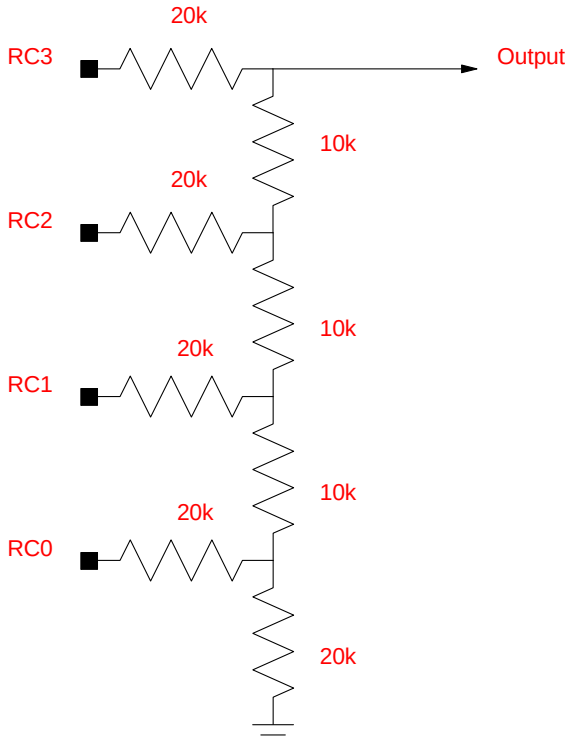
Analog Outputs (Digital to Analog Conversion)

R-2R Ladder

To output an analog signal, an R-2R ladder is used. For example, the circuit below converts a 4-bit binary number to analog signal whose voltage is

$$V_o = \tfrac{1}{2}(RC3) + \tfrac{1}{4}(RC2) + \tfrac{1}{8}(RC1) + \tfrac{1}{16}(RC0)$$

(hint: use superposition and Thevenin equivalents to verify this).



If RC3:RC2:RC1:RC0 represents a binary number from 0..15 and the PIC outputs 0V/5V, the output voltage is

$$V_o = \left(\frac{\text{binary data}}{16}\right) \cdot 5V$$

In general, for an R-2R ladder, the output voltage for an N-state R-2R ladder with 0V/5V inputs is

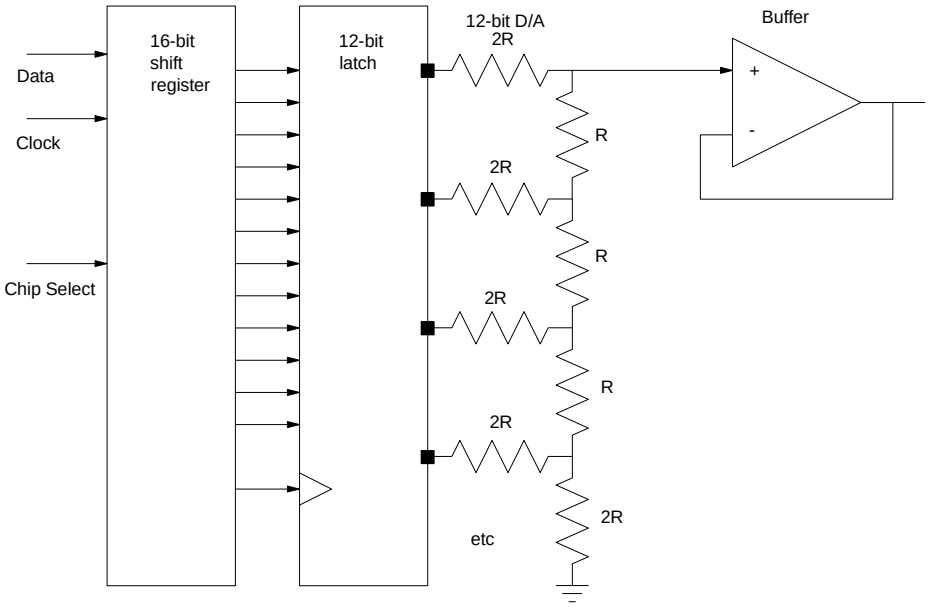
$$.V_o = \left(\frac{\text{binary data}}{2^N}\right) \cdot 5V$$

R-2R Ladder with Buffers:

You can buy R-2R ladders on a chip - termed D/A converters. (Digikey sells 8,066 different D/A converters as of this writing). These chips offer

- Different numbers of bits, from 1 to 26
- Different ways to write the N inputs, ranging from serial to parallel communications, and
- Various buffers at the output.

A typical D/A is the TLV5618, which is provided in your lab kit. This chip has essentially the following schematic:



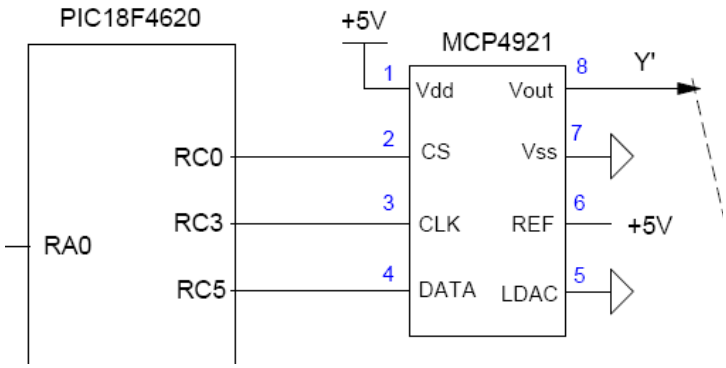
A shift register allows the PIC to send 12-bits of data to this chip using only three wires. The data is clocked in while Chip Select is low. Once Chip Select is brought high, the data is passed to a latch.

The latch sends the data to the R-2R ladder, which converts the data to 0..5V.

An output buffer helps to prevent the external circuit from loading the output of the R-2R ladder.

The MCP4921 in your lab kit is a 12-bit D/A. It's sent out as a 16-bit message with the data parsed as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	12-bit data (0..4095)											



A subroutine to drive this chip follows:

```
void D2A(unsigned int X)
{
   unsigned char i;

   TRISC0 = 0;
   TRISC3 = 0;
   TRISC5 = 0;

// Add 0011 to the first four bits to set up the D/A

   X = X & 0x0FFF;
   X = X + 0x3000;

   RC0 = 1;
   RC3 = 1;

// CS goes low to select the D/A chip
   RC0 = 0;

// Send out 16 bits of data
   for (i=0; i<16; i++) {
      if (X & 0x8000) RC5 = 1; else RC5 = 0;
      RC3 = 0;
      X = X << 1;
      RC3 = 1;
      }

// CS goes high to terminate the communicaitons
   RC0 = 1;
   }
```

Analog Inputs (Analog to Digital Conversion)

Introduction:

Contrary to popular belief, not all the world is binary. Sometimes you'll want to read an analog signal, such as

- The angle of a joystick for a computer game,
- The resistance of a thermister to measure temperature (a thermister is a resistor which changes with temperature)
- The voltage from a photovoltaic cell to measure light intensity, etc.

In the world of a computer, all numbers are binary. If a computer is to monitor the status of an analog signal, some device is required to convert this signal to a digital one. An analog-to-digital converter (A-to-D to A/D) is such a device.

Definitions:

A/D ..... Analog to Digital Converter. Interface from the real world to the computer

D/A ..... Digital to Analog Converter. Interface from the computer to the real world.

N-bit A/D ..... The result of the A/D conversion will be an N-bit unsigned number.

Quantization Level .. a) The number of possible results from an A/D conversion. (A 10-bit A/D has 210 or 1024 Quantization levels)
b) The size of one count. If 0V = 0 and 5V = 1023, then one count = 5V/1024 = 4.88mV

Qualtization Noise .. Since you need to round to the nearest integer, the analog signal will be distorted slightly. This distortion is termed Quantization Noise.

Aliasing ..... Sampling is not a linear process. As a result, it produces harmonics of the signal being sampled centered at harmonics at the sampling rate. This phenominum is termed aliasing.

Sampling Rate ..... The frequency at which the analog signal is sampled.

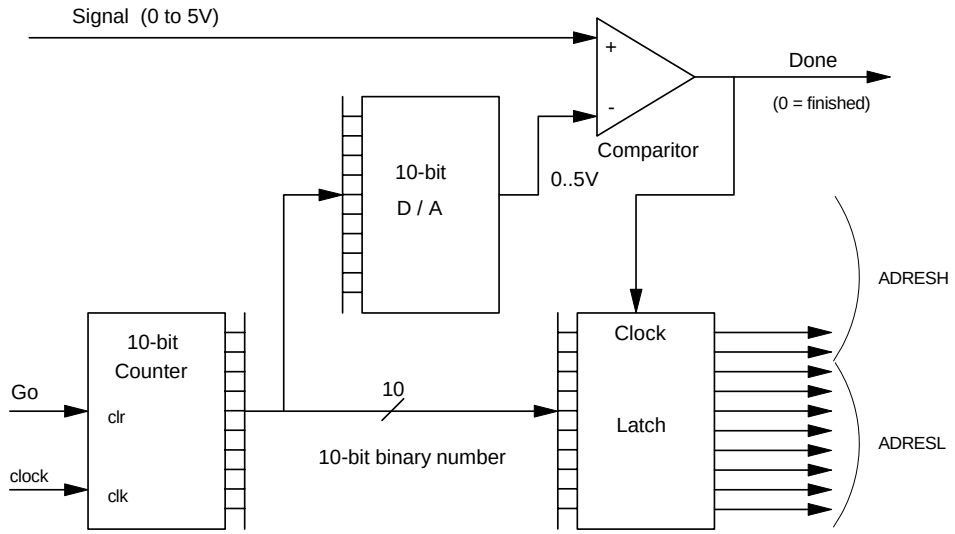
Sampling Period .... The time between samples.
note: Constant sampling rates are almost always used since this greatly simplify the analysis of digital systems.

Voltage Measurement:

Goal: Measure a voltage between 0V and +5V

Approach: A/D converter. A 10-bit A/D (line on the PIC16F876) converts as follows:

- 1) Clear the counter to start the conversion.
- 2) Wait for the Done flag to be set.
- 3) When set, you're ready to read the A/D data. (about 19.7us later)



Note: With this approach, the reading is linear with voltage:

$$Reading = \left(\frac{Voltage}{5}\right) \cdot 2^n$$

For a 10-bit A/D

$$Reading = \left(\frac{V}{409.6}\right)$$

The sampling rate is also much higher:

$$F_{sample} = \left(\frac{1}{19.7us}\right) = 50.76kHz \quad .$$

A/D Conversion on the PIC18F4626:

PORTA is can be used for analog or digital inputs. If you want to use PORTA, ADCON0, ADCON1, and TRISA need to be set up to tell the PIC chip how to use PORTA. The pins and bit assignments for an analog input follow:

Address	Register Name	Bit							
		7	6	5	4	3	2	1	0
0xFC0	ADCON2	ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
0xFC1	ADCON1	—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
0xFC2	ADCON0	—	—	CHS3	CHS2	CHS1	CHS0	GODONE	ADON
0xFC3	ADRES	16 bit register (0..65535)							

ADCON0:

- CHS: Channel to convert: You must wait 14us if you change channels
  - 0000 = Channel 0 (RA0/AN0)
  - 0001 = Channel 1 (RA1/AN1)
  - 0010 = Channel 2 (RA2/AN2)
  - 0011 = Channel 3 (RA3/AN3)
  - 0100 = Channel 4 (RA5/AN4)
  - 0101 = Channel 5 (RE0/AN5)
  - 0110 = Channel 6 (RE1/AN6)
  - 0111 = Channel 7 (RE2/AN7)
  - 1000 = Channel 8 (RB2/AN8)
  - 1001 = Channel 9 (RB3/AN9)
  - 1010 = Channel 10 (RB1/AN10)
  - 1011 = Channel 11 (RB4/AN11)
  - 1100 = Channel 12 (RB0/AN12)

- ADON: 1 = turn on the A/D (and draw an additional 180uA)

- GODONE: Start the A/D conversion. Conversion is complete when bit GODONE = 0 (about 34us later)

ADCON1

bit 5 VCFG1: Voltage Reference Configuration bit (VREF- source)

- 1 = VREF- (AN2)
- 0 = VSS

bit 4 VCFG0: Voltage Reference Configuration bit (VREF+ source)

- 1 = VREF+ (AN3)
- 0 = VDD

PCFG3:PCFG0 determine whether certain pins are analog inputs (A) or binary I/O (D)

PCFG3: PCFG0	RB0 AN12	RB4 AN11	RB1 AN10	RB3 AN9	RB2 AN8	RE2 AN7	RE1 AN6	RE0 AN5	RA5 AN4	RA3 AN3	RA2 AN2	RA1 AN1	RA0 AN0
0000	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

ADCON2

ADFM: A/D Result Format Select bit

- 1 = Right justified
- 0 = Left justified

Result from an A/D Conversion 10-bit Result = abcdefghij		
ADFM	ADRESH	ADRESL
0	abcd efgh	ij00 0000
1	0000 00ab	cdef ghij

bit 6 Unimplemented: Read as ‘0’

bit 5-3 ACQT2:ACQT0: A/D Acquisition Time Select bits

- 110: Automatically restart the A/D conversion every 16th clock (3.2us)
- 000: Manual operation of the A/D (user must set GODONE to start conversions)

bit 2-0 ADCS2:ADCS0: A/D Conversion Clock Select bits

- 101 = FOSC/16 (use with a 20MHz crystal)

Example: Set up the A/D so that

- PORTA/E are analog inputs, PORTB/C/D are binary
- The conversion will be right justified (ADFM = 1)
- A 20MHz crystal is used. (ADCS = 10: F_{OSC} / 32)

Solution:

	7	6	5	4	3	2	1	0
ADCON2	ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
	1	0	0	0	0	1	0	1

	7	6	5	4	3	2	1	0
ADCON1	—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
	0	0	0	0	0	1	1	1

	7	6	5	4	3	2	1	0
ADCON0	—	—	CHS3	CHS2	CHS1	CHS0	GODONE	ADON
	0	0	0	0	0	0	0	1

```
void A2D_Init(void)
{
    TRISA  = 0xFF;
    TRISE  = 0x0F;
    ADCON2 = 0x85;
    ADCON1 = 0x07;
    ADCON0 = 0x01;
    }
```

Read RA0 and return the A/D result as an integer:

- Set CHS = 0000 to select RA0
- Set GODONE
- Wait for Go/Done to go low

Solution #1: Always read channel 0:

```
unsigned int A2D_Read(void)
{
   unsigned int result;
   ADCON0 = 0x01;                    // Select channel 0, turn on, CLK/32
   GODONE = 1;                       // Start conversions
   while(GODONE);                    // wait until done (approx 8us)
   return(ADRES);                    // and return the result
}
```

Solution #2: Allow you to read channels 0..7:

```
unsigned int A2D_Read(unsigned char c)
{
   unsigned int result;
   unsigned char i;
   c = c & 0x0F;
   ADCON0 = (c << 2) + 0x01;              // set Channel Select
   for (i=0; i<3; i++);                   // wait 2.4us (approx)
   GODONE = 1;                            // start the A/D conversion
   while(GODONE);                         // wait until complete (approx 8us)
   return(ADRES);
}
```