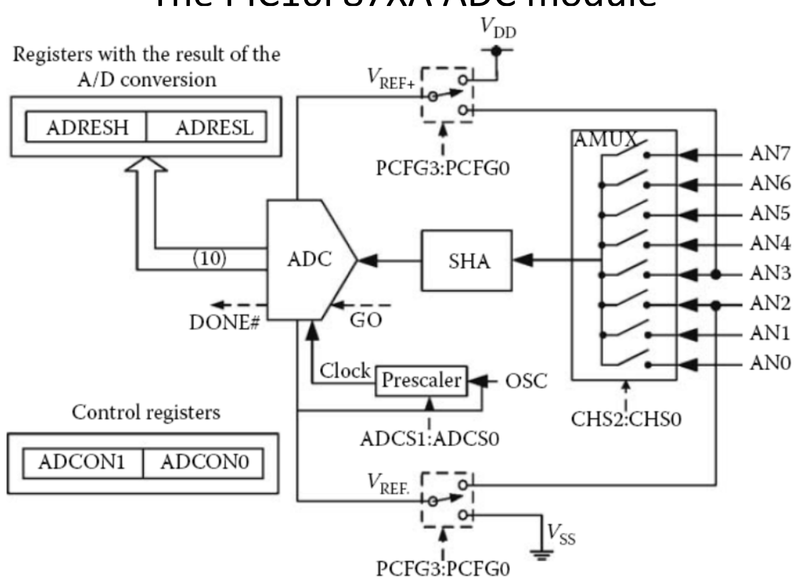
Analog input with PIC 16 Series

Device	Pins	Features
16F873A 16F876A	28	3 parallel ports, 3 counter/timers, 2 capture/compare/PWM, 2 serial, **5 10-bit ADC,** 2 comparators
16F874A 16F877A	40	5 parallel ports, 3 counter/timers, 2 capture/compare/PWM, 2 serial, **8 10-bit ADC,** 2 comparators

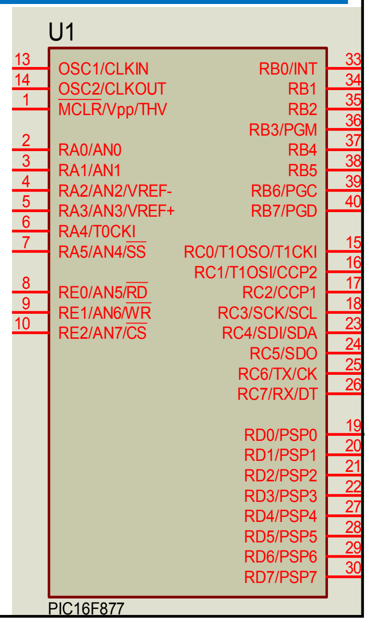
The PIC16F87XA ADC module



Functional blocks for the A/D converter in medium-end PIC microcontrollers. A/D, successive approximation analog-to-digital converter; SHA, sample-and-hold amplifier; AMUX, analog multiplexer.

Analog Setup

- The PIC 16F877 has eight analog inputs: RA0, RA1, RA2, RA3, RA5, RE0, RE1, RE2 being renamed AN0 to AN7.



CCS C Analog Input Functions

Action	Description	Example
ADC SETUP	Initialize ADC	setup_adc(ADC_CLOCK_INTERNAL);
ADC PINS SETUP	Initialize ADC pins	setup_adc_ports(RA0_ANALOG);
ADC CHANNEL SELECT	Select ADC input	set_adc_channel(0);
ADC READ	Read analog input	inval=read_adc();

CCS C Analog Input Functions

setup_adc_ports( )

Syntax:	setup_adc_ports (***value***)
Parameters:	***value*** - a constant defined in the devices .h file
Returns:	undefined
Function:	Sets up the ADC pins to be analog, digital, or a combination and the voltage reference to use when computing the ADC value. The allowed analog pin combinations vary depending on the chip and are defined by using the bitwise OR to concatenate selected pins together. Check the device include file for a complete list of available pins and reference voltage settings. The constants ALL_ANALOG and NO_ANALOGS are valid for all chips. Some other example pin definitions are: • ANALOG_RA3_REF- All analog and RA3 is the reference • RA0_RA1_RA3_ANALOG- Just RA0, RA1 and RA3 are analog

CCS C Analog Input Functions

set_adc_channel( )

Syntax:	set_adc_channel (***chan***)
Parameters:	***chan*** is the channel number to select. Channel numbers start at 0 and are labeled in the data sheet AN0, AN1
Returns:	undefined
Function:	Specifies the channel to use for the next read_adc() call. Be aware that you must wait a short time after changing the channel before you can get a valid read. The time varies depending on the impedance of the input source. In general 10us is good for most applications. You need not change the channel before every read if the channel does not change.
Availability:	This function is only available on devices with A/D hardware.
Requires:	Nothing

Examples:	`set_adc_channel(2);`
	`delay_us(10);`
	`value = read_adc();`

Example Files:	ex_admm.c
Also See:	read_adc(), setup_adc(), setup_adc_ports(), ADC Overview

read_adc( )

Syntax: value = read_adc ([*mode*])

Parameters: ***mode*** is an optional parameter. If used the values may be:
ADC_START_AND_READ (continually takes readings, this is the default)
ADC_START_ONLY (starts the conversion and returns)
ADC_READ_ONLY (reads last conversion result)

Returns: Either a 8 or 16 bit int depending on #DEVICE ADC= directive.

Function: This function will read the digital value from the analog to digital converter. Calls to setup_adc(), setup_adc_ports() and set_adc_channel() should be made sometime before this function is called. The range of the return value depends on number of bits in the chips A/D converter and the setting in the #DEVICE ADC= directive as follows:

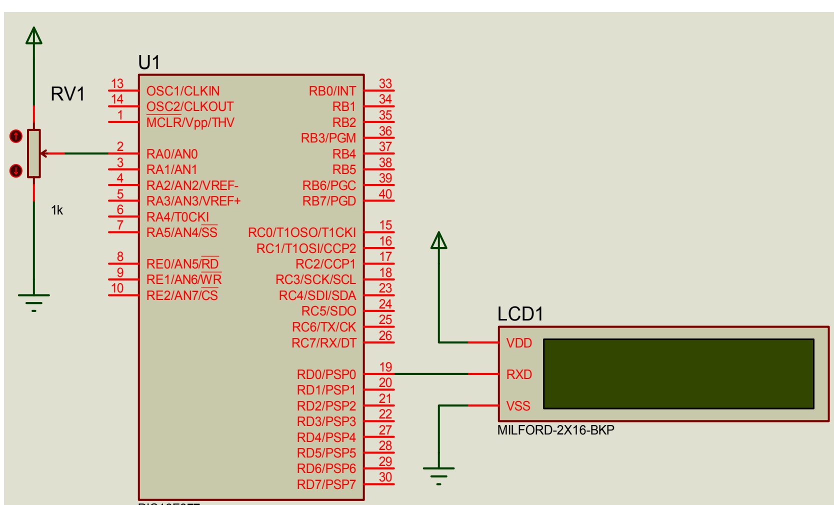
#DEVICE	8 bit	10 bit	11 bit	12 bit	16 bit
ADC=8	00-FF	00-FF	00-FF	00-FF	00-FF
ADC=10	x	0-3FF	x	0-3FF	x
ADC=11	x	x	0-7FF	x	x
ADC=16	0-FF00	0-FFC0	0-FFEO	0-FFF0	0-FFFF

Note: x is not defined

Availability: This function is only available on devices with A/D hardware.

Requires: Pin constants are defined in the devices .h file.

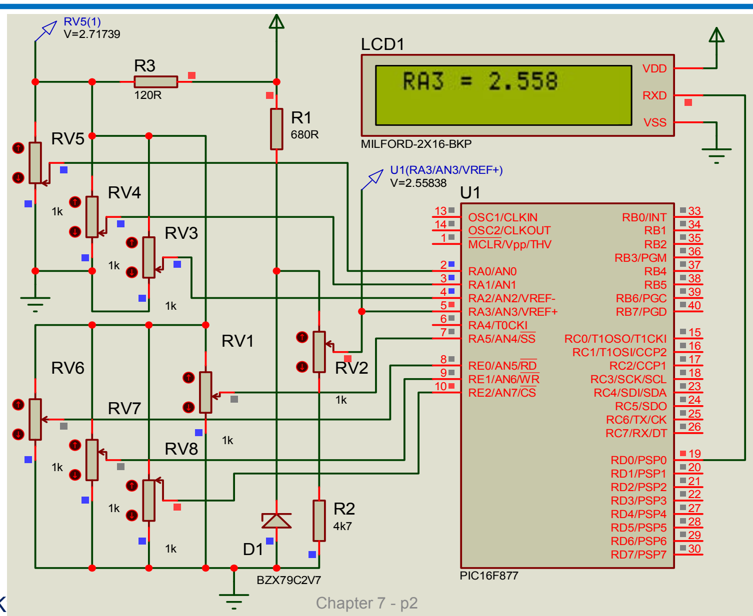
Example



Example

```
#include "16F877A.h"
#device ADC=8          // 8-bit conversion

#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_D0, rcv=PIN_D1)   // LCD output

void main() {
    int vin0;                              // Input variables
    setup_adc(ADC_CLOCK_INTERNAL);         // ADC clock
    setup_adc_ports(ALL_ANALOG);           // Input combination
    set_adc_channel(0);                    // Select RA0

    for(;;) {
        delay_ms(500);
        vin0 = read_adc();                 // Get input byte
        vin0 = (vin0/32)+0x30;             // Convert to ASCII

        putc(254); putc(1); delay_ms(10);  // Clear screen
        printf("Input = "); putc(vin0);    // Display input
    }
}
```


Voltage measurement

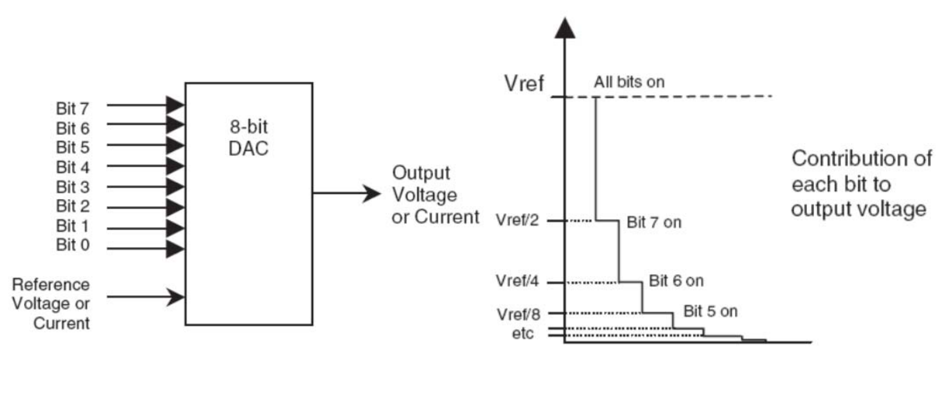


Voltage measurement

```
#include "16F877A.h"
#device ADC=10                                      // 10-bit operation
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_D0, rcv=PIN_D1)

void main()
{
    int     chan;
    float   analin[8], disvolts[8];                 // Array variables

    setup_adc(ADC_CLOCK_INTERNAL);                  // ADC clock source
    setup_adc_ports(AN0_AN1_AN2_AN4_AN5_AN6_AN7_VSS_VREF);   // ADC inputs

    while(1)
    {
        for(chan=0;chan<8;chan++)
        {   delay_ms(1000);
            set_adc_channel(chan);
            analin[chan] = read_adc();
            disvolts[chan] = (analin[chan])/400;        // Scale input
            putc(254);putc(1);delay_ms(10);             // Clear display
            printf(" RA%d = %4.3g",chan,disvolts[chan]);     // Display volts
        }
    }
}
```


7.3. Analog Output

Basic digital to analogue converter



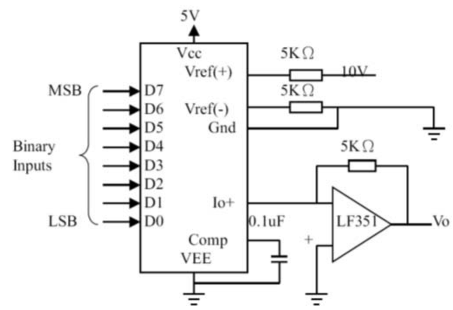
a) general DAC hardware b) output voltage

DAC

DAC – The Digital Input

There is no control signal required. However, the digital input must be kept stable when the digital-to-analogue conversion is taking place. Typically, the conversion time is less than 1µs.

Hardware Example: +10 V Output Digital-to-Analog Converter using DAC0808



Determine Vo :

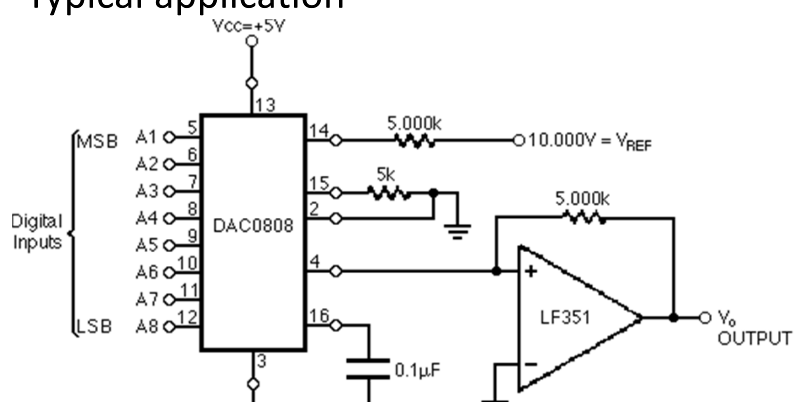
- Io+=Vref(+)/5KΩ * {D_7/2+D_6/4+D_5/8+D_4/16+D_3/32+D_2/64+D_1/128+D_0/256}
- Vref(+)/5KΩ = 10V/5KΩ = 2 mA
- Vo = 5KΩ*2mA*{D_7/2+D_6/4+D_5/8+D_4/16+D_3/32+D_2/64+D_1/128+D_0/256}
  =10 V *{D_7/2+D_6/4+D_5/8+D_4/16+D_3/32+D_2/64+D_1/128+D_0/256}

Example: What is the output voltage Vo if the binary inputs to the DAC mentioned above is 6Eh?
Answer :

Vo = 10V/256{$2^7D_7+2^6D_6+2^5D_5+2^4D_4+2^3D_3+2^2D_2+2^1D_1+2^0D_0$}
= 10V * 6Eh / 256 = 10 *110 / 256 = 4.3 V

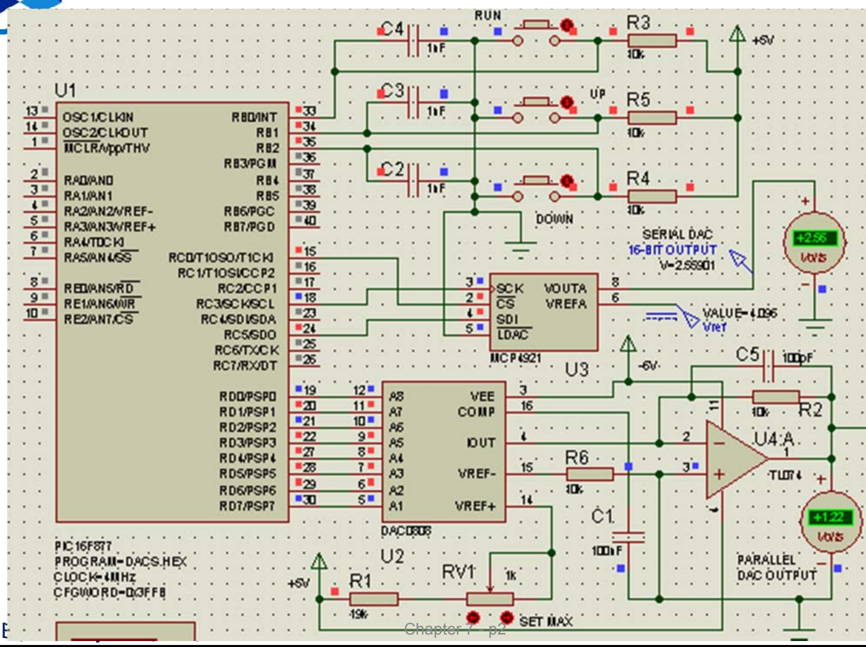
DAC Circuit

- Typical application

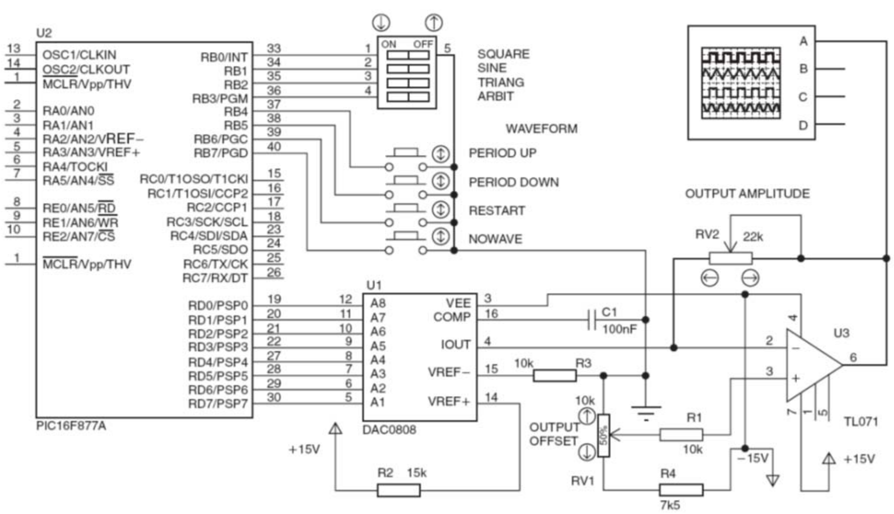


$$V_o = 10V\left(\frac{A1}{2} + \frac{A2}{4} + \cdots + \frac{A8}{256}\right)$$

Ex: DAC0808 & Serial DAC



Waveform Generator



Waveform Generator Source Code (1/3)

```
// DACWAVE.C MPB 5-7-07
// Outputs waveforms to DAC, simulation file DAC.DSN

#include "16F877A.H"
#include "MATH.H"
#use delay(clock=20000000)
#use fast_io(D)                    // High speed output functions

int n, time=10;
float step, sinangle;
float stepangle = 0.0174533;       // 1 degree in radians
int amp[91];                       // Output instant voltage array

// ISR to read push buttons *****************************************

#int_rb
  void change()
  {
    if(time!=255)
      {if (!input(PIN_B4)) time ++;}              // Increase period
    while(!input(PIN_B4));

    if(time!=0)
      {if (!input(PIN_B5)) time--;}               // Decrease period
    while(!input(PIN_B5));

    if(!input(PIN_B6))reset_cpu();                // Restart program
    if(!input(PIN_B7))for(n=0;n<91;n++)amp[n]=0;  // Zero output
  }
```


Waveform Generator Source Code (2/3)

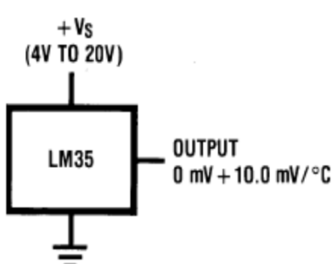
```
void setwave()  // Arbitrary waveform values **********************
{
  amp[0] =00;amp[1] =00;amp[2] =00;amp[3] =00;amp[4] =00;
  amp[5] =00;amp[6] =00;amp[7] =00;amp[8] =00;amp[9] =00;
  amp[10]=10;amp[11]=00;amp[12]=00;amp[13]=00;amp[14]=00;
  amp[15]=00;amp[16]=00;amp[17]=00;amp[18]=00;amp[19]=00;
  amp[20]=20;amp[21]=00;amp[22]=00;amp[23]=00;amp[24]=00;
  amp[25]=00;amp[26]=00;amp[27]=00;amp[28]=00;amp[29]=00;
  amp[30]=30;amp[31]=00;amp[32]=00;amp[33]=00;amp[34]=00;
  amp[35]=00;amp[36]=00;amp[37]=00;amp[38]=00;amp[39]=00;
  amp[40]=40;amp[41]=00;amp[42]=00;amp[43]=00;amp[44]=00;
  amp[45]=00;amp[46]=00;amp[47]=00;amp[48]=00;amp[49]=00;
  amp[50]=50;amp[51]=00;amp[52]=00;amp[53]=00;amp[54]=00;
  amp[55]=00;amp[56]=00;amp[57]=00;amp[58]=00;amp[59]=00;
  amp[60]=60;amp[61]=00;amp[62]=00;amp[63]=00;amp[64]=00;
  amp[65]=00;amp[66]=00;amp[67]=00;amp[68]=00;amp[69]=00;

  amp[70]=70;amp[71]=00;amp[72]=00;amp[73]=00;amp[74]=00;
  amp[75]=00;amp[76]=00;amp[77]=00;amp[78]=00;amp[79]=00;
  amp[80]=80;amp[81]=00;amp[82]=00;amp[83]=00;amp[84]=00;
  amp[85]=00;amp[86]=00;amp[87]=00;amp[88]=00;amp[89]=00;
  amp[90]=90;
}
```

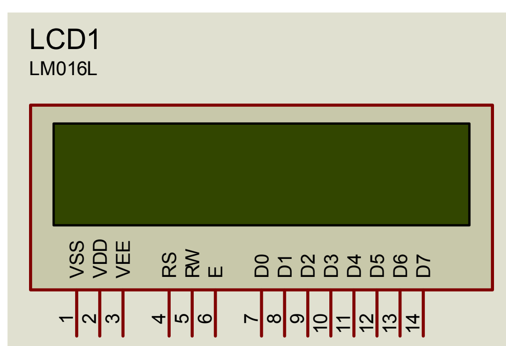
Waveform Generator Source Code (3/3)

```
void main()  //****************************************************
{
  enable_interrupts(int_rb);       // Port B interrupt for buttons
  enable_interrupts(global);
  ext_int_edge(H_TO_L);
  port_b_pullups(1);
  set_tris_D(0);

  // Calculate waveform values ************************************

  step=0;
  for(n=0;n<91;n++)
  {
    if(!input(PIN_B0)) amp[n] = 100;     // Square wave offset
    if(!input(PIN_B1))                   // Calculate sine values
    { sinangle = sin(step*stepangle);
      amp[n] = floor(sinangle*100);
      step = step+1;
    }
    if(!input(PIN_B2)) amp[n] = n;       // Triangular wave
    if(!input(PIN_B3)) setwave();        // Arbitrary wave
  }

  // Output waveform vales ****************************************

  while(1)
  { for(n=0;n<91;n++) {output_D(100+amp[n]); delay_us(time);}
    for(n=89;n>0;n--) {output_D(100+amp[n]); delay_us(time);}
    for(n=0;n<91;n++) {output_D(100-amp[n]); delay_us(time);}
    for(n=89;n>0;n--) {output_D(100-amp[n]); delay_us(time);}
  }
}
```


Class Assignment

- Design a system to read temperature data from the sensor LM35, and show the result on LCD





Serial communication

Serial transmission of binary data consists of sending the bits of a word one by one in a consecutive form and using the same pins.

Parallel vs. Serial IO

Parallel IO Pros/Cons

Pros: Speed, can increase bandwidth by either making data channel wider or increasing clock frequency

Cons: Expensive (wires cost money!). Short distance only – long parallel wire causes crosstalk, data corruption.

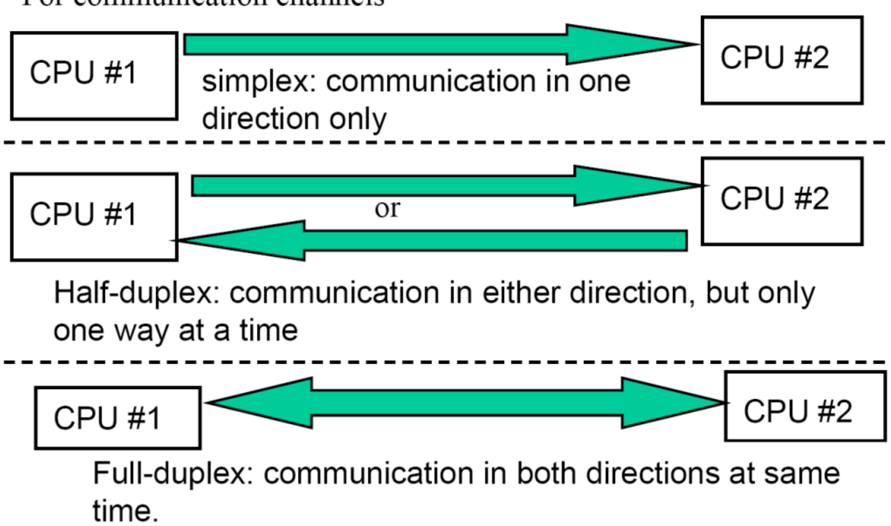
Serial IO Pros/Cons

Pros: Cheap, very few wires needed. Good for long distance interconnect.

Cons: Speed; the fastest serial link will typically have lower bandwidth than the fastest parallel link. However, for long distances (meters), new fast serial IO standards (USB2, Firewire) have replaced older parallel IO standards.

simplex vs half-duplex vs full-duplex

For communication channels



Serial Communication

1.PIC16 USART Serial Link

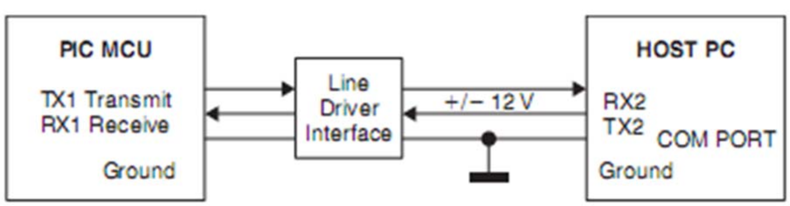
2.PIC16 SPI Serial Bus

3.I2C Serial Bus

USART

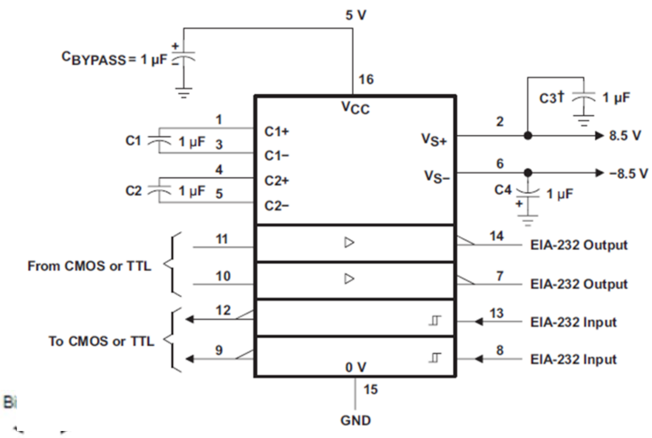
- The universal synchronous/asynchronous receive transmit (USART) device is typically used in asynchronous mode to implement off-board, one-to-one connections.
- The term asynchronous means no separate clock signal is needed to time the data reception, so only a data send, data receive, and ground wires are needed.
- It is quick and simple to implement if a limited data bandwidth is acceptable.

USART Operation



USART

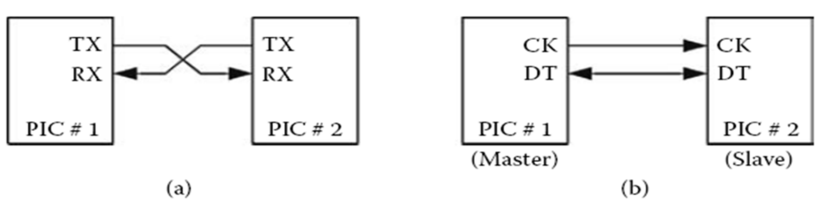
- RS232 driver
  - MAX232



USART RS232 Signal

USART with PIC microcontroller

- The PIC has a dedicated hardware RS232 port, but CCS C allows any pin to be set up as an RS232 port, providing functions to generate the signals in software.



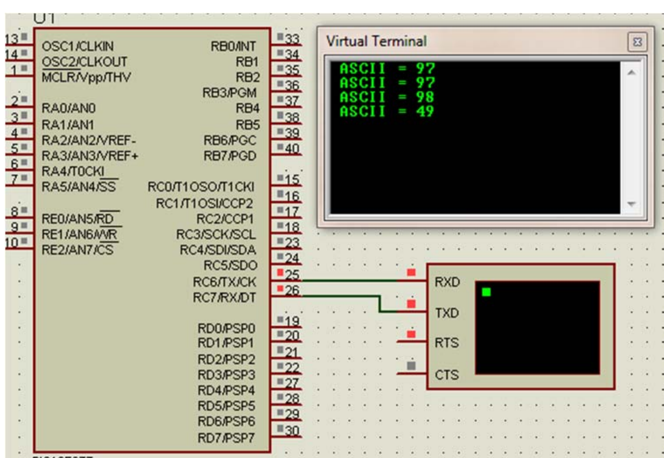
(a) Connection between two PICs in asynchronous mode

(b) Synchronous mode connection. CK is the clock pin, output in the master device and input in the slave device

CCS RS232 Serial Port Functions

Title	Description	Example
RS232 SET BAUD RATE	Set hardware RS232 port baud rate	setup_uart(19200);
RS232 SEND BYTE	Write a character to the default port	putc(65)
RS232 SEND SELECTED	Write a character to selected port	s=fputc("A",01);
RS232 PRINT SERIAL	Write a mixed message	printf("Answer:%4.3d",n);
RS232 PRINT SELECTED	Write string to selected serial port	fprintf(01,"Message");
RS232 PRINT STRING	Print a string and write it to array	sprintf(astr,"Ans=%d",n);
RS232 RECEIVE BYTE	Read a character to an integer	n=getc();
RS232 RECEIVE STRING	Read an input string to character array	gets(spoint);
RS232 RECEIVE SELECTED	Read an input string to character array	astring=fgets(spoint,01);
RS232 CHECK SERIAL	Check for serial input activity	s=kbhit();
RS232 PRINT ERROR	Write programmed error message	assert(a<3);

RS232 Periperal Simulation



The program

```
// Serial I/O using hardware RS232 port

#include "16F877A.h"
#use delay(clock=8000000)      // Delay functioin needed for RS232
#use rs232(UART1)              // Select hardware UART

void main() {
    int incode;
    setup_uart(9600);          // Set baud rate

    while(1)
    {   incode = getc();       // Read character from UART
        printf(" ASCII = %d ",incode);   // Display it on
        putc(13);              // New line on display
    }
}
```


Outline

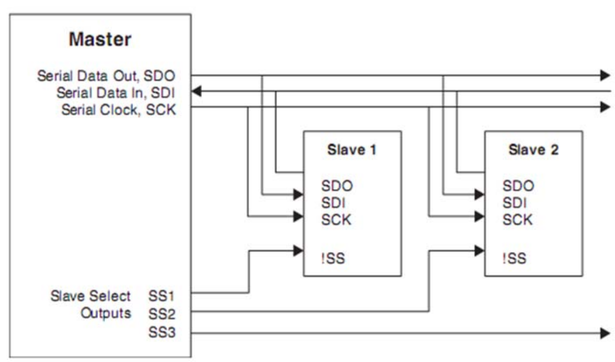
1. PIC16 UART Serial Link
2. PIC16 SPI Serial Bus
3. I2C Serial Bus

SPI Bus

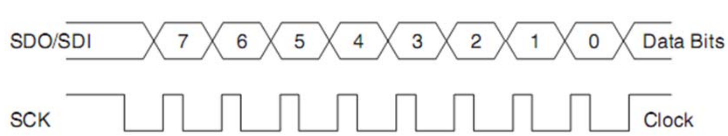
- The serial peripheral interface (SPI) bus provides high-speed synchronous data exchange over relatively short distances (typically within a set of connected boards), using a master/slave system with hardware slave selection
- One processor must act as a master, generating the clock. Others act as slaves, using the master clock for timing the data send and receive.
- The slaves can be other microcontrollers or peripherals with an SPI interface.
- The SPI signals are:
  - Serial Clock (SCK)
  - Serial Data In (SDI)
  - Serial Data Out (SDO)
  - Slave Select (!SS)

SPI Bus

SPI Connection



SPI Signals



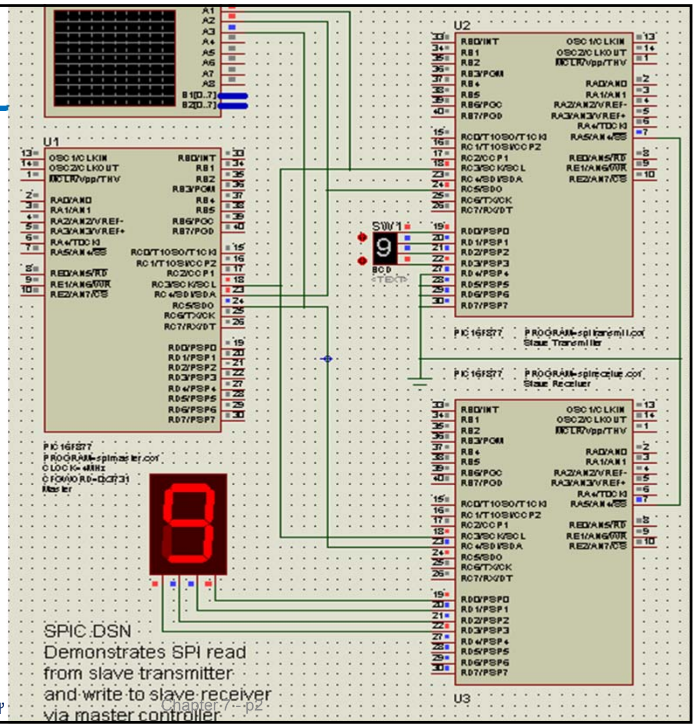
SPI Operation

- To transfer data, the master selects a slave device to talk to, by taking its SS line low.
- Eight data bits are then clocked in or out of the slave SPI shift register to or from the master. No start and stop bits are necessary, and it is much faster than RS232.
- The clock signal runs at the same speed as the master instruction clock, that is, 5MHz when the chip is running at the maximum 20MHz (16 series MCUs).

SPI Driver Functions

Operation	Description	Example
SPI SETUP	Initializes SPI serial port	setup_spi(spi_master);
SPI READ	Receives data byte from SPI port	inbyte=spi_read();
SPI WRITE	Sends data byte via SPI port	spi_write(outbyte);
SPI TRANSFER	Sends and receives via SPI	inbyte=spi_xfer(outbyte);
SPI RECEIVED	Checks if SPI data received	done=spi_data_is_in();

- SPI Test System Schematic



Master and slaves source code

```
// Master program U1

#include "16F877A.h"
void main()
{
    int number;
    setup_spi(spi_master); // SPI master mode

    while(1)
    {   number = spi_read(); // Read SPI input BCD
        spi_write(number);   // Resend BCD code to slave
    }
}
```

```
// Transmitter program U2

#include "16F877A.h"
void main()
{
    int sendnum;
    setup_spi(spi_slave);    // SPI slave mode
while(1)
    {   sendnum = input_D(); // Get BCD input
        spi_write(sendnum);  // Send BCD code to master
    }
}
```

```
// Receiver program U3

#include "16F877A.h"
void main()
{
    int recnum;
    setup_spi(spi_slave);     // SPI slave mode

    while(1)
    {   recnum = spi_read();  // Read BCD code at SPI port
        output_D(recnum);     // Display it
    }
}
```

Outline

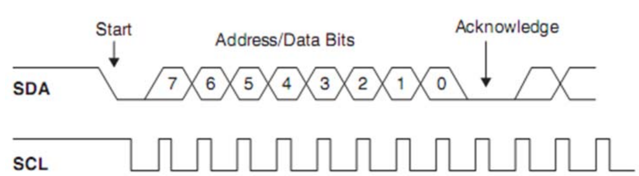
1. PIC16 UART Serial Link
2. PIC16 SPI Serial Bus
3. I2C Serial Bus

I^2C Bus

- The interintegrated circuit (I^2C) bus is designed for short-range communication between chips in the same system using a software addressing system.
- It requires only two signal wires and operates like a simplified local area network.

I2C connection

I^2C signals

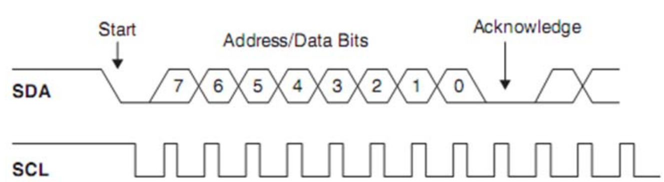


I^2C Bus

- The I^2C slave chips are attached to a two-wire bus, which is pulled up to logic 1 when idle. Passive slave devices have their register or location addresses determined by a combination of external input address code pins and fixed internal decoding.
- As for SPI, the clock is derived from the instruction clock, up to 5MHz at the maximum clock rate of 20MHz.

I^2C Bus

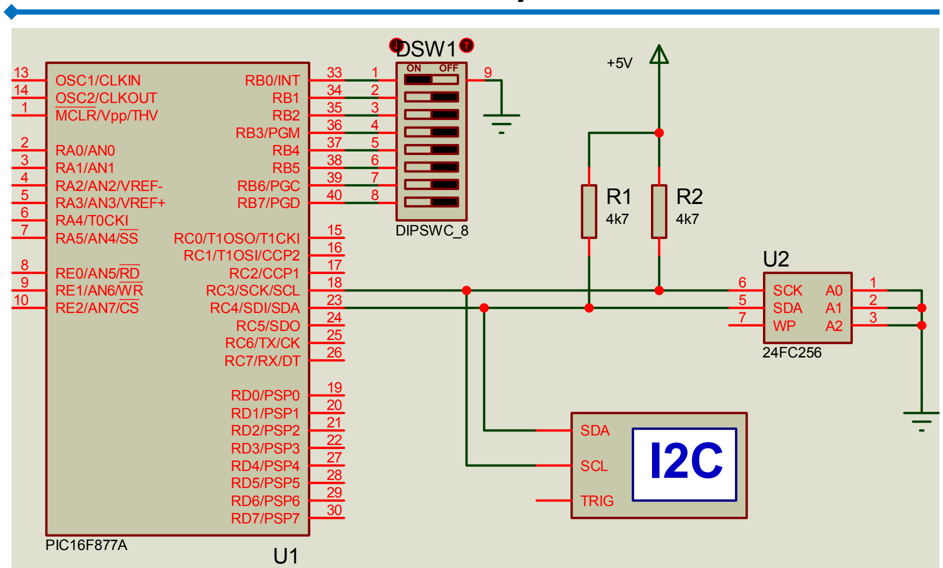
- To send a data byte, the master first sends a control code to set up the transfer, then the 8-bit or 10-bit address code, and finally the data. Each byte has a start and acknowledge bit, and each byte must be acknowledged before the next is sent, to improve reliability.
- The sequence to read a single byte requires a total of 5 bytes to complete the process, 3 to set the address, and 2 to return the data



I²C Functions

Operation	Description	Example
I2C WRITE	Send a single byte	i2c_write(outbyte);
I2C READ	Read a received byte	inbyte=i2c_read();
I2C STOP	Issue a stop command in master mode	i2c_stop();
I2C POLL	Check to see if byte received	sbit=i2c_poll();

I²C Test System



I^{2}C Test System

```
// Serial I/O using I2C synchronous link

#include "16F877A.h"
#use delay(clock=4000000)
#use i2c(MASTER,SCL=PIN_C3,SDA=PIN_C4)

void main()
{
    int sendbyte, lowadd;

    lowadd=0;
    port_b_pullups(1);
    sendbyte=(input_B());

    while(1)
    {
        i2c_start();                // start write cycle
        i2c_write(0xA0);            // send control byte
        i2c_write(0x00);            // send high address
        i2c_write(lowadd);          // send low address
        i2c_write(sendbyte);        // send data
        i2c_stop();

        delay_ms(5);                // wait for write
        lowadd++;                   // inc address
    }
}
```