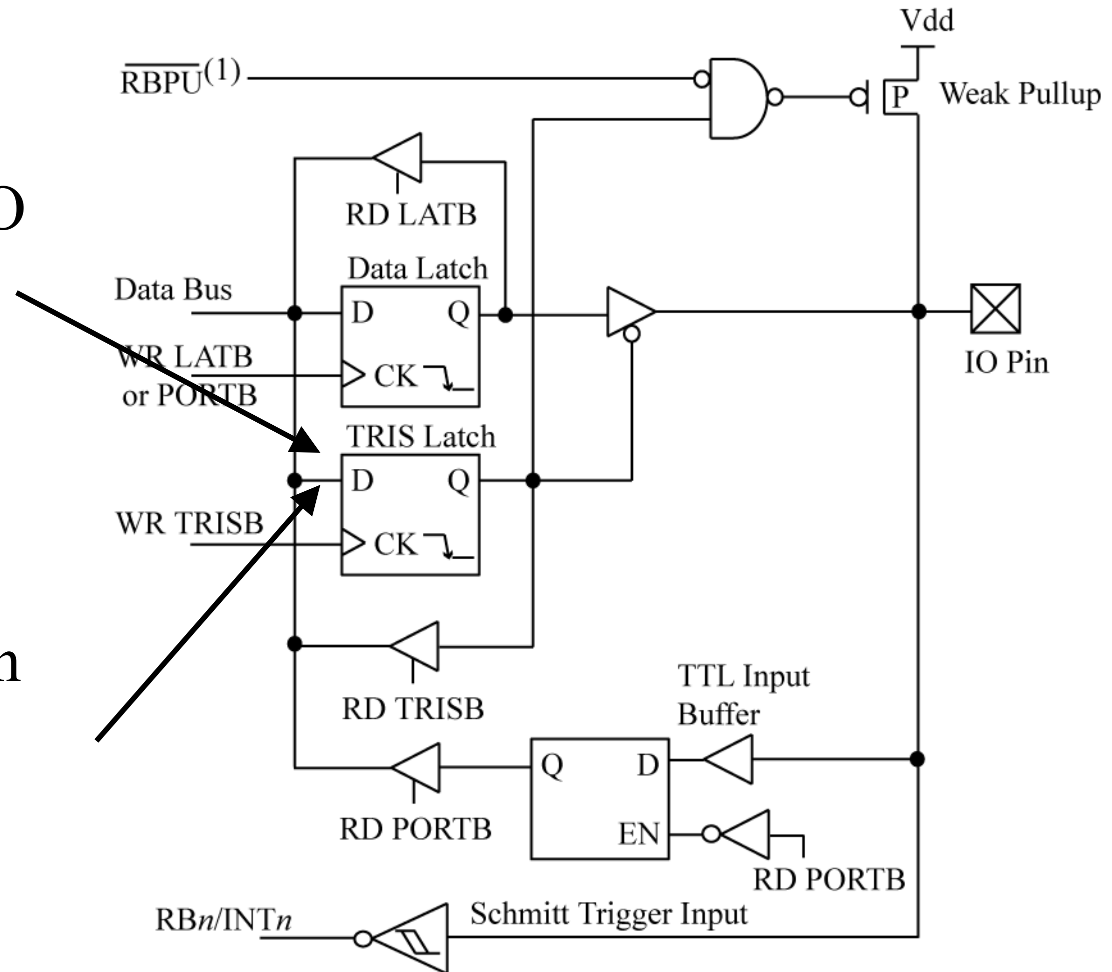
When switch is pressed RB3 reads as '0', else reads as '1'.

If pullup not present, then input would float when switch is not pressed, and input value may read as '0' or '1' because of system noise.

PORTB Pin Diagram: RB2:RB0

If TRIS bit a 0, tri-state buffer is enabled, value in data latch drives the IO pin.

If TRIS bit a 1, tri-state buffer is disabled, IO pin either floats or is driven by an external driver.

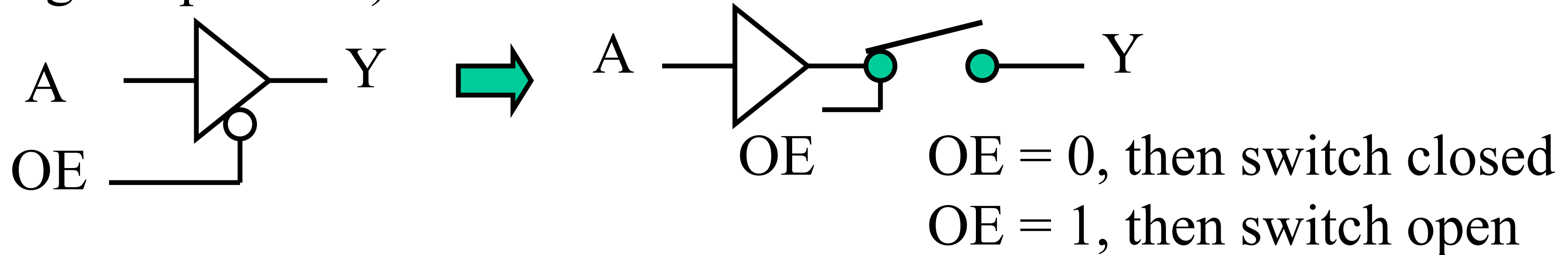


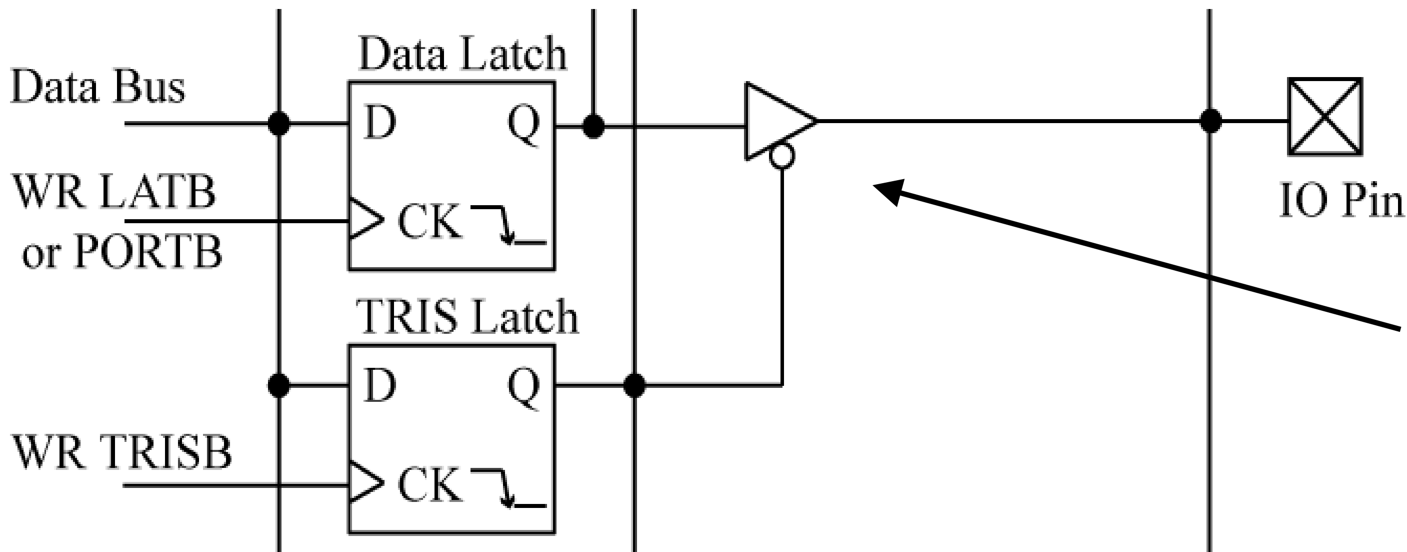
(1) To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the $\overline{\text{RPBU}}$ bit (INTCON2[7])

Figure redrawn by author from PIC18Fxx2 datasheet (DS39564B), Microchip Technology Inc.

Aside: Tri-State Buffer (TSB) Review

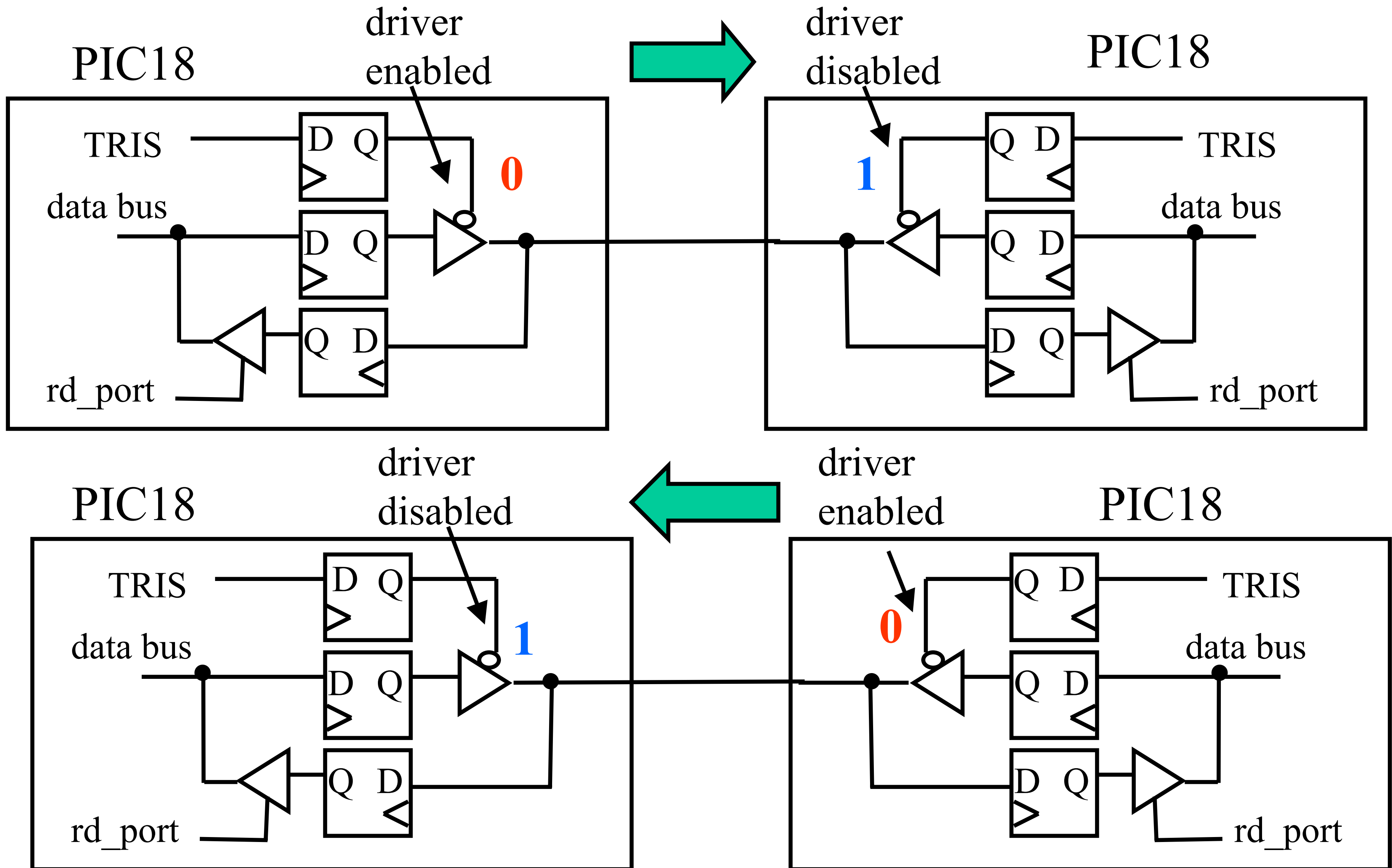
A tri-state buffer (TSB) has input, output, and output-enable (OE) pins. Output can either be '1', '0' or 'Z' (high impedance).





Output Enable (OE) of TSB. If asserted, output = input. If negated, output is high impedance (output disconnected)

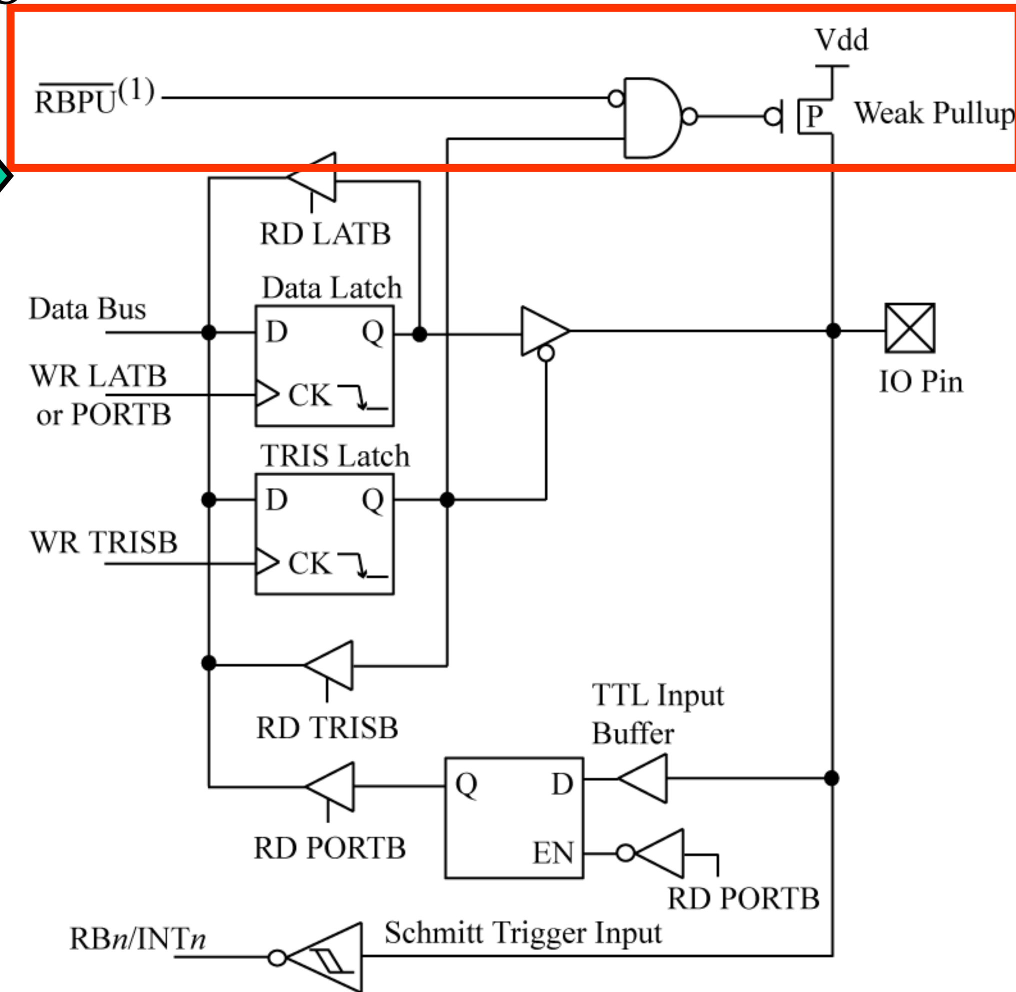
Bi-directional, Half-duplex Communication



PORTB weak pullups

Can enable weak pullups on all RB pins configured to be inputs by clearing the RBPU bit in the OPTION register

```
RBPU = 0;
   or
bitclr(INTCON2,7);
```



(1) To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the $\overline{\text{RPBU}}$ bit (INTCON2[7])

Figure redrawn by author from PIC18Fxx2 datasheet (DS39564B), Microchip Technology Inc.

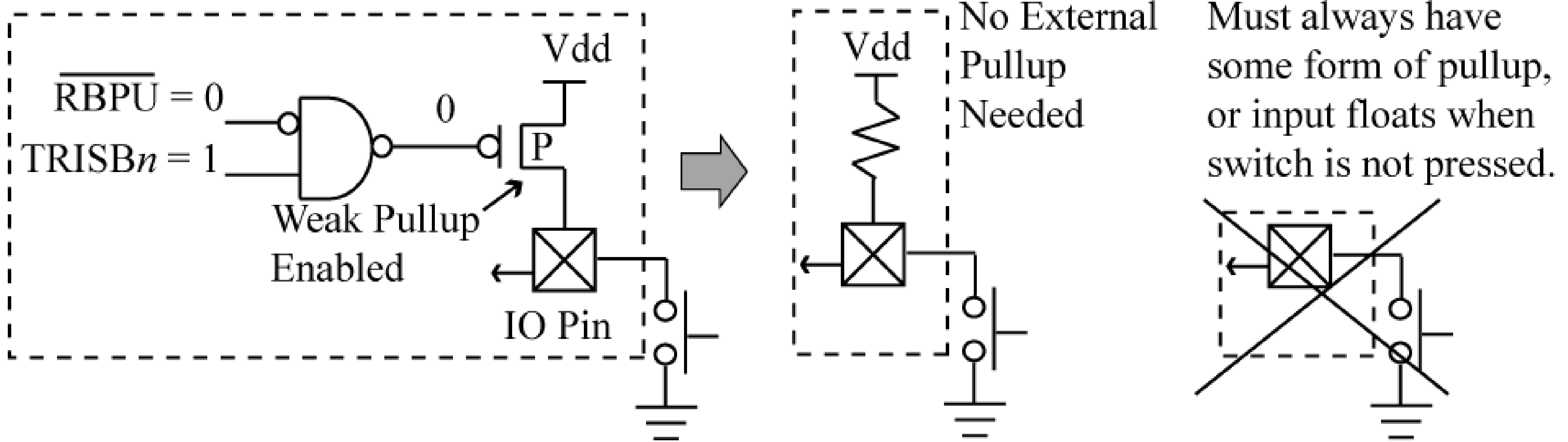
TABLE 9-4: SUMMARY (

Name	Bit 7	Bit 6
PORTB	RB7	RB6
LATB	LATB Data Output Reg	
TRISB	PORTB Data Direction	
INTCON	GIE/ GIEH	PEIE/ GIEL
INTCON2	RBPU	INTEDG0
INTCON3	INT2IP	INT1IP

Copyright Microchip, from PIC18xx2 Datasheet DS39564B

Removes the need for an external pullup.

Why are weak pullups useful?



If enabled, removes need for external pullup resistor on low-true input switch.

LATB versus PORTB

A read of LATB returns the value of the latch that drives the external pin.

A read of PORTB returns the value of the external PIN.

LATB holds the last value written to PORTB while PORTB is the value of the external pin.

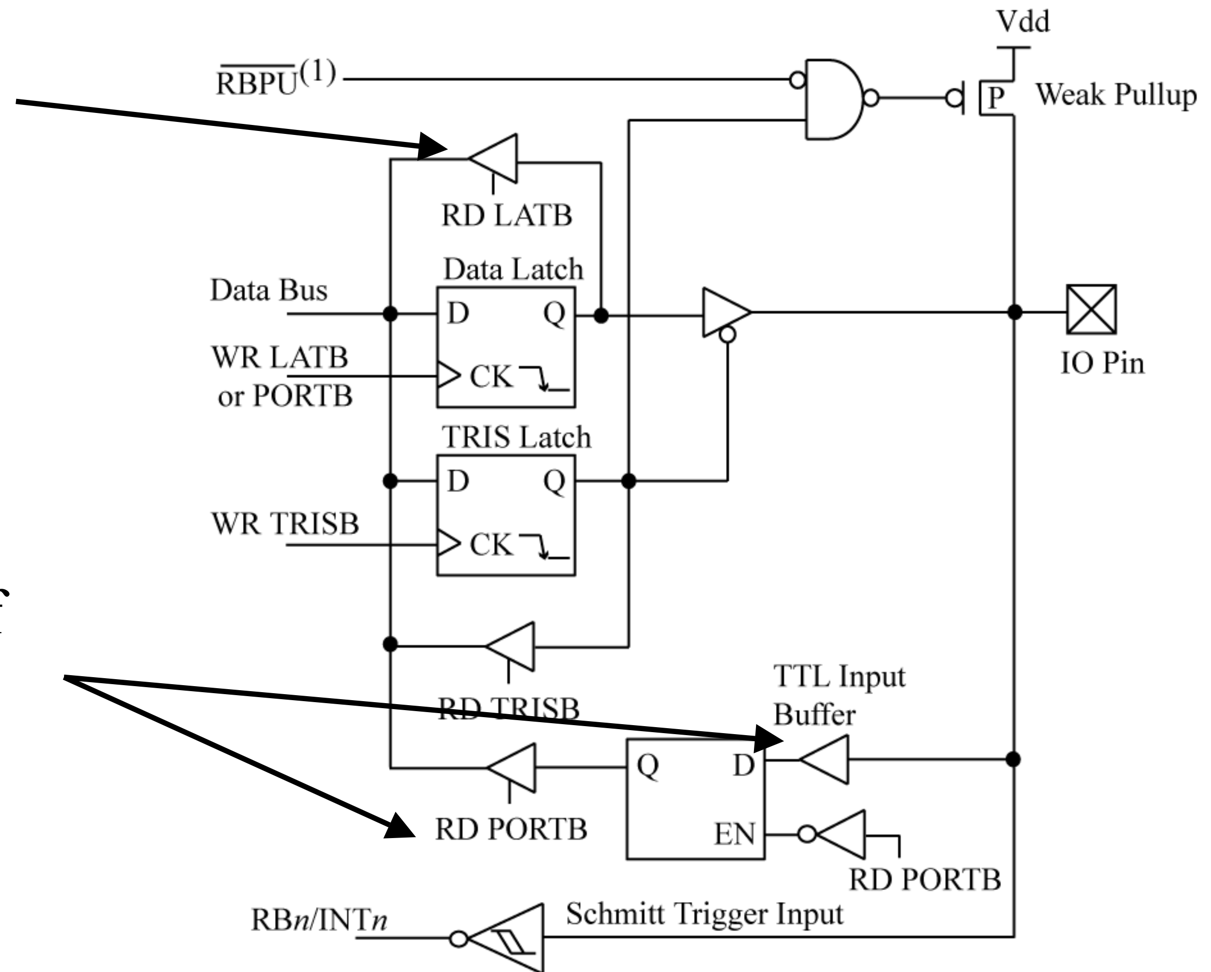


Figure redrawn by author from PIC18Fxx2 datasheet (DS39564B), Microchip Technology Inc.

PORTA Parallel IO

On the 18F242, the PORTA RA0:RA3 and RA5 pins are also used for as the inputs to the analog-to-digital converter module.

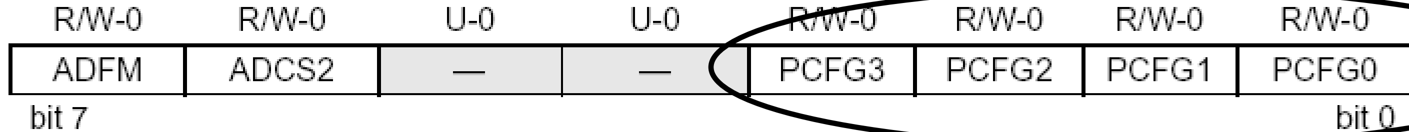
By default, they are analog input pins, not bi-directional digital I/O pins. If a read is done on these pins while they are configured as analog inputs, a '0' will always be returned.

To enable RA0:RA3, RA5 pins to functions as digital pins, the ADCON1 (A/D configuration register) must be set to the value 0x06:

```
// configure port A to be all digital inputs
 TRISA = 0xff;
 ADCON1 = 0x06;
```

PORT A Pin Configuration

REGISTER 17-2: ADCON1 REGISTER



bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C / R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8 / 0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7 / 1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5 / 0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4 / 1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3 / 0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2 / 1
011x	D	D	D	D	D	D	D	D	—	—	0 / 0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6 / 2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6 / 0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5 / 1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4 / 2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3 / 2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2 / 2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1 / 0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1 / 2

A = Analog input D = Digital I/O
C/R = # of analog input channels / # of A/D voltage references

Bits 3:0 of ADCON1 control the configuration of the PORTA pins in terms of digital vs. analog.

The datasheet in the A/D section has a complete description.

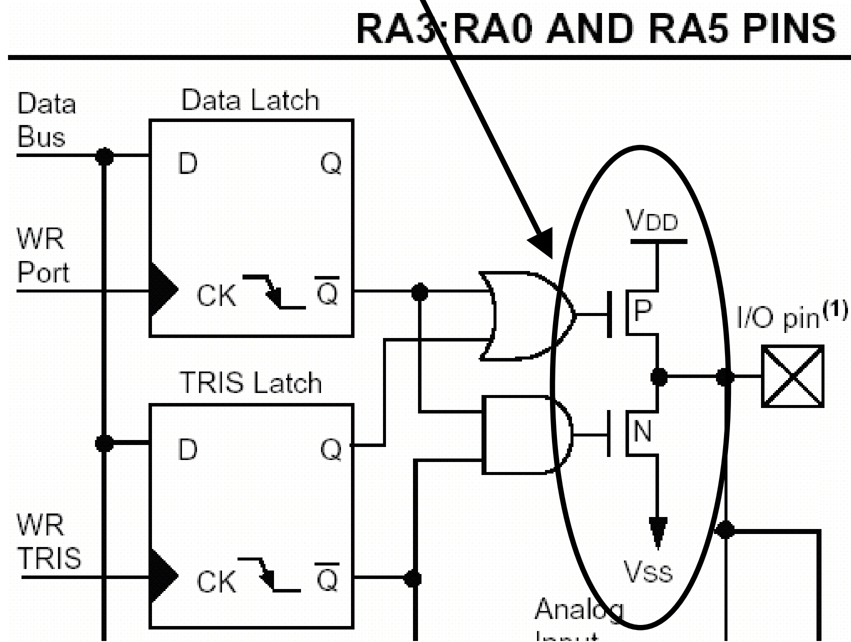
Copyright Microchip, from PIC18xx2 Datasheet DS39564B

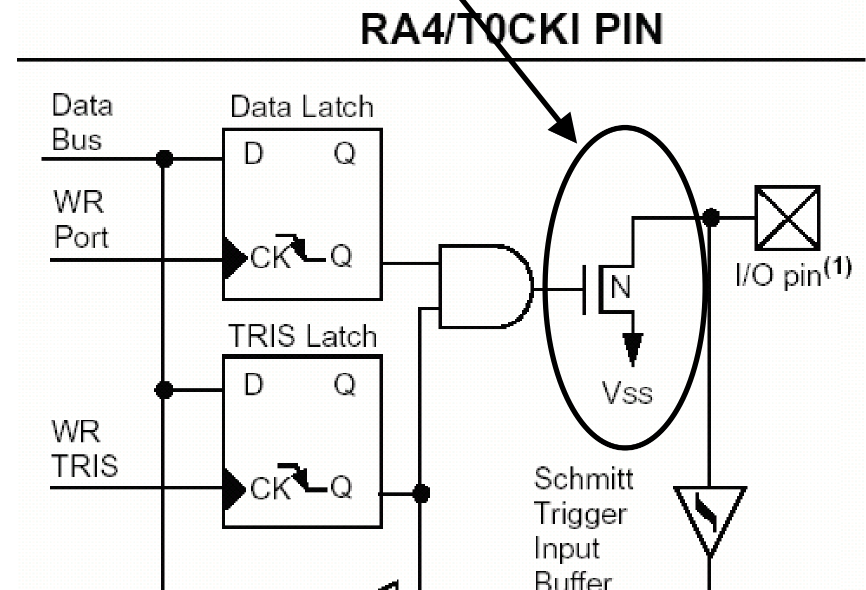
RA4 Pin: Open Drain Output

RA4 is different from RA0:RA3, RA5 in that it is an open drain output. RA4 can only pull LOW, it cannot pull high.

P/N transistors both present, can pull high/low

Only N present, can only pull low

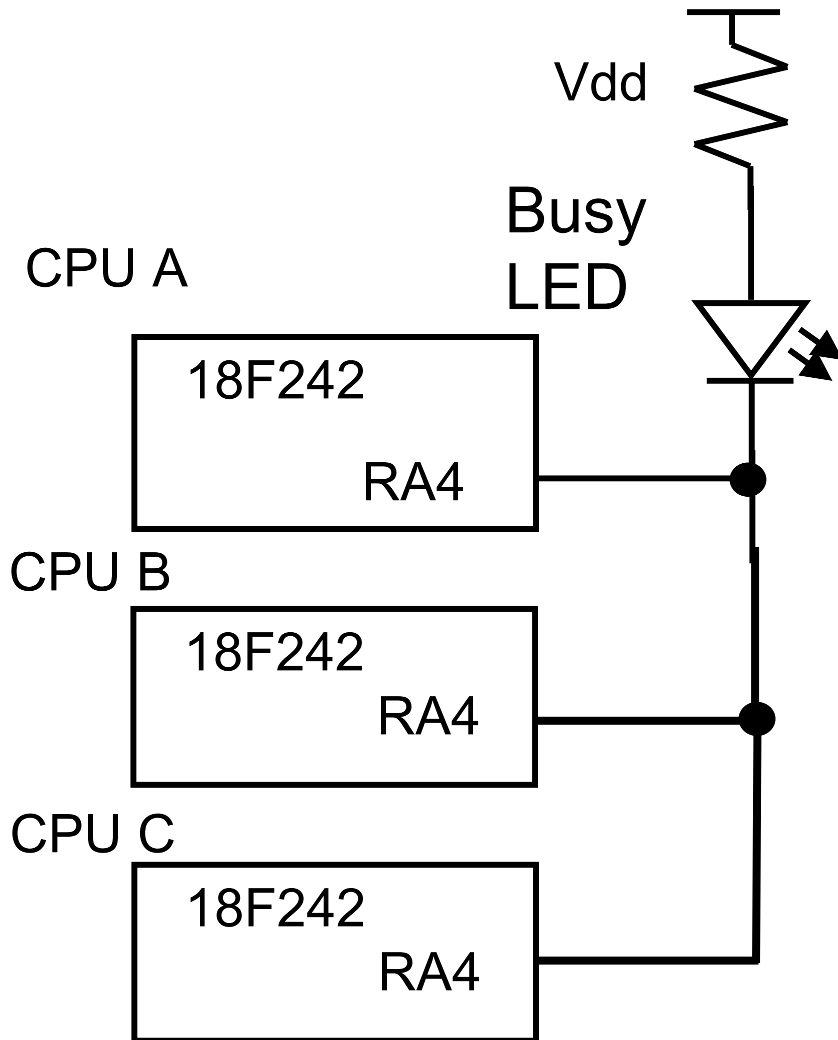




Copyright Microchip, from PIC18xx2 Datasheet DS39564B

Copyright Microchip, from PIC18xx2 Datasheet DS39564B

Why Open Drain?



Useful because can tie open-drain outputs together without external logic, only an external pullup.

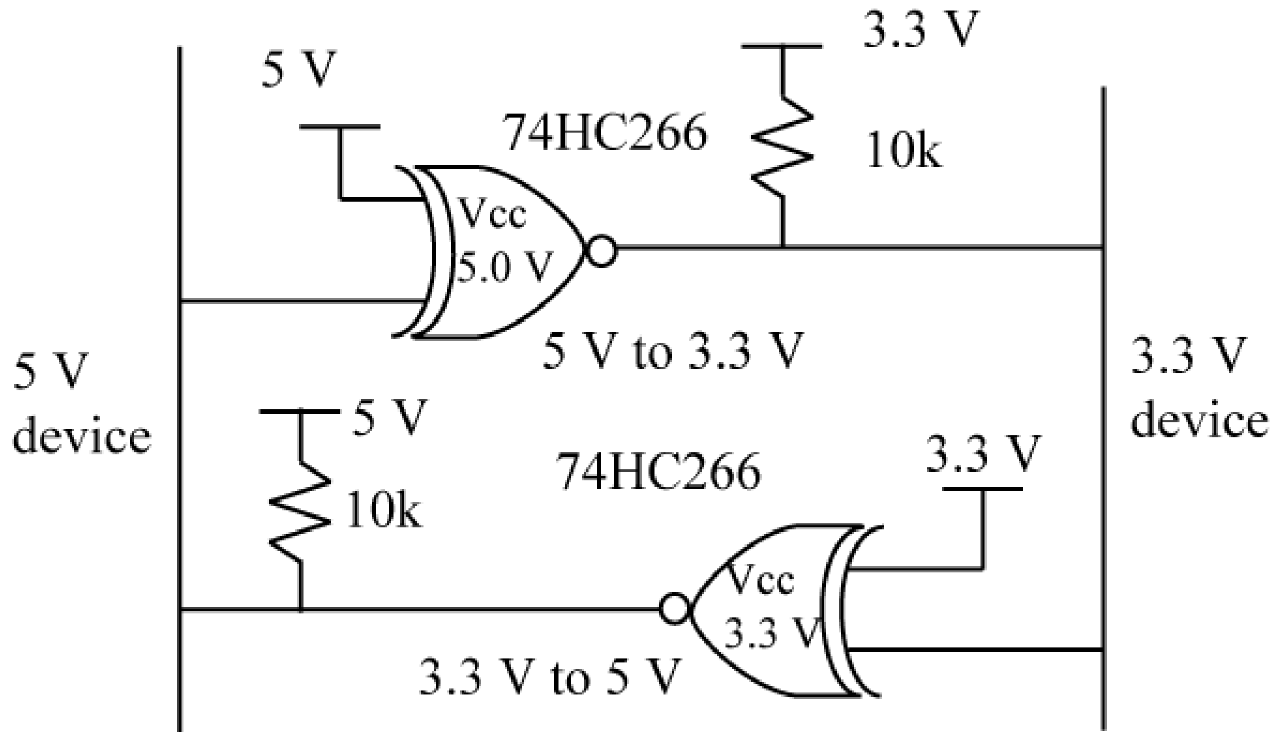
Assume CPUs A,B,C are all working on different tasks, and want to know when all are finished.

When working on a task, a CPU asserts RA4 low. When finished, negate RA4. If any CPU is busy, LED will turn on. If all CPUs finished, LED will turn off.

Cannot do this with non-open drain output because of clash of some outputs driving low, some high.

This type of connection is often called a **wired-or**. If CPU A *or* B *or* C is busy, then LED is on.

Mixed Voltage Interfacing using Open Drain Gates



The 74HC266 is a 2-input exclusive NOR gate with open drain outputs. By tying one input high, it becomes a non-inverting buffer. The pullup resistor on the output provides the logic high.

PORTC Parallel IO

- We will not look at PORTC Parallel IO
- PORTC pins shared with many other functions of the PIC18F242
- If parallel IO is needed, will always use PORTB first, then PORTA if needed

ledflash.c Program Listing

```
#include <pic18.h>
```
← Standard PIC18 header file

```
__CONFIG(1,HSPLL);
__CONFIG(2, BORDIS & PWRTDIS & WDTDIS);
__CONFIG(4, DEBUGDIS & LVPDIS);
```
← Specifies configuration bits

Time delay implemented via software loops – delay depends upon clock frequency and loop count values.

```
void a_delay(void)
{
  unsigned int i,k;
  // change count values to alter delay
  for (k=1800; --k;) {
    for(i = 200 ; --i ;);
  }
}
```
← Subroutine for software delay

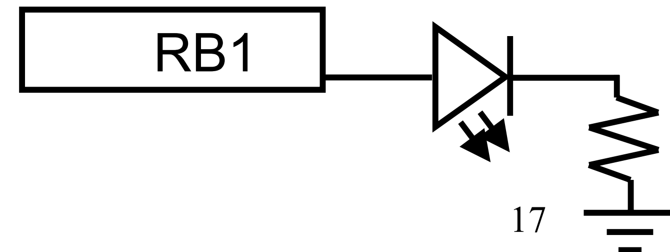
```
// just flash LED on port PB1
// use this to test if PIC board is alive
main(void){
  TRISB1 = 0;       // configure RB1 as output
  RB1 = 0;          // set RB1 low initially

  while(1) {
    a_delay();  // call delay subroutine
    RB1 = 1;    // turn on RB1 (LED)
    a_delay();
    RB1 = 0;    // turn off RB1 (LED)
  }
}
```
← Infinite loop that blinks LED

Infinite loop simply alternates between turning LED ON (output port = 1), then OFF (output port = 0), with a time delay between each port operation.

Only exit is through MCLR# reset or power cycle

18F242



Copyright Thomson/Delmar Learning 2005. All Rights Reserved.

DelayUs, DelayMS

```
#define FOSC     29491L   // Internal clock freq in KHz
#define MHZ      *1000L   // number of KHz in a MHz
#define KHZ      *1       // number of KHz in a KHz

#define DelayUs(x) { unsigned char _dcnt; \
                     _dcnt = ((x* FOSC)/(12MHZ)); \
                     while(--_dcnt != 0) \
                       continue; }

void DelayMs(unsigned char cnt)
{
        unsigned char   i;
        do {
            i = 20;
            do {
                DelayUs(50);
            } while(--i);
        } while(--cnt);
}
```

Be cautious of overflow in `_dcnt` variable. For 40 MHz, `x` max is 76.

Compiles to loop with 12 clock cycles, 1 µs per iteration if FOSC = 12 MHz

```
        movlb ??
loop    decfsz _dcnt,f
        bra loop
      ..loop exit...
```

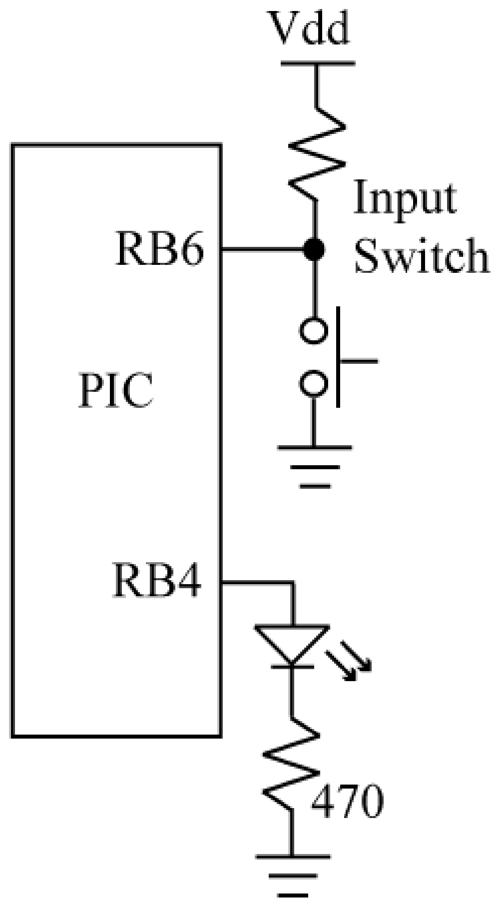
Call DelayUs(50) 20 times for 1 ms delay

When using these functions, must use full optimization!

Better software delay loop routines. Maximum value of DelayUs parameter depends upon clock value FOSC; _dcnt calculation cannot exceed 255.

Copyright Thomson/Delmar Learning 2005. All Rights Reserved.

LED/Switch IO: Count number of press/releases



Count number of switch presses.

```
main(){
int i;
 TRISB = 0xEF;
 RB4 = 0;
 i = 0;
 while (1) {
    if (!RB6) {
    //switch pressed
    //increment i
    i++;
    }
 }
}
```

a. Incorrect, variable `i` is incremented as long as the switch is pushed, which could be a long time!

```
main(){
 int i;
 TRISB = 0xEF;
 RB4 = 0;
 i = 0;
 while (1) {
   // wait for press
   while (RB6); //loop (1)
   DelayMs(30); //debounce
   // wait for release
   while (!RB6); //loop (2)
   DelayMs(30); // debounce
   i++;
 }
```

b. Correct, loop 1 executed while switch is not pressed. Once pressed, becomes trapped in loop 2 until switch is released, at which point variable `i` is incremented.

Copyright Thomson/Delmar Learning 2005. All Rights Reserved.

Count #number of switch press/releases.

A common error:

```
int i;
TRISB = 0xEF;
RB4 = 0;
i = 0;
while (1) {
  if (!RB6) {
   /* switch pressed, increment */
  i++;
  }
}
```

alternate statements:

if (RB6 == 0) {

if (!(bittst(PORTB,6))) {

if (bittst(PORTB,6) == 0) {

Will be incremented MANY times. A human CANNOT press/release a switch faster than a few milliseconds, so code in loop executed an unknown number of times.

Count #number of switch press/releases

Correct Solution:

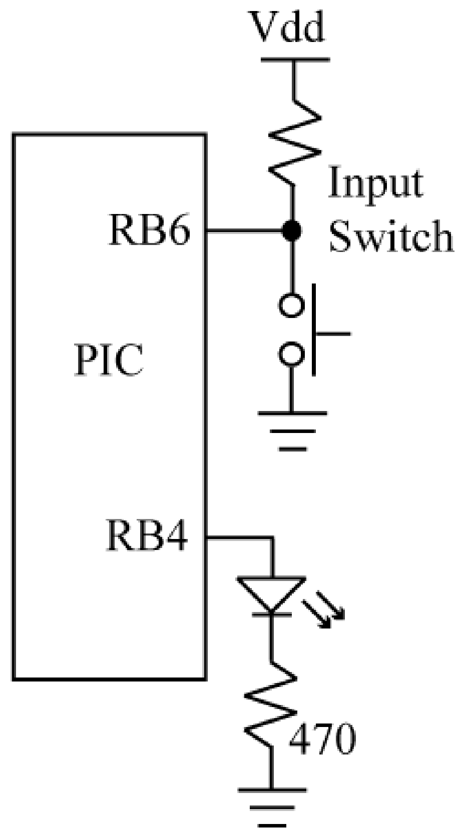
```
int i;
TRISB = 0xEF;
RB4 = 0;
i = 0;
while (1) {
  // wait until button is pressed
  while (RB6);DelayMs(30);
 // wait for button to be released
  while (!RB6);DelayMs(30);
  i++;
}
```

while(bittest(PORTB,6));

while(!bittest(PORTB,6));

Delay to let switch bounce settle out.

Toggle LED for each switch press/release



Toggle LED for each switch press.

```
main(){
 TRISB = 0xEF;
 RB4 = 0;
 while (1) {
   if (!RB6){
     //switch pressed,
     //turn on LED
     RB4 = 1;
   }
   if (RB6) {
   // switch released,
   // turn off LED
     RB4 = 0;
    }
 }
```

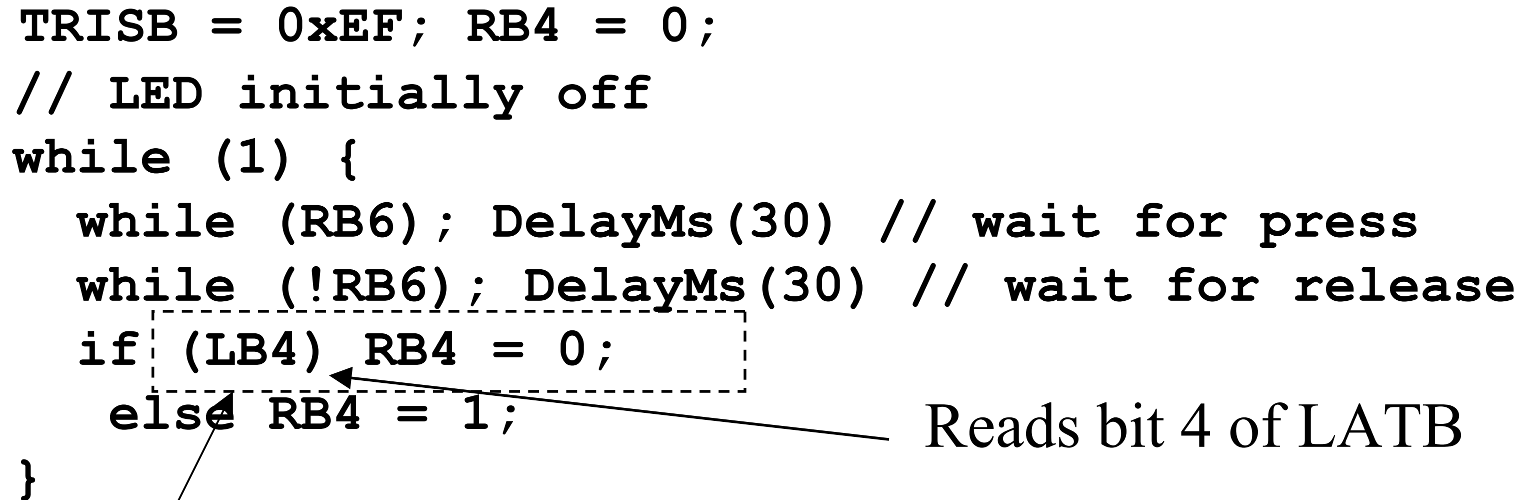
a. Incorrect, LED is "on" only when switch is pressed; it is "off" when switch is released.

```
main(){
 TRISB = 0xEF;
 RB4 = 0;
while (1) {
   // wait for press
   while (RB6);
   DelayMs(30); //debounce
   //wait for release
   while (!RB6);
   DelayMs(30); //debounce
   RB4 = 1; // turn on
   // wait for press
   while (RB6);
   DelayMs(30); //debounce
   // wait for release
   while (!RB6);
   DelayMs(30); //debounce
   RB4 = 0; // turn off
 }
```

b. Correct, LED is toggled for each press and release of the switch.

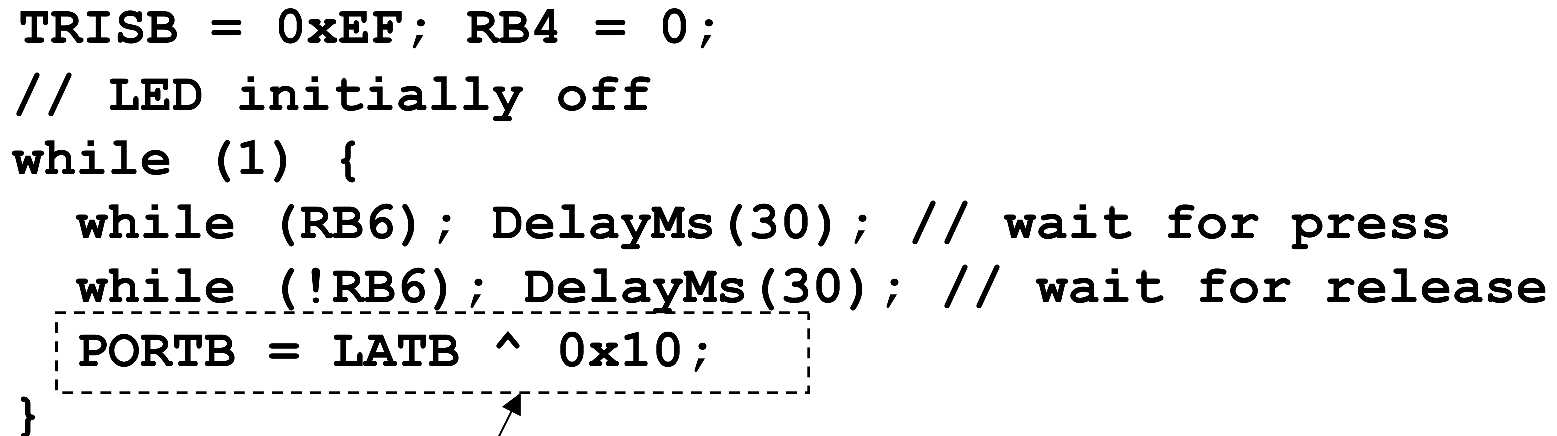
Copyright Thomson/Delmar Learning 2005. All Rights Reserved.

Toggle LED for each switch press/release, alternate solution



Read LATB, bit 4 to return the last value written to PORTB, bit4. This will set the RB4 output to a '0' if it is currently a '1', to a '1' if it is currently a '0'.

Toggle LED for each switch press/release



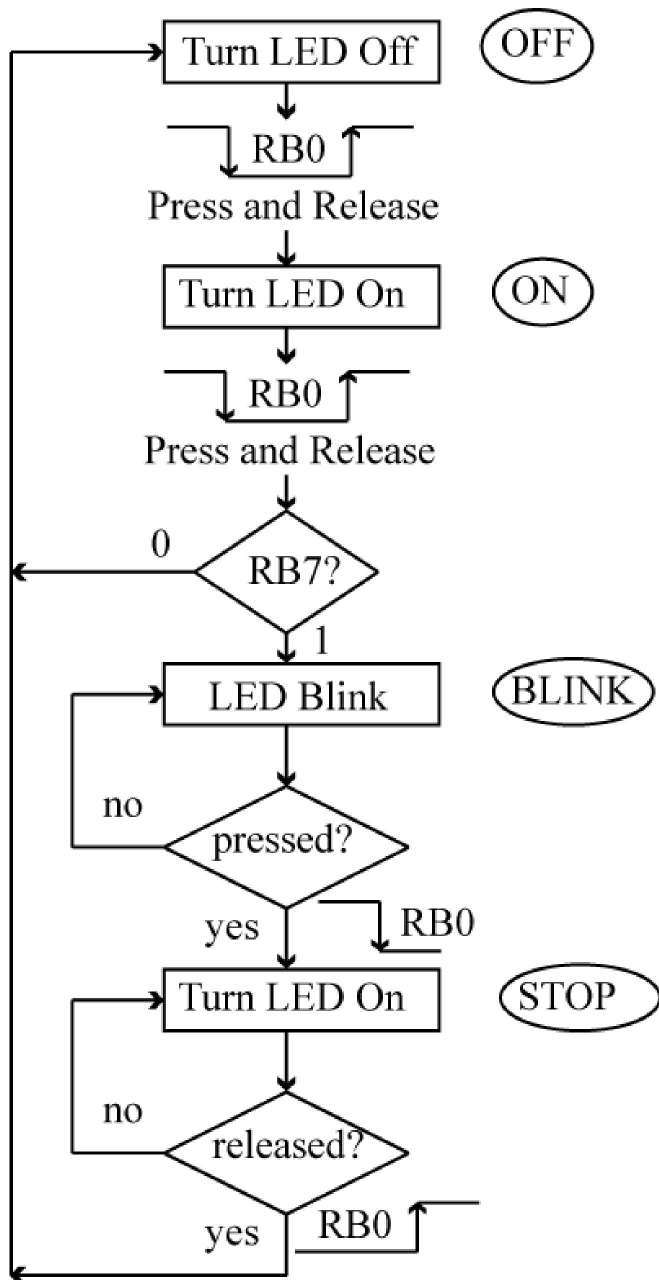
```
TRISB = 0xEF; RB4 = 0;
// LED initially off
while (1) {
  while (RB6); DelayMs(30); // wait for press
  while (!RB6); DelayMs(30); // wait for release
  PORTB = LATB ^ 0x10;
}
```

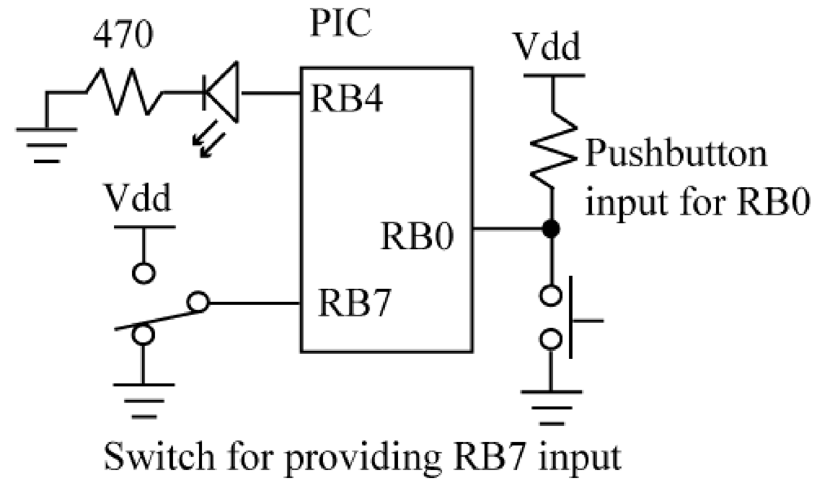
This works because the ^ is the exclusive OR operation, and an exclusive OR with a '1' value will toggle that bit.

Another LED/Switch Problem

- a. LED initially off
- b. A press & release of switch turns LED on
- c. On next press & release, if RB7 is 0, go back to "a", else start LED blinking.
- d. On switch press, stop blinking, freeze LED ON while button is held down. On button release, goto "a".

Translate the above word description into a Finite State Machine (FSM) description (on next page). This gives you a structured approach for solving IO problems.

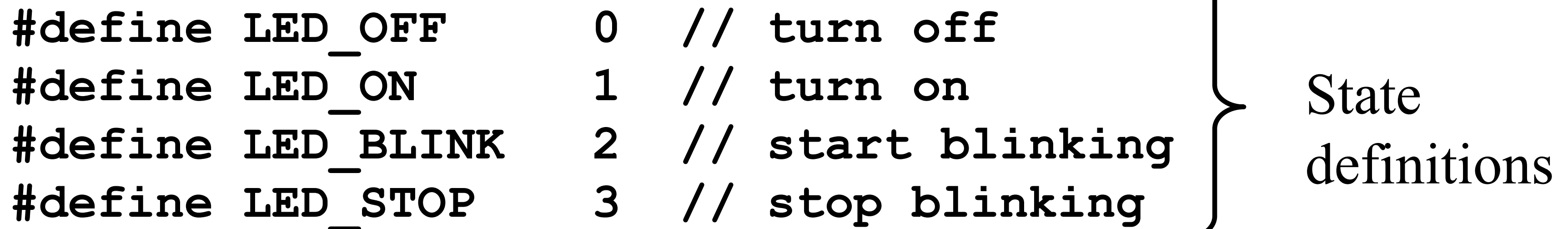




If PORTB weak pullup is enabled, external resistor on RB0 and Vdd selection on RB7 are not required.

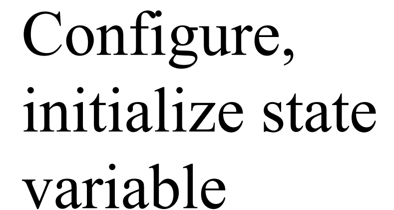
Finite State Machine approach for IO problem – provides a more structured technique for coding.

Copyright Thomson/Delmar Learning 2005. All Rights Reserved.




```
#define LED_OFF      0  // turn off
#define LED_ON       1  // turn on
#define LED_BLINK    2  // start blinking
#define LED_STOP     3  // stop blinking
unsigned char state;
```

```
main(void){
  serial_init(95,1);
  pcrlf();
  printf("Led Switch/IO started");
  pcrlf();
  state = LED_OFF;
  // RB7, RB0 are inputs
  TRISB = 0xEF;
  LATB = 0x00;
  // enable the weak pullup on port B
  RBPU = 0;
```



```
while(1) {
    switch (state) {
      case LED_OFF:
         printf("LED_OFF");pcrlf();
          LATB4 = 0;
          while(RB0);DelayMs(30);  // wait for press
          while(!RB0); DelayMs(30); // wait for release
          state = LED_ON;
          break;
      case LED_ON:
         printf("LED_ON");pcrlf();
          LATB4 = 1;
          while(RB0); DelayMs(30); // wait for press
          while(!RB0); DelayMs(30); // wait for release
          if (RB7) state = LED_BLINK;
           else state = LED_OFF;
          break;
```

print statement for debugging

Could use RB4 here as well

change state so next time through case statement will choose different case.

Choose next state based on RB7

```
    case LED_BLINK:
      printf("LED_BLINK");pcrlf();
      // while not pressed
      while (RB0) {
        // toggle LED
        if (LATB4) LATB4 = 0;
           else LATB4 = 1;
        DelayMs(250);
      }
      DelayMs(30);
      state = LED_STOP;
     break;
   case LED_STOP:
      printf("LED_STOP");pcrlf();
      LATB4 = 1;  // freeze on
      // wait for release
      while(!RB0);DelayMs(30);
      state = LED_OFF;
      break;
   }
  }
}
```

Blink LED as long as switch is not pressed

Exit loop and change state on switch press

Freeze LED on and exit state when switch is released.

Typical Output to Console

```
Led Switch/IO started
LED_OFF
LED_ON
LED_BLINK
LED_STOP
LED_OFF
LED_ON
LED_BLINK
LED_STOP
LED_OFF
LED_ON
LED_OFF
LED_ON
LED_OFF
```

Initial state

Press & Release

Press & Release, RB7 = 1, so blink LED

Press, stop blinking

Release, turn off

Press & Release

Press & Release, RB7 = 1, so blink LED

Press, stop blinking

Release, turn off

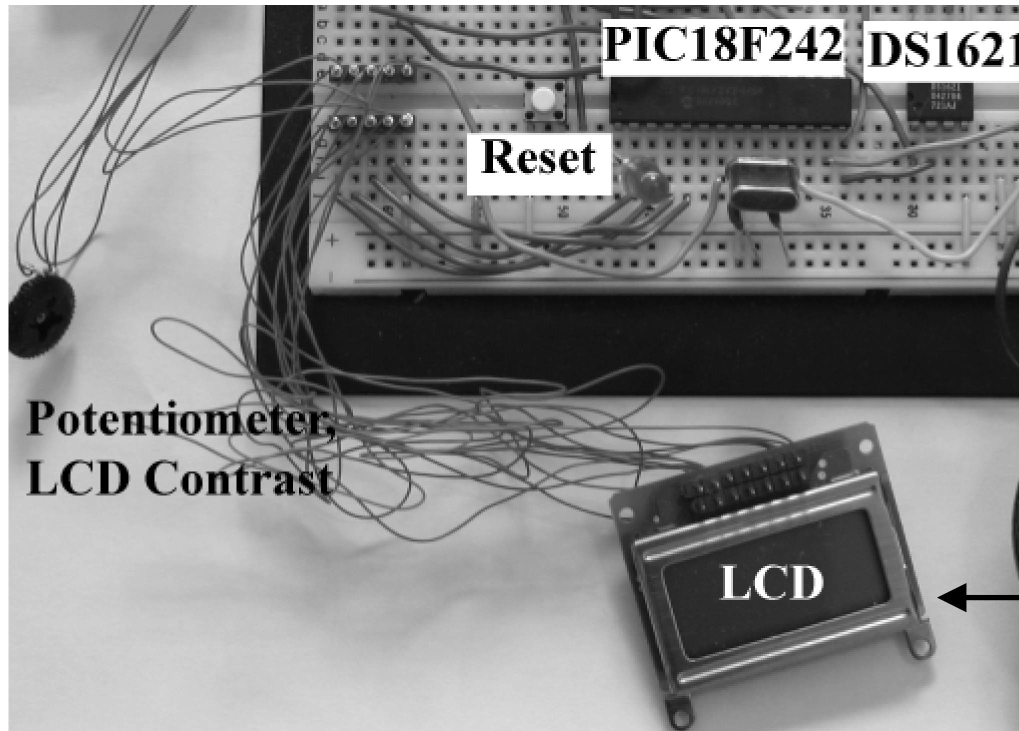
Press & Release

Press & Release, RB7 = 0, so go back to OFF

Press & Release

Press & Release, RB7 = 0, so go back to OFF

LCD – Liquid Crystal Display Module

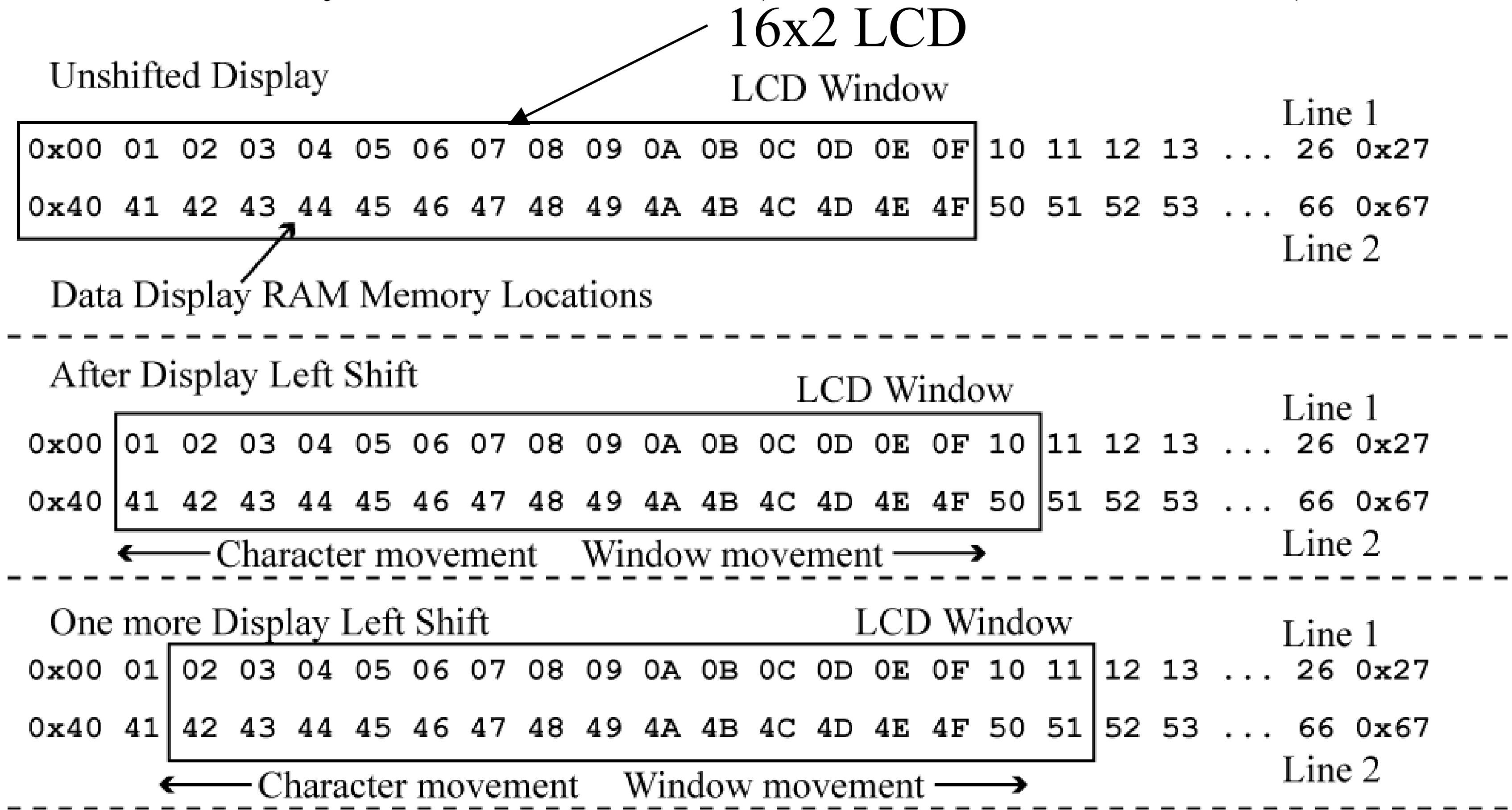


Liquid Crystal Display (LCD) module : K x N where k is number of characters displayed, N is number of lines.

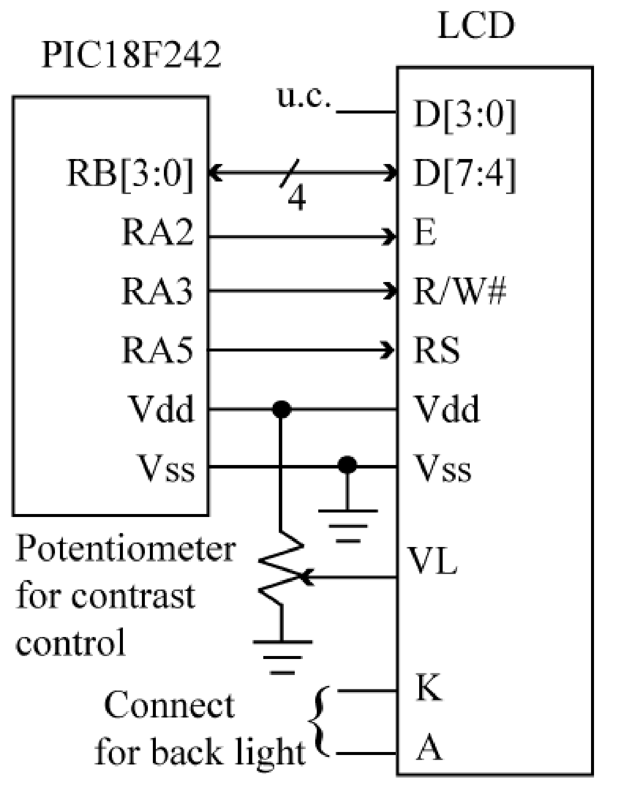
8x2 Module

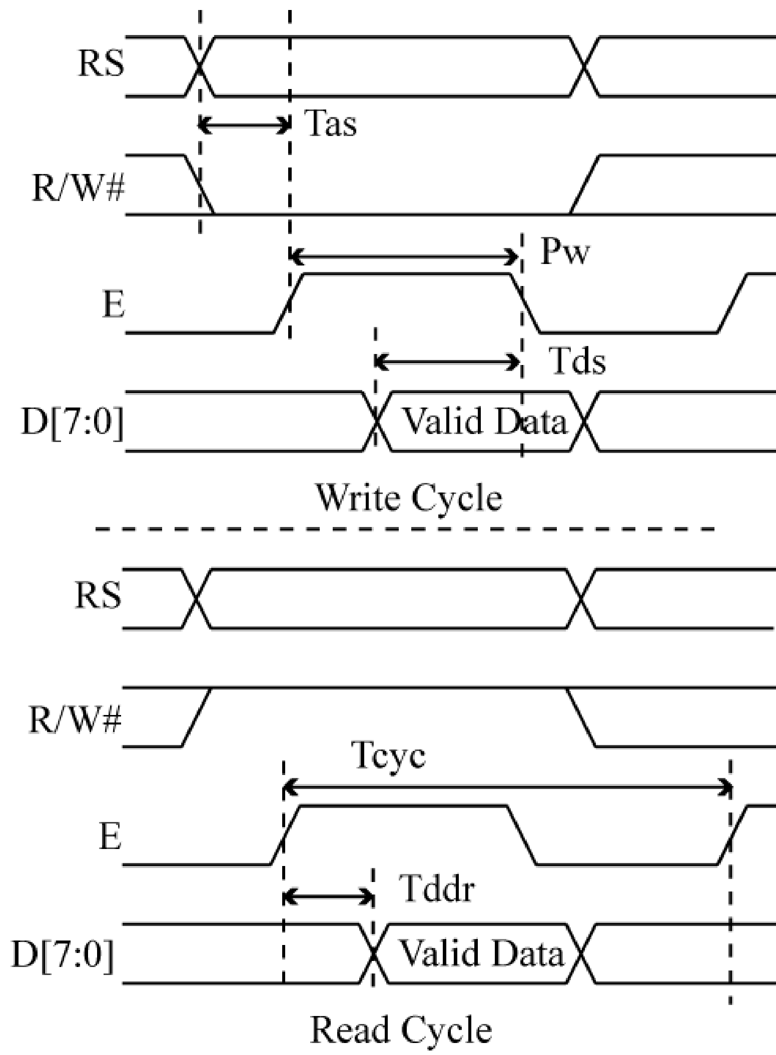
Liquid Crystal Display (LCD) module : K x N where k is number of characters displayed, N is number of lines.

Internal memory has 80 locations (2 lines of 40 characters)



© Thomson/Delmar Learning, 2005





Item	Max (ns)	Min (ns)
RS, R/W# setup (Tas)		40
E pulse width (Pw)		230
Data setup time (Tds)		60
Data delay time (Tddr)	360	
Cycle Time (Tcyc)		500

4-bit Interface, E is a data strobe that clocks in data.

© Thomson/Delmar Learning, 2005

LCD Commands

When RS = 0, then byte that is sent is a command byte (clear display, shift left, shift right, move cursor, etc).

When RS = 1, then byte that is sent is ASCII data that is written to current cursor location, which is incremented by one after the write.

If the R/W# input is low, then writing data. If R/W# is high, then reading data. Can read a status code to determine if the LCD module is finished with the last command or not.

LCD Commands

Command	RS	R/W#	D7:D0	Description
Clear Display	0	0	0000 0001	Clear display, return cursor to home position (82 us ~ 1.64ms)
Return Home	0	0	0000 001x	Returns cursor and shifted display to home (40 us ~ 1.64 ms)
Entry Mode Set	0	0	0000 01*d*0	Enable the display, set cursor move direction (*d*=1 increment, *d*=0 decrement) (40 us)
Display On/Off	0	0	0000 1*dcb*	Display on/off (*d*), Cursor on/off (*c*), Blink at cursor position on/off (*b*) (40 us)
Cursor & Display Shift	0	0	0001 *cr*00	*c*=1 shift display, *c*=0 move cursor, *r* = 1 right shift, *r* = 0 left shift
Function Set	0	0	001*i* *n*000	8-bit interface (*i*=1), 4-bit interface (*i*=0),One line (*n*=0), Two lines (*n*=1) (40 us)
Set DD Address	0	0	1*nnn nnnn*	DD RAM address set equal to *nnnnnnnn* (40 us)
Read Busy Flag	0	1	*fnnn nnnn*	Busy flag (*f*) returns in D7 (1= Busy), D6:D0 contains address counter value (1 us)
Write Data	1	0	*nnnn nnnn*	Data *nnnnnnnn* written at current DD RAM address (46 us)
Read Data	1	1	*nnnn nnnn*	Data *nnnnnnnn* at current address location in DD RAM is returned (46 us)

© Thomson/Delmar Learning, 2005

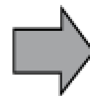
LCD Interface Code

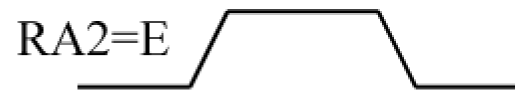
These macros improve code portability if you need to use different pins for the LCD interface.

```
// macros to isolate interface dependencies
#define EHIGH            RA2=1
#define ELOW             RA2=0
#define E_OUTPUT         {ADCON1=0x06;TRISA2 = 0;}
#define RSHIGH           RA5=1
#define RSLOW            RA5=0
#define RS_OUTPUT        TRISA5 = 0
#define RWHIGH           RA3=1
#define RWLOW            RA3=0
#define RW_OUTPUT        TRISA3 = 0
#define BUSY_FLAG        RB3
#define DATA_DIR_RD      TRISB = 0xFF
#define DATA_DIR_WR      TRISB = 0x00
#define OUTPUT_DATA(x) {PORTB = x;}
```

Macros to isolate code from port pins used to implement LCD interface

```
void epulse(void){
  DelayUs(1);  EHIGH;  DelayUs(1);
  ELOW; DelayUs(1);
}
```



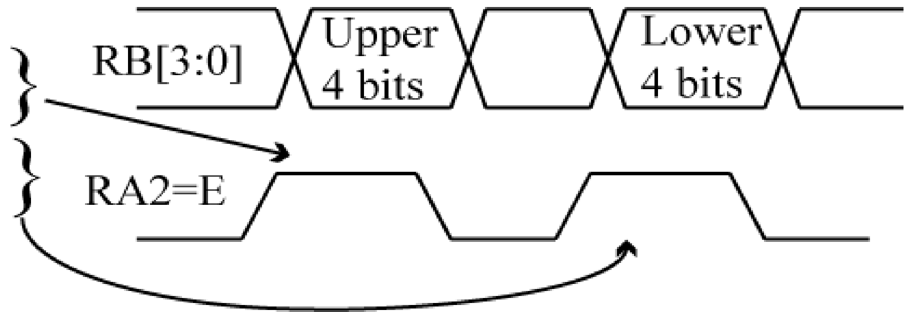


© Thomson/Delmar Learning, 2005

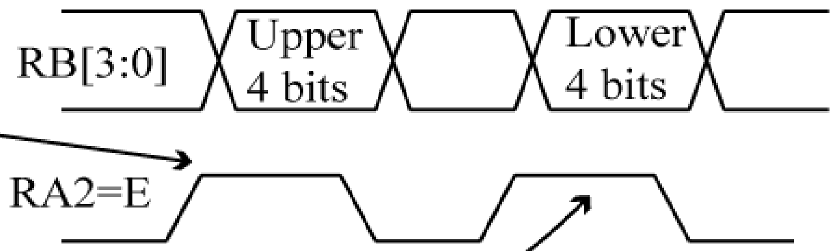
write data or command to LCD

```
void lcd_write(
    unsigned char cmd, unsigned char data_flag,
    unsigned char chk_busy, unsigned char dflag){
  char bflag,c;
  if (chk_busy) {
    RSLOW;         //RS = 0 to check busy
    // check busy
    DATA_DIR_RD;   //set data pins all inputs
    RWHIGH;        // R/W = 1, for read
    do {
      EHIGH; DelayUs(1);  // upper 4 bits
      bflag = BUSY_FLAG;
      ELOW; DelayUs(1);
      epulse();
    } while(bflag);
  } else {
    DelayMs(10); // don't use busy, just delay
  }
  DATA_DIR_WR;
  if (data_flag)  RSHIGH;   // RS=1, data byte
  else     RSLOW;   // RS=0, command byte
  // device is not busy
  RWLOW;         // R/W = 0, for write
  c = cmd >> 4;  // send upper 4 bits
  OUTPUT_DATA(c);
  epulse();
  if (dflag) {
    c = cmd & 0x0F;  //send lower 4 bits
    OUTPUT_DATA(c);
    epulse();
  }
}
```

RA5 = RS = 0 (command)
RA3 = R/W# = 1 (read)
Busy flag in RB3



RA5 = RS = data_flag
RA3 = R/W# = 0 (write)



© Thomson/Delmar Learning, 2005

Comments on lcd_write

- ‘cmd’ is 8-bit data to write to LCD
- ‘data_flag’ – if ‘1’, then this is a data byte (set RS=1), if ‘0’, then this is a command byte (set RS=0).
- If ‘chk_busy’ is nonzero, then read status flag from LCD and wait until LCD is non busy. If ‘chk_busy’ is zero, don’t check status flag, just delay.
- if ‘dflag’ is zero, then only send upper four bits of ‘cmd’ (only needed after power up when LCD is in 8-bit mode).

```
void lcd_init(void) {
 DelayMs(50);  //wait for device to reset on power-on, pessimistic
 lcd_write(0x20,0,0,0);  // 4 bit interface
 lcd_write(0x28,0,0,1);  // 2 line display, 5x7 font
 lcd_write(0x28,0,0,1);  // repeat
 lcd_write(0x06,0,0,1);  // enable display
 lcd_write(0x0C,0,0,1);  // turn display on; cursor and blink is off
 lcd_write(0x01,0,0,1);  // clear display, move cursor to home
 DelayMs(3);   // wait for busy flag to be ready
}
```

← Initialize the display

```
// send 8 bit char to LCD
void putch (char c) {
    lcd_write(c,1,1,1);
}
```

Define `putch()` as a character write to the LCD so that `printf` can be used for formatted output.

lcd_init() initializes the LCD module.

putch() is used by *printf()* function to write one character; define this as write one character to LCD.

© Thomson/Delmar Learning, 2005

```
main(void){
  // configure control pins as outputs
  // initialize as low
  E_OUTPUT;  RS_OUTPUT;  RW_OUTPUT;
  ELOW; RSLOW;  RWLOW;
  lcd_init ();
  printf("******Hello, my name is Bob********");
  lcd_write(0xC0,0,1,1);  // cursor to 2nd line
  printf("-----these lines are moving!-------");
  while(1) {
      // shift left
      lcd_write(0x18,0,1,1);
      DelayMs(100);
      DelayMs(100);
      DelayMs(100);
  }
}
```

Configure control, initialize low

Write line 1,
Set address counter to first
location of line 2,
write line 2

Loop continually left shifts,
causing lines 1 and 2 to scroll across
the display, moving right to left.

characters move across screen, right to left because of shift loop

© Thomson/Delmar Learning, 2005

What do you have to know?

- Parallel port usage of PORTA, PORTB
- How to use the weak pullups of PORTB
- How N/P type transistors work
- How a Tri-state buffer works
- How an open-drain output works and what it is useful for.
- How to write C code for finite state machine description of LED/Switch IO.
- Strobed IO for LCD display