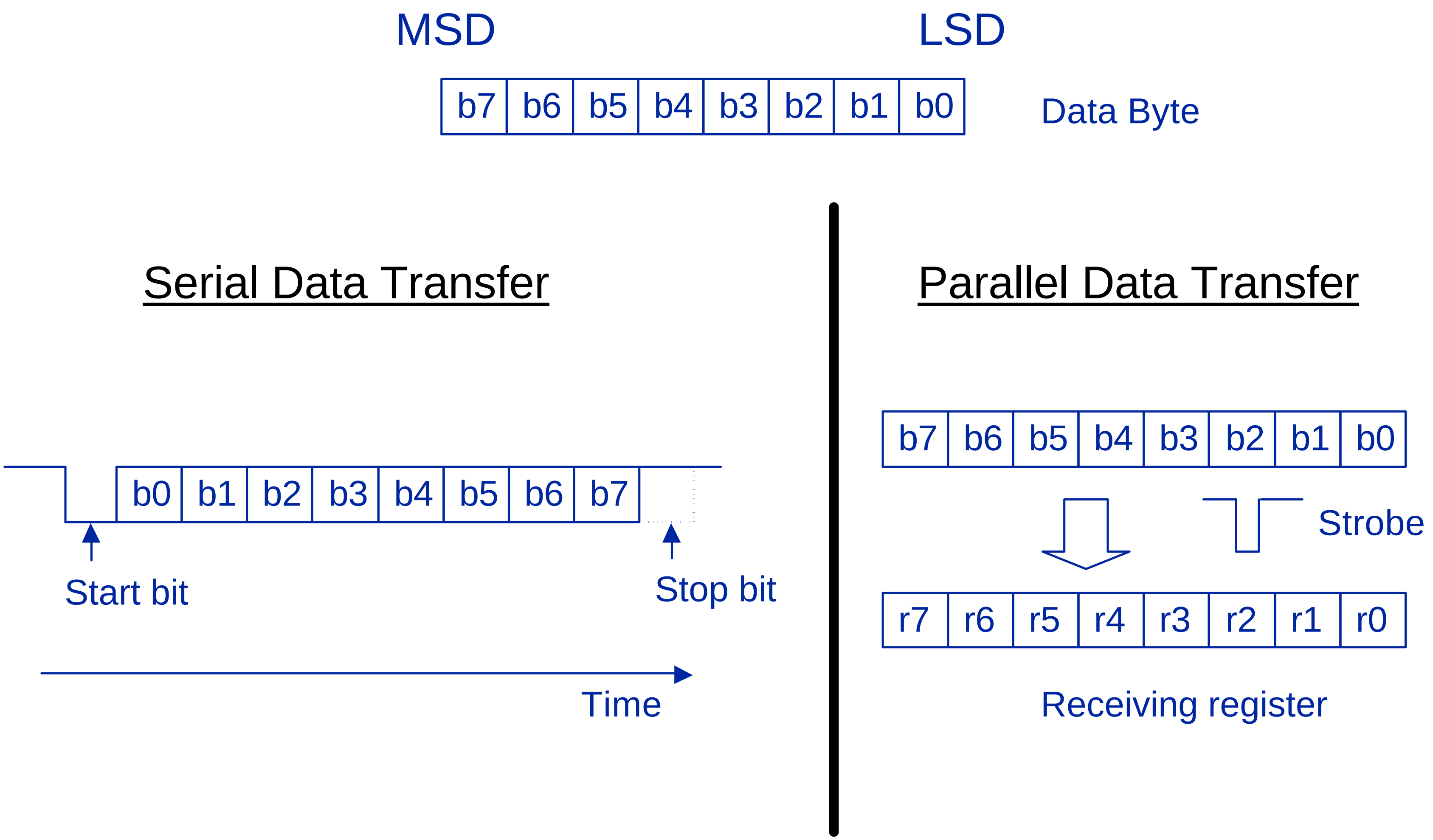
More PIC Programming

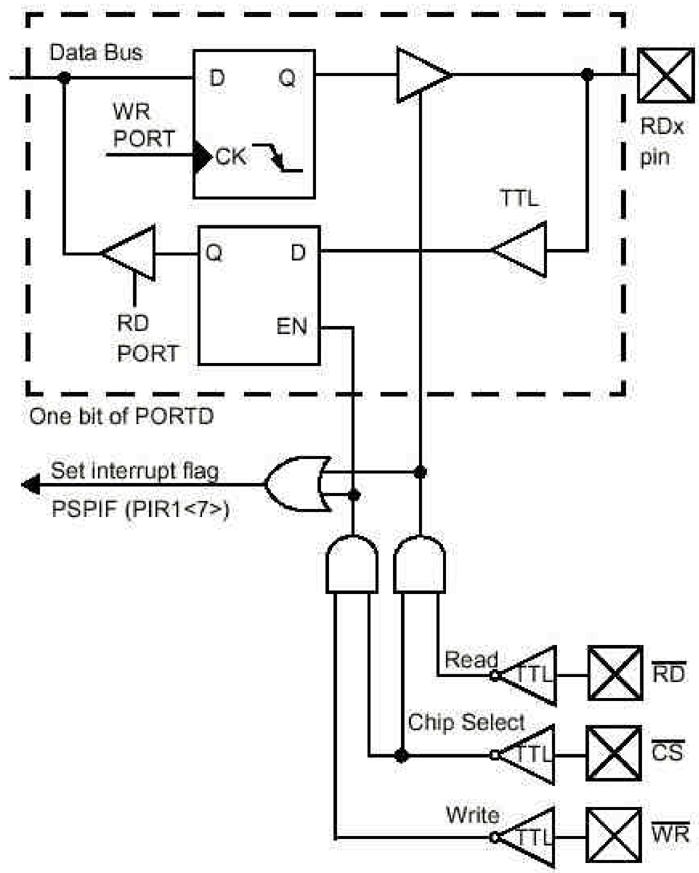
- Serial and parallel data transfer
- External busses
- Analog to digital conversion

Serial vs. Parallel Data Transfer



Parallel Slave Port

- It is asynchronously readable and writable by the external world through RDx, control input pin RE0/RD, and WR control input pin RE1/WR.
- Port can directly interface to an 8-bit microprocessor data bus.



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

Serial Input and Output

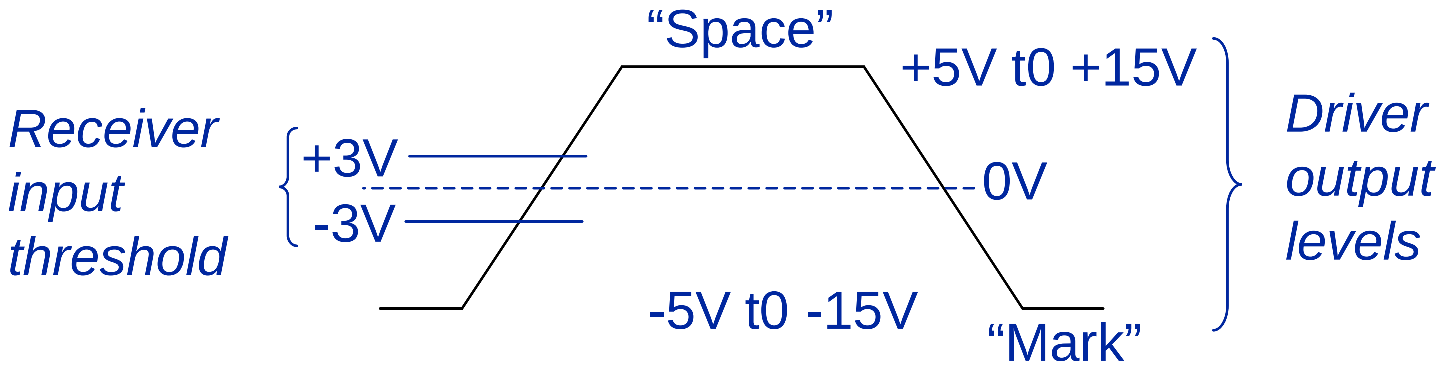
- **Any pin on the PIC can be configured as serial input or output**
- **Use the #USE RS232 directive to initialize serial port**

e.g.,

```
#use rs232(baud=9600, xmit=PIN_A3, rcv=PIN_A2)
/* sets baud rate to 9600,
   sets transmit pin to Port A, bit 3
   sets receive pin to Port A, bit 2 */
```

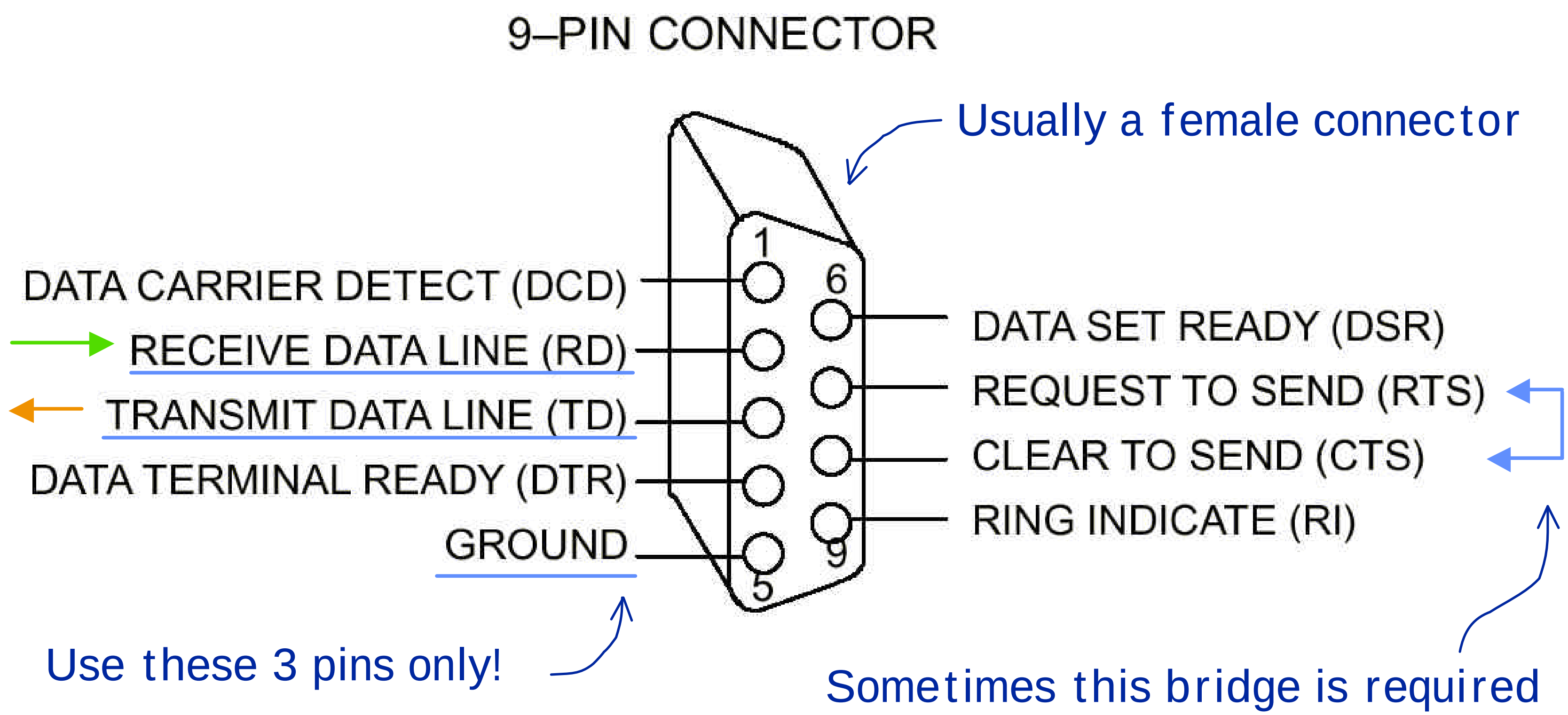
RS-232 Logic Level Specifications

- **Logic High (“Mark”) = anywhere from -5 V to -15 V**
- **Logic Low (“Space”) = anywhere from +5 V to +15 V**
- **Logic Threshold = +3V for low-to-high,**
  -3V for high-to-low
- **Standard defines maximum data rate of 20 k bit/sec**
  - **Though some of today’s devices guarantee up to 250 k bit/sec.**
- **Maximum load capacitance: 2500 pF**



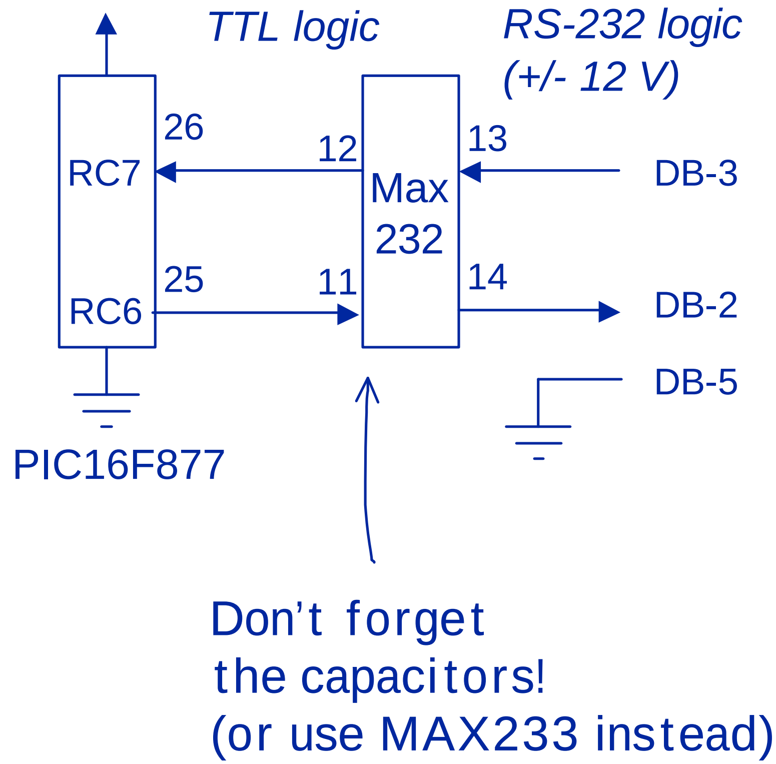
PC Serial Interface Cable

- **Although RS-232 specifies a 25-pin connector, the most popular implementation uses a 9-pin connector instead.**



Serial Interface Circuit to PC: Method #1

- **Use a RS-232 interface circuit**
  - **MAX232(A) requires external capacitors**
  - **MAX233 no external capacitors required**
  - **Protects PIC**



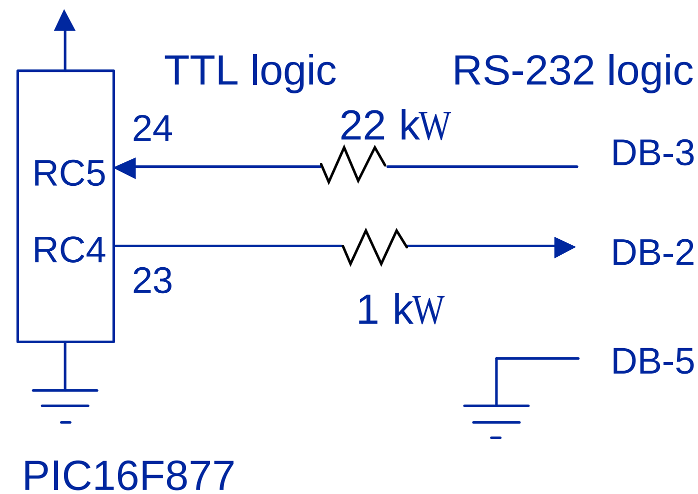
Use the directive

```
#use rs232(baud=9600, xmit=PIN_C6,rcv=PIN_C7)
```

NOTE: use this method if you want to use USART interrupts with CCS compiler.

Serial Interface Circuit to PC: Method #2

- **Use resistors for interfacing**
  - **Internal clamping diodes limit the +/- 12 V RS232 logic to 0, 5 V**
  - **The 22 kW resistor limits the input current to within safe ranges**
- **Cheaper, easier to build**
  - **Less components required**
  - **PIC is more susceptible to damage**



Use the directive

```
#use rs232(baud=9600, xmit=PIN_C4,rcv=PIN_C5, INVERT)
```

NOTE: this method does not allow USART interrupts with CCS compiler.

Serial Interfacing in C

- Setting up a serial protocol

- Set up TX,RX hardware

```
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
```

- Interrupt called whenever a byte is in the receive register

```
#int_rda receive_handler()  {}
```

- To enable, call `enable_interrupts(INT_RDA);`

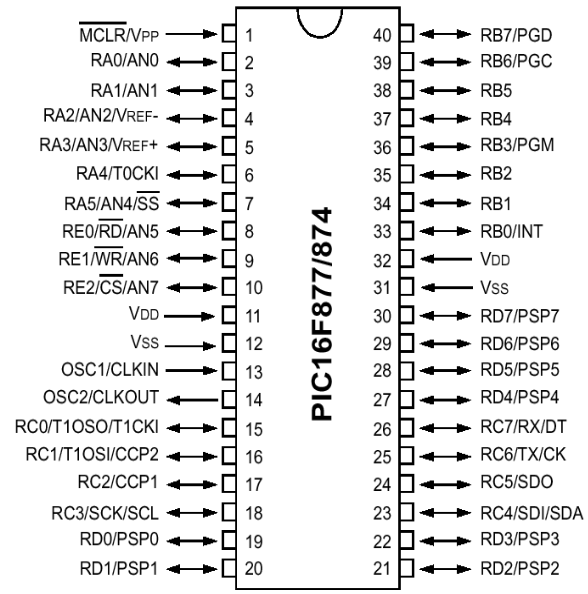
- Interrupt called whenever transfer register is cleared. This happens as soon as byte is written to output register (allows maximum data transfer)

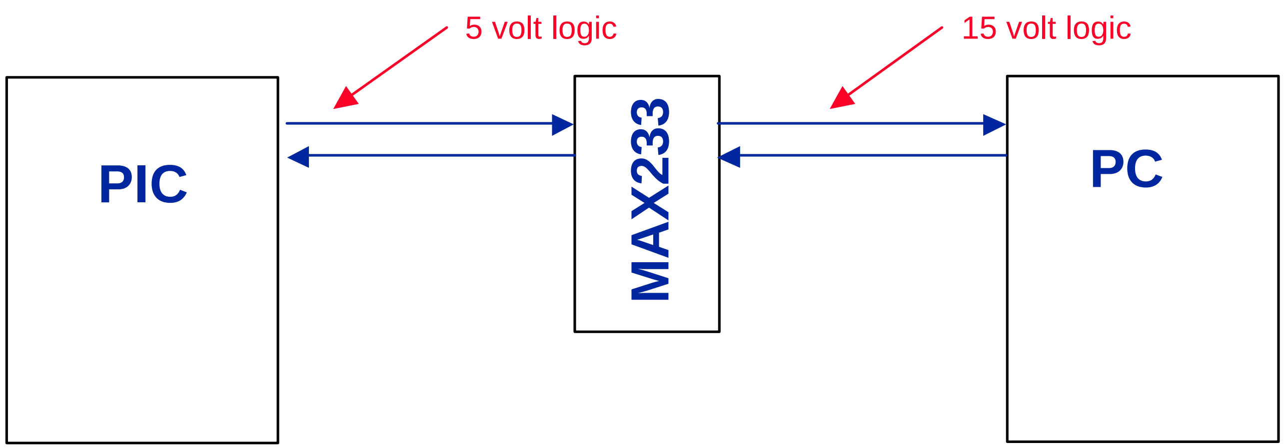
```
#int_tbe t_handler() { }
```

- To enable, call `enable_interrupts(INT_TBE);`

PC Interface

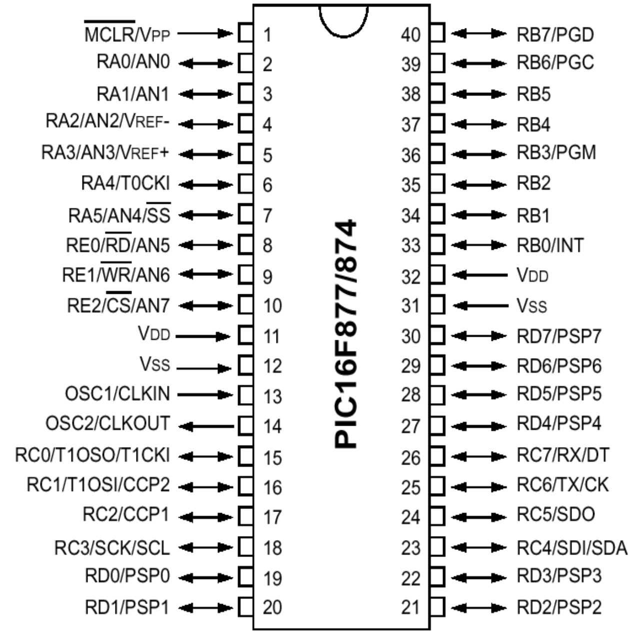
- **RS232**
  - **Can be performed in software and hardware**
  - **Hardware supports interrupts**
  - **Received bytes stored in temp buffer**
  - **Transmit bytes sent out as soon as channel open**





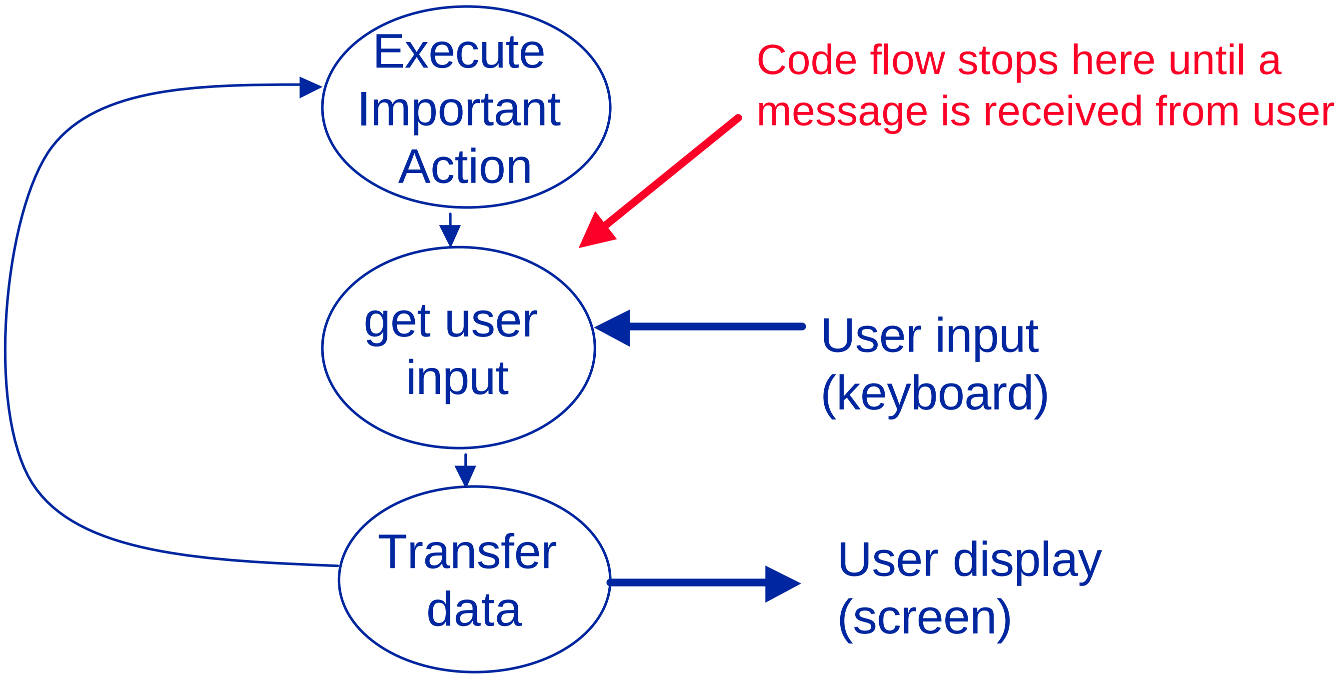
RS232

```
#use rs232(baud=4800, xmit=PIN_C6, rcv=PIN_C7)

#int_tbe t_handler() {
  if(t_head == t_tail) disable_interrupts(INT_TBE);
  else {
    man_putc(t_buffer[t_tail]);
    t_tail++; if(t_tail == T_BUFFER_SIZE) t_tail = 0;
  }
}

#int_rda receive_handler()  {
   rxbyte = man_getc();
   HandleCharacter();
   rxcharacter = true;
}

void send_byte(byte txbyte) {
  t_buffer[t_head] = txbyte;
  t_head++; if(t_head == T_BUFFER_SIZE) t_head = 0;
  enable_interrupts(INT_TBE);
}

void put_receive_byte (byte input) {
  r_buffer[r_head]=input;
  r_head
  if(r_head == R_BUFFER_SIZE) r_head = 0;
}
```



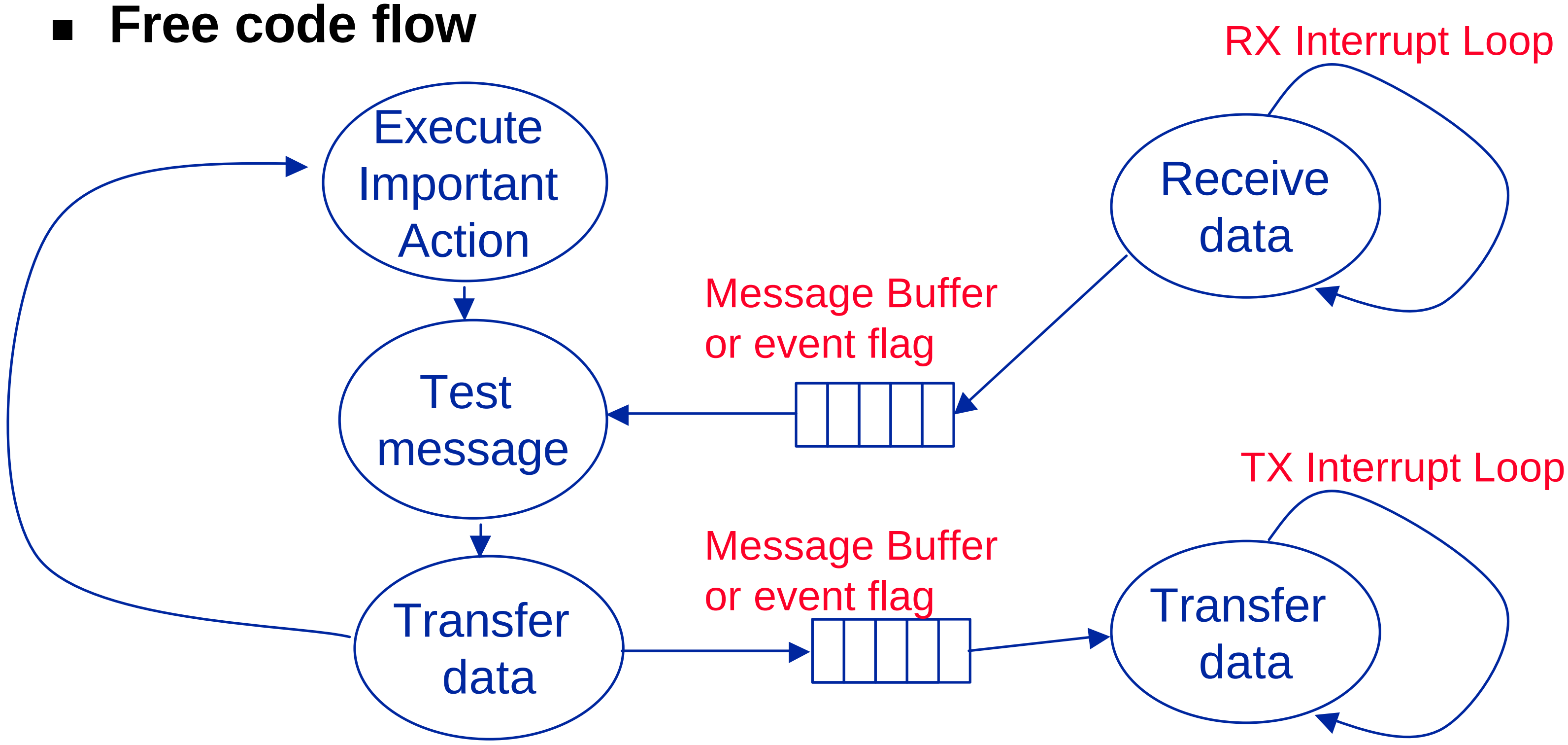
RS232 - In Line

- **PUTC / GETC / PUTS / GETS**
  - **Functions to allow passing information back and forth to PC via serial**
  - **putc, getc, puts, gets are blocking functions**
  - **OK for simple code flow**



RS232 - Interrupts

- **PUTC / GETC / PUTS / GETS**
  - **Functions to allow passing information back and forth to PC via serial**
  - **Interrupts handle monitoring of communication channel**
  - **Free code flow**



RS232 – Software with Interrupts

Define RX, TX pins to hardware

```
#use rs232(baud=4800, xmit=PIN_C6,
rcv=PIN_C7)
```

Define RX, TX Buffers and ptrs to head and tail

```
byte r_buffer[R_BUFFER_SIZE]; // receive
buffer
byte r_head;  // head of the queue
byte r_tail;  // tail of the queue
byte t_buffer[T_BUFFER_SIZE];  //
transmit buffer
byte t_head;  // head of the transmit
queue
byte t_tail;  // tail of the transmit
queue
```

HandleCharacter(rxbyte);
Function that appends new character to message string and tests whether it is complete

TX interrupt allows next byte to be sent as soon as previous byte clears TX

```
#int_tbe t_handler() {
  if(t_head == t_tail)
disable_interrupts(INT_TBE);
  else {
    putc(t_buffer[t_tail]);
    t_tail++;
    if(t_tail == T_BUFFER_SIZE) t_tail = 0;
  }
}
```

RX interrupt - signals when new byte is in receive buffer. Byte passed to state machine to concatinate and test

```
#int_rda receive_handler(){
   rxbyte = getc();
   HandleCharacter(rxbyte);
   rxcharacter = true;
}
```

Appending characters to transmit buffer (called faster than info being sent)

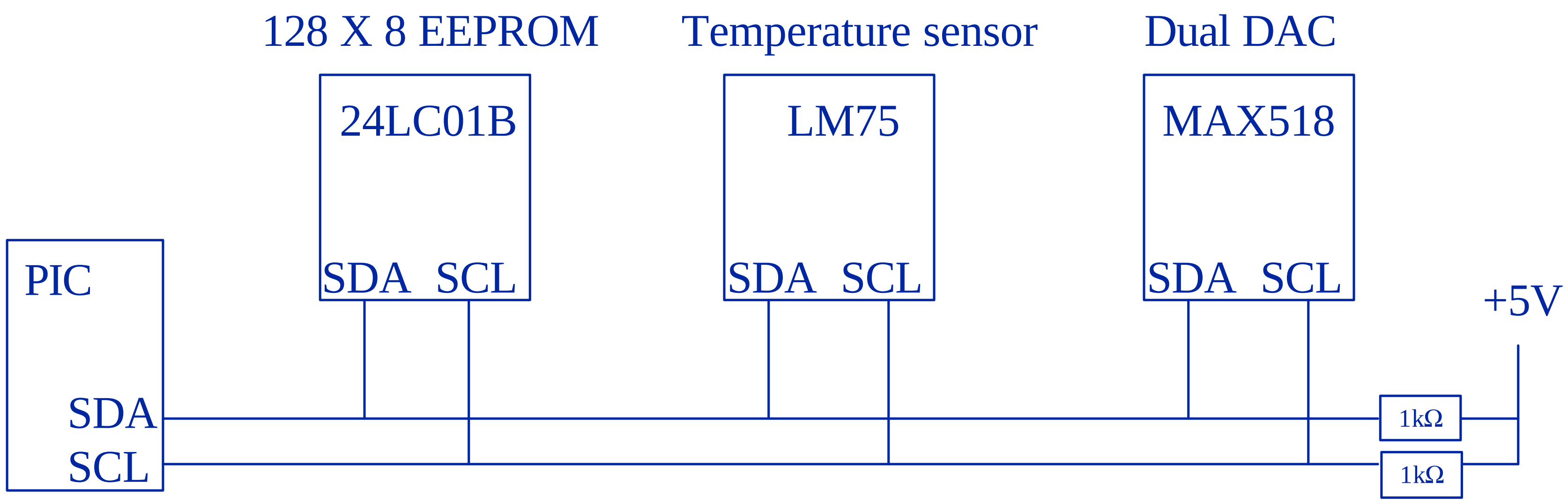
```
void send_byte(byte txbyte) {
  t_buffer[t_head] = txbyte;
  t_head++;
  if(t_head == T_BUFFER_SIZE) t_head = 0;
  enable_interrupts(INT_TBE);
}
```

Master Synchronous Serial Port Module (MSSP)

- Serial interface for communicating with other devices
  - Serial EEPROMs
  - Shift registers
  - Display drivers
  - A/D converters
- Two modes:
  - Serial Peripheral Interface (SPI)
  - Inter-Integrated Circuit (I^2C)

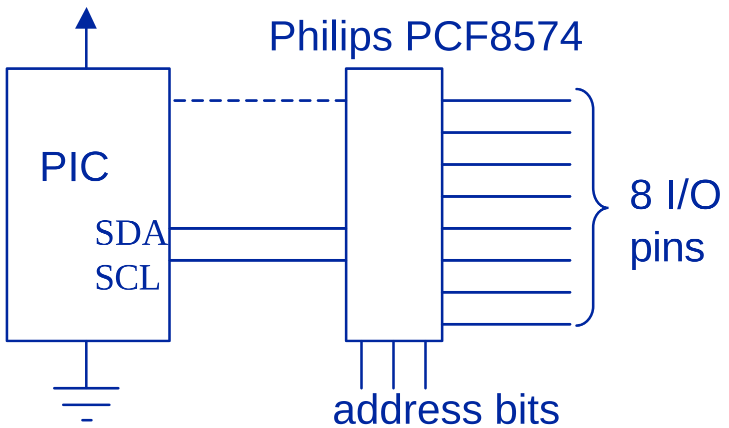
I^2C Bus for Peripheral Chip Access

- 2-wire interface
- Each device is assigned to a different address



I/O Expansion

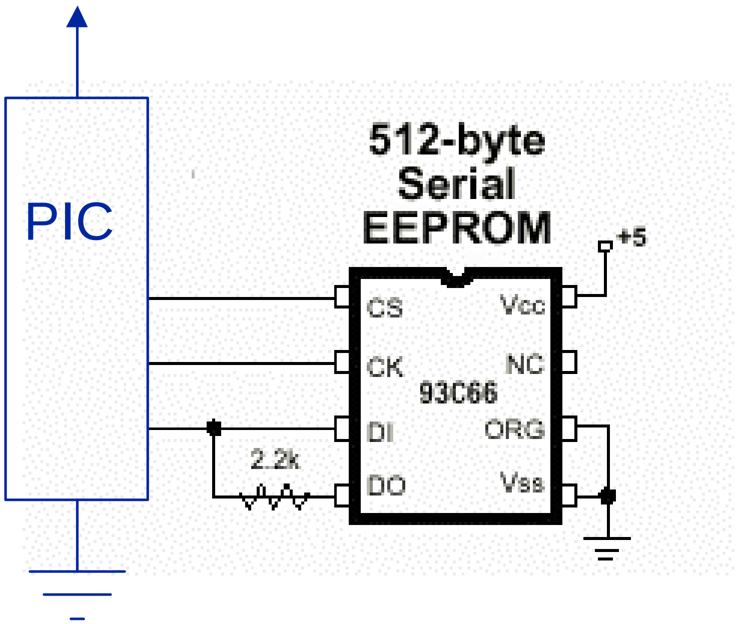
- The Philips I/O Expander allows the expansion of 8 I/O pins to the PIC
- I^2C 2-wire interface used
- Optionally, can generate an interrupt when any of the 8 I/O lines changes state
- Addressable, allowing up to seven additional devices to share the same data busses



http://www.phanderson.com/PIC/PICC/CCS_PCM/8574_1.html

External Memory Expansion

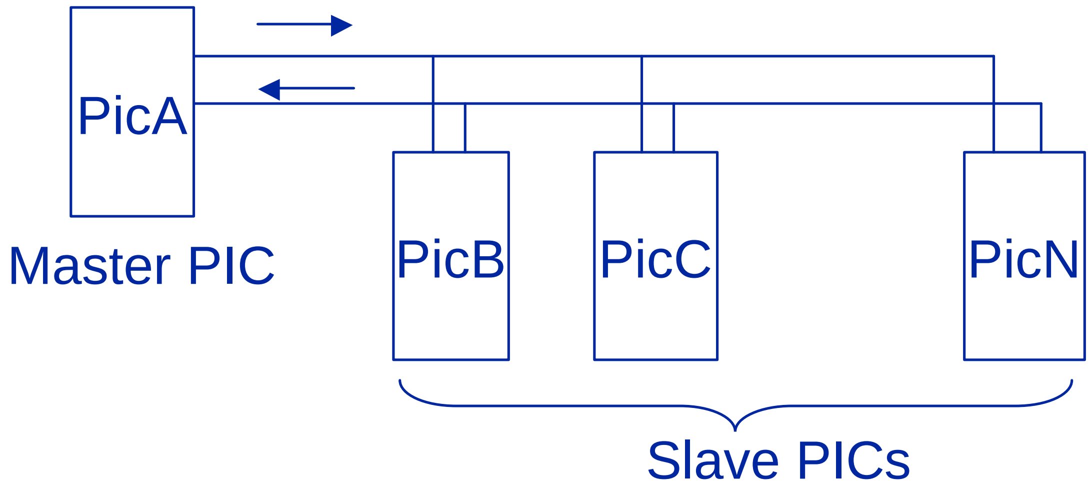
- External memory can be added via a 2- or 3-wire interface
- Slow write speed, fast read speed
- Data can be written via PIC or via external device
- Data is non-volatile



See the sample code EX_EXTEE.C

PIC Networking

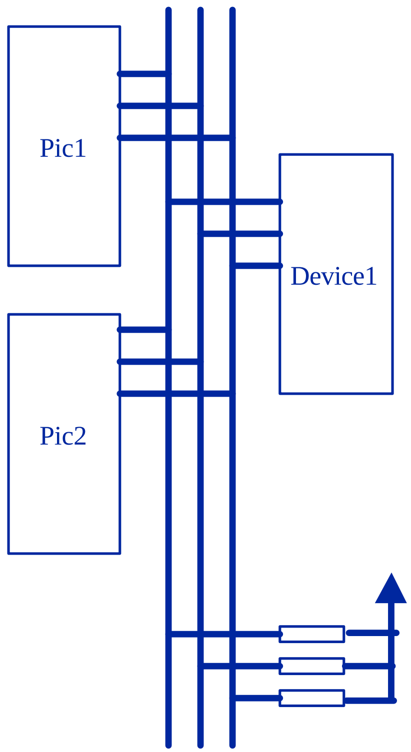
- PICs communicate over 2-wire bus (TX/RX)
- Master PIC synchronizes communications by initiating either command or query message
- Slave PICs only respond when queried
- Target PIC can be identified as part of message (token based) or via external lines
- Advanced communication modes available to allow any PIC to generate communications



http://ccsinfo.com/ep3.html

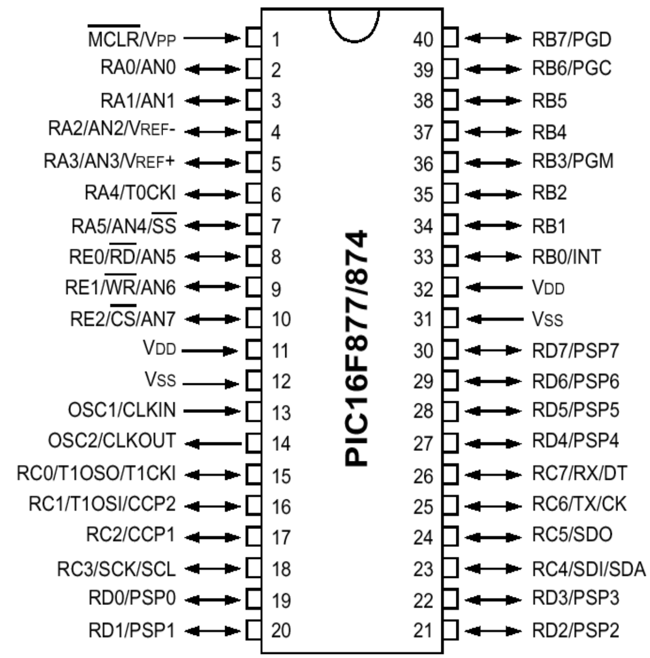
Port B Communications Bus Configuration

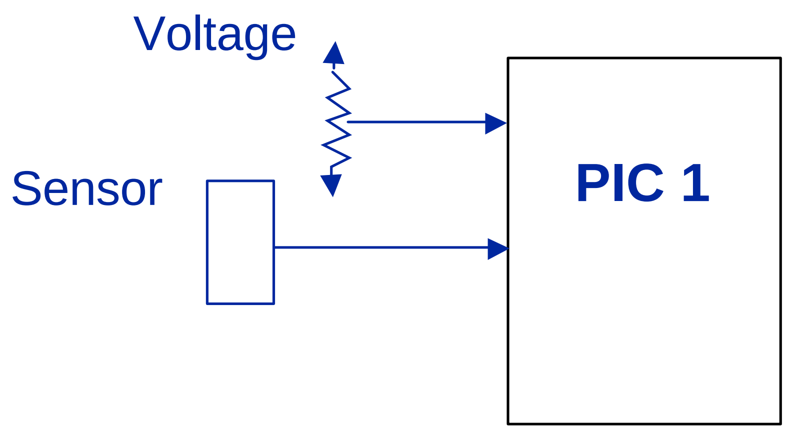
- **Port B can be configured as weak pullup**
- **Weak pullups allow multiple devices to drive a common bus or data line**
- **3 states**
  - **high**
  - **low**
  - **high impedence**
- **Requires external pullup resistor**



Analog to Digital Converter (ADC)

- **ADC**
  - **Measure voltage from up to 8 sources**
  - **10 bit resolution**
  - **1MHz max clock rate**
  - **Acquisition time ~ 12 – 20 µs (slow for audio)**
  - **Can dedicate 2 lines for input of high and low voltage references to specify the range**





Analog to Digital Converter (ADC)

- 10-bit resolution, 8 input channels
- Alternate function of Port A.
  - Port pins can be configured as analog inputs or digital I/O
- Two control registers:
  - ADCON0 controls the operation of the A/D module
  - ADCON1 configures the functions of the port pins

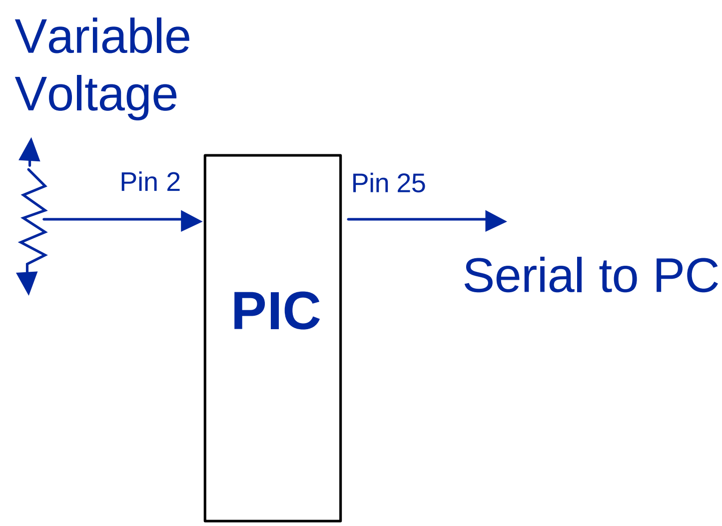
ADC Sample Code

```
#include <16F877.H>
#use delay(clock=4000000)
#use rs232(baud=9600,xmit=PIN_A3,rcv=PIN_A2)

  main() {
    int i,value,min,max;
    printf("Sampling:");
    setup_port_a( ALL_ANALOG );
    setup_adc( ADC_CLOCK_INTERNAL );
    set_adc_channel( 0 );
    do {
      min=255;
      max=0;
      for(i=0;i<=30;++i) {
        delay_ms(100);
        value = Read_ADC();
        if(value < min) { min=value; }
        if(value > max) { max=value; }
      }
      printf("\n\rMin: %2X  Max:
   %2X\r\n",min,max);
    } while (TRUE);
  }
```

Set ADC pins as analog read

Use internal clock

Which ADC channel to convert



Using the ADC

1. Configure the A/D module:

- **Configure analog pins / voltage reference / and digital I/O (ADCON1)**
- **Select A/D input channel (ADCON0)**
- **Select A/D conversion clock (ADCON0)**
- **Turn on A/D module (ADCON0)**

2. Configure A/D interrupt (if desired):

- **Clear ADIF bit**
- **Set ADIE bit**
- **Set GIE bit**

3. Wait the required acquisition time.

4. Start conversion:

- **Set GO/$\overline{\text{DONE}}$ bit (ADCON0)**

5. Wait for A/D conversion to complete, by either:

- Polling for the GO/$\overline{\text{DONE}}$ bit to be cleared

  OR

- Waiting for the A/D interrupt

6. Read A/D Result register pair

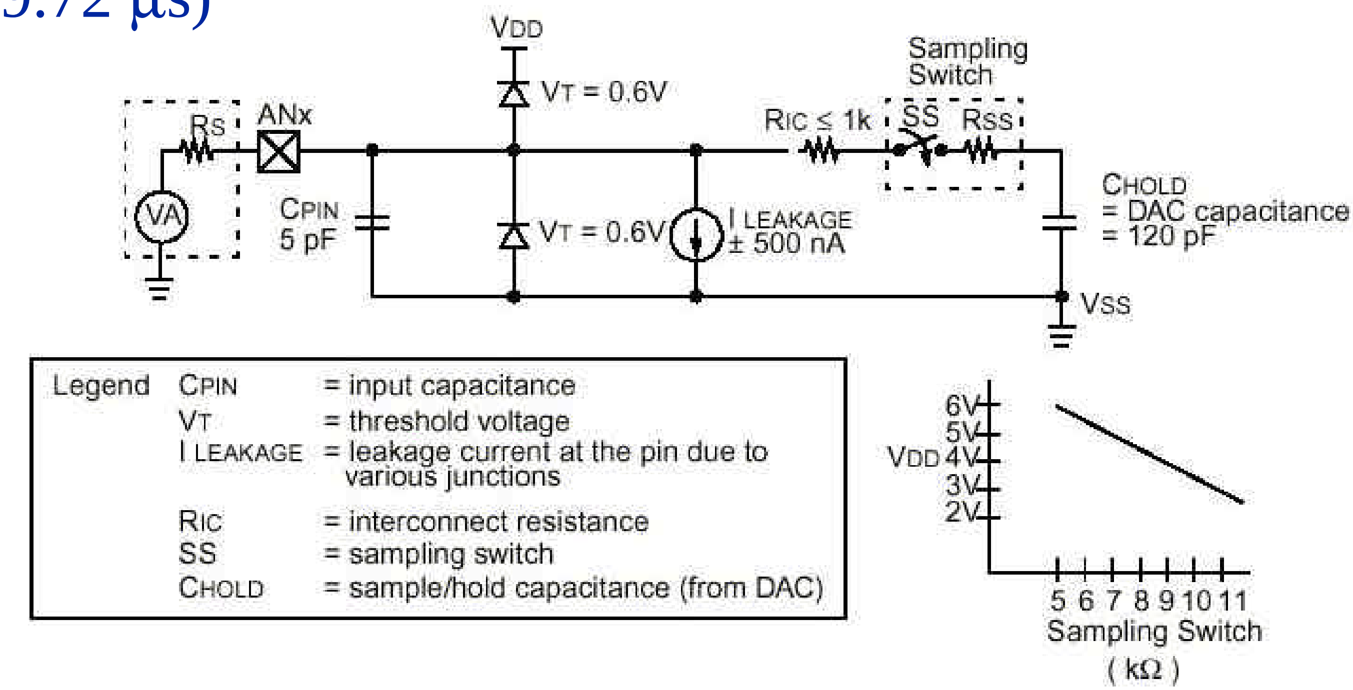
- (ADRESH:ADRESL), clear bit ADIF if required.

7. For next conversion, go to step 1 or step 2 as required. The A/D conversion time per bit is defined as TAD. A minimum wait of 2TAD is required before next acquisition starts.

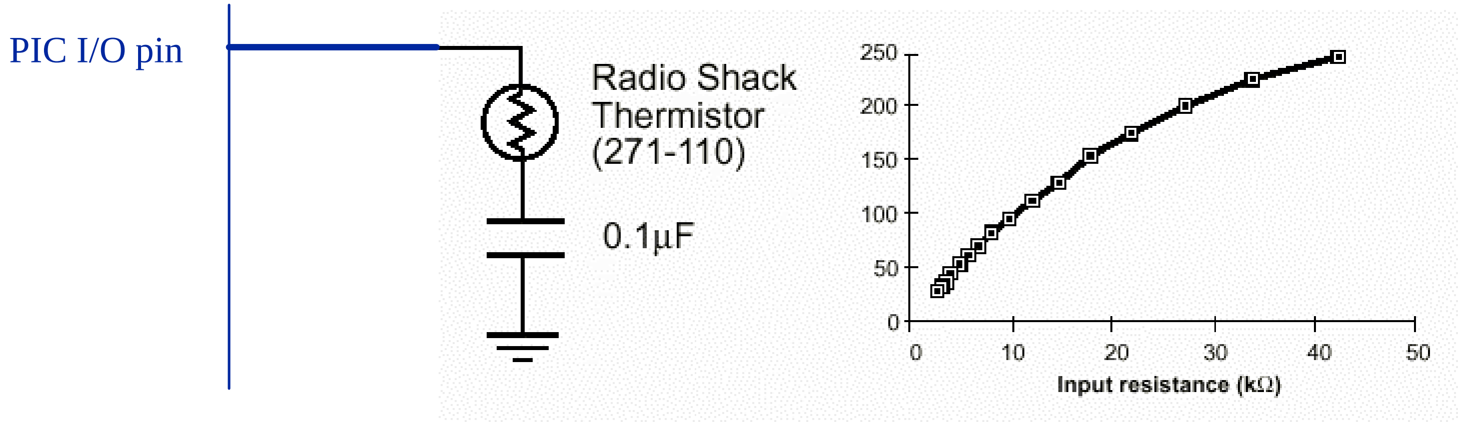
ADC (cont.)

Acquisition requirements:

The charge holding capacitor (C_{HOLD}) must be allowed to fully charge to the input channel voltage level. ($T_C \approx 16.47\mu s$; $T_{ACQ} \approx 19.72\ \mu s$)



Poor Man’s ADC



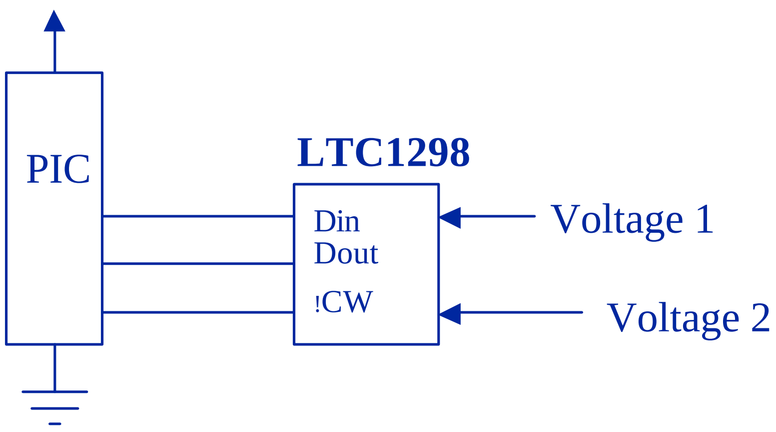
- Allows the PIC to read the value of a resistive element with a single pin.
- Works by measuring the RC time constant of the circuit.
- Drawback is that the mapping is non-linear but can be accomplished with a lookup table.

Sequence:

- switch pin to output and drive to logic high
- wait a few ms for capacitor to charge
- start internal counter
- switch pin to input and poll pin
- if pin is logic high (voltage > 2.5v) increment counter
- if pin is logic low, exit and return count

External ADC interface

- An external Analog to Digital Converter interface can be implemented using a 3-wire connection
- Acquisition time can be much faster than the built-in ADC

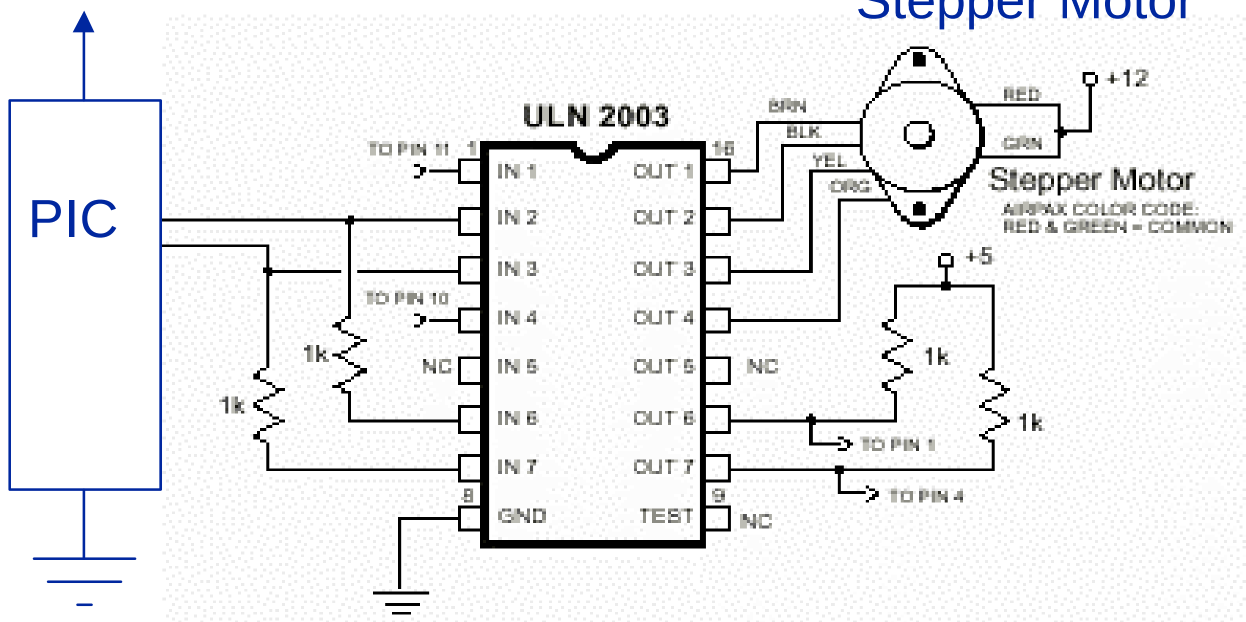


See the sample code EX_AD12.C

Stepper Motor Control

- **Stepper motors can be used for high precision motion control**
- **The PIC generates the necessary timing of the four stepper coils**

Stepper Motor



High current driver

http://ccsinfo.com/ep1.html