Brock University  Physics Department

St. Catharines, Ontario, Canada L2S 3A1

Phys 3P92: Experimental Physics II (Electronics)

E. Sternin

Contents

Experiment 1

Op-amps and basics of signal conditioning

The operational amplifier (op-amp) is the most versatile piece of analog hardware yet developed. Originally named for its ability perform mathematical operations on analog voltages, the op-amp has become an essential building block of much of modern electronics. In this experiment, we will analyze the input-output characteristics of an op-amp as well become acquainted with some of the basic circuits in which it is used.

1.1 Introduction

Ideal op-amps

An op-amp is a differential amplifier with an inverting V_- input and non-inverting V_+ input. The output voltage V_{out} is given by the difference of these two input voltages times the open loop gain A_v:

$$V_{\text{out}} = A_v \times V_{\text{in}} = A_v \times (V_+ - V_-) \tag{1.1}$$

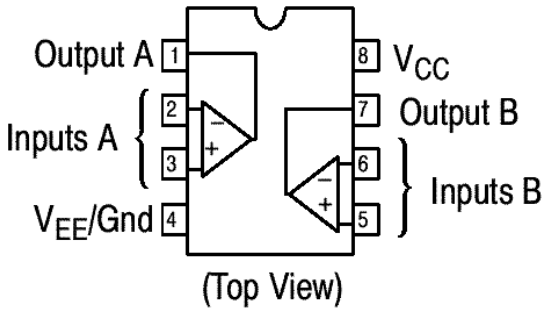
A standard way to derive approximate theoretical equations for the circuits involving op-amps is to assume that the op-amp is an *ideal* device having the following electrical characteristics:

1. the inputs draw no current, hence $i_+ = i_- = 0$ and the input impedance $Z_+ = Z_- = \infty$;
2. the output can supply an infinite amount of current, hence $Z_o = 0$;
3. the open loop gain, or voltage amplification $A_v = \infty$.
4. The op-amp adjusts the output voltage so that $V_- = V_+$. This follows from Equation 1.1 since V_{out} cannot exceed the finite power supply voltage. This equivalence is used to determine the gain equation for an (ideal) op-amp circuit.

The LM358 op-amp

The LM358 consists of a pair of general purpose operational amplifiers capable of amplifying signals ranging from 0 Hz (DC) to 1 MHz. The chip can operate using a dual power supply of up to ± 15 V

down to a single 3 V battery. It can be used in mixed analog/digital circuits that typically operate from a single 3-5V power supply.



? From the LM358 data sheet, determine the values of A_v, the input bias current and output source current. Does the LM358 approximate the characteristics of an ideal op-amp? Explain.

? The slew rate dV_{out}/dt defines the maximum rate of change in V_{out}. What is the LM358 slew rate? Does the frequency response of the amplifier depend on the amplitude of the signal?

! **Care should be taken to ensure that all integrated circuits (IC's) are powered with both $\mathbf{V_{CC}}$ (+) and $\mathbf{V_{EE}}$/Gnd (-) whenever an input signal is supplied! Failure to do this will destroy IC's.**

1.2 Open-loop operation

Since A_v of the op-amp is very large, a tiny voltage difference between the inputs causes the output to swing between the positive $V_{\text{out}}^{(\max)}$ and negative $V_{\text{out}}^{(\min)}$ power supply limits, or saturate. This effect can be used to implement a *voltage comparator* or *level detector*.

To implement a comparator, one input is set to a reference voltage. The output changes state as the voltage at the other input swings above and below the reference voltage.

Due to input signal noise and non-ideal op-amp operation, voltage differences between V_+ and V_- that are consistently less than a few millivolts will cause the op-amp output to oscillate or otherwise behave in an erratic fashion.

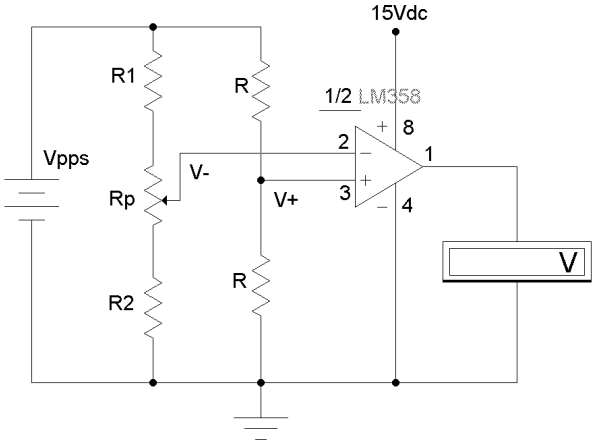


Figure 1.1: LM358 open loop analysis

Null voltage measurement

(!) Assemble the circuit shown in Figure 1.1. Start by making the power connections to the op-amp, then connect the HP benchtop programmable power supply (PPS) to the workstation and set it to the 6 V DC range. Turn on the output and adjust the output voltage to around 5 V.

? Why might you not want to use the already available workstation 5 V power supply?

(!) Complete the connections to the op-amp V_+ and V_- terminals. R_p is a 1 KΩ variable resistor, (a **p**otentiometer). A small screw allows for the centre tap connection to be set anywhere between the two resistor ends. The resistors labelled R can be of similar value, in the 10-100 KΩ range.

(!) Measure the resulting V_+ voltage, then determine values for R_1 and R_2 that will allow the V_- voltage to be adjusted above and below V_+.

[?] Are R_1 and R_2 necessary? What is their function in this circuit? How did you determine their values? Explain how the op-amp input bias current determines the practical range of values for the various resistors used in this circuit.

(!) Connect and scope to monitor the op-amp output and adjust V- until a transition from one output voltage limit to the other occurs. Measure and record the positive and the negative voltage limits of the op-amp output.

Note: type `GDS-1102A` in a terminal window to export the scope screen to the computer monitor and optionally save a screenshot of the scope screen for inclusion in your lab report.

(!) In the same circuit, connect the benchtop digital multimeter (DMM) to the V- op-amp input. Adjust the potentiometer carefully to where the op-amp output just begins to decrease from its positive limit (as observed on the scope), where it is as close to zero output as you can set it, and where it is not quite at the negative limit. Record these three V_- values. Repeat these observations several times.

(!) Without changing any settings, use the DMM to measure V_+.

[?] Estimate the *open loop gain*, A_v, of the op-amp and its input offset voltage from the above measurements. Compare with the nominal value that you obtained from the LM358 data sheet.

[?] Estimate the op-amp *slew rate* and compare it with the stated nominal value.

[?] Connect both inputs to the op-amp to ground and measure the *output offset voltage*. What should this value be for an ideal op-amp?

1.3 Closed-loop operation

Application of feedback from V_{out} to V_- causes the op-amp to conform to Rule 4 mentioned in the introduction. This arrangement, shown in Figure 1.2, is known as a *voltage follower* or *unity-gain amplifier*.

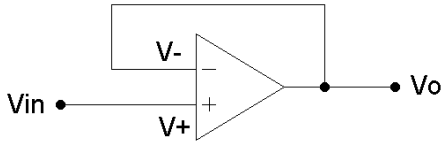


Figure 1.2: A voltage follower

[?] How might this op-amp arrangement be useful? What is being amplified? Derive the gain equation.

An analog memory cell

It is sometimes necessary to temporarily store a voltage. This is required when converting a voltage to a digital value, or to implement an analog signal delay. Figure 1.3 shows the schematic of a typical *track-and-hold* circuit. When the switch is closed, V_{out} tracks V_{in}. With the switch open, the capacitor is effectively isolated from V_{in} and V_{out} reflects the voltage stored in the capacitor.

? Which op-amp characteristics are desirable in this type of circuit? What are the benefits/limitations imposed upon the circuit by the resistor and capacitor?

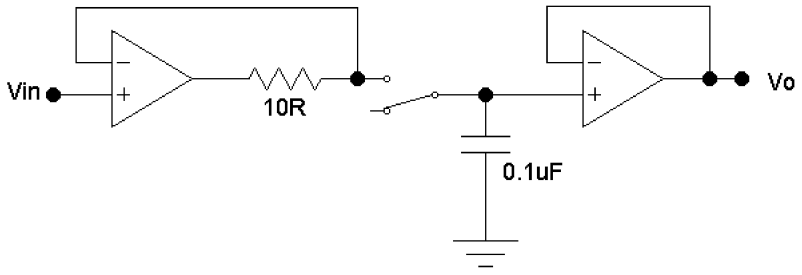


Figure 1.3: Sample and hold circuit

! Sketch the circuit of Figure 1.3 in your lab book, clearly labelling all the connections to the LM358 dual op-amp chip.

You can use a jumper in place of the switch if one is not installed on the breadboard.

? Set V_{in} to a 1 Hz sine wave. Describe the output as the switch is opened and closed.

? In track mode, with the switch closed, how are V_{in} and V_{out} related? As you increase the V_{in} frequency, what do you observe?

? With the circuit in hold mode and the switch open, describe V_{out}. Does V_{out} change in time? If so, determine the discharge rate of the capacitor. How long before V_{out} drops by 1%?

1.4 Analog computation

The op-amp was originally designed to perform mathematical operations from addition to multiplication, exponentiation and the solution of differential equations. The electrical behaviour of resistors, capacitors and diodes are used to this end. While not as precise as digital devices, analog computers are very fast and simple to implement and do not require data conversion to and from the digital domain.

Figure 1.4 shows a two op-amp circuit that can be used to evaluate the equation

$$Y = mX + b = m \times (X + b/m). \tag{1.2}$$

The first adds an offset b/m to V_{in}. The second op-amp sets the gain, or slope m.

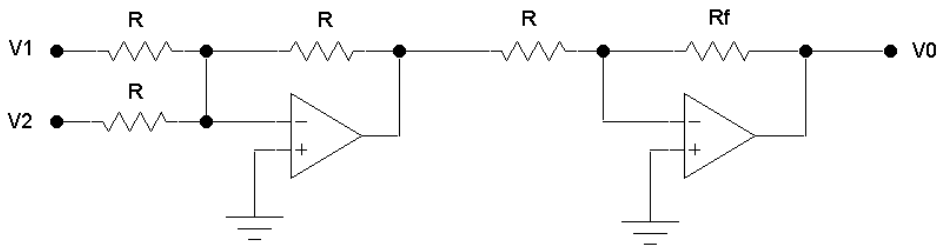


Figure 1.4: Two op-amp solution of $Y = mX + b$

? Derive the transfer function for the two op-amp circuit of Figure 1.4.

The above is not the only way to implement our equation $Y = mX + b$ using op-amps. It may not seem readily apparent, but the circuit of Figure 1.5 also evaluates $Y = mX + b$, this time using a single op-amp. Due to the feedback path, the op-amp adjusts the output V_{out} so that $V_- = V_+$. Because of the very large input impedance of the op-amp, no appreciable current flows into the op-amp inputs. Thus the presence of an op-amp is not affecting the currents flowing through the resistors, and we can draw the electrically equivalent circuit as two separate voltage dividers as shown on the right-hand side of Figure 1.5.

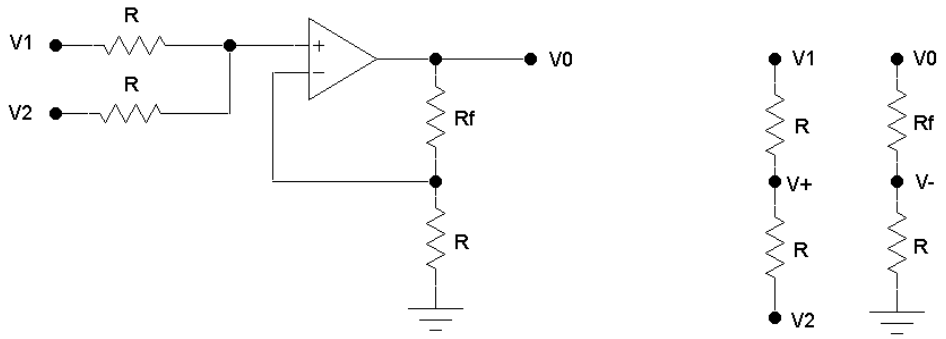


Figure 1.5: Single op-amp solution of $Y = mX + b$

? Show that the equation below is valid and that it does represent the equation $Y = mX + b$:

$$V_{\text{out}} = \frac{R_f + R}{2R} \times (V_1 + V_2) \tag{1.3}$$

A practical example

It is often useful to convert a transducer output voltage to a voltage range that quantitatively represents the actual quantity that the sensor measures.

Suppose that you wish to build an analog thermometer calibrated to display temperature on a voltmeter in units of 100mV/°C so that 0°C displays 0V, 10°C displays 1V, and so on.

Suppose that the temperature sensor used is an LM61 temperature-to-voltage converter. The output voltage of this device corresponds to 600 mV at 0°C and varies linearly at a rate of 10 mV/°C.

(!) Determine for the circuit of Figure 1.4 the transfer function parameters required to properly display the LM61 output as temperature on the voltmeter.

(!) Build the circuit. Set $V_{CC} = +15$ V and $V_{EE} = -15$ V. Select $R \approx 10\text{k}\Omega$.

(!) Simulate the LM61 output voltage with the PPS. Vary the PPS voltage and verify that the circuit behaves as expected.

(!) Now replace the PPS with the LM61. The LM61 is a three-pin device. With the flat face toward you, connect the left pin to +5 V and the right pin to 0 V. The centre pin is the output voltage, as described.

Connect the LM61 to the op-amp circuit and verify that the circuit converts the LM61 output voltage as predicted.

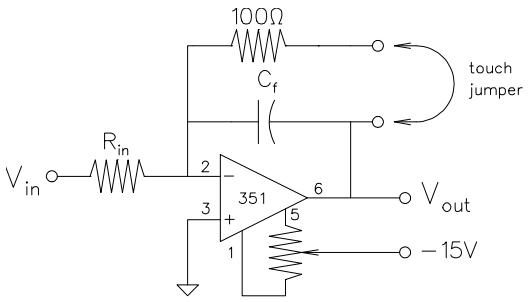
[?] What output voltage do you expect at room temperature? When you hold the LM61 with your fingers, does the output voltage increase?

Experiment 2

Advanced op-amp designs

2.1 Op-amp integrator

The purpose of this section is to wire up and analyze an analog integrator, using a carefully balanced op-amp and a low-leakage quality capacitor. We will observe the circuit response to both dc input signals and to the ac waveforms generated by the FG.



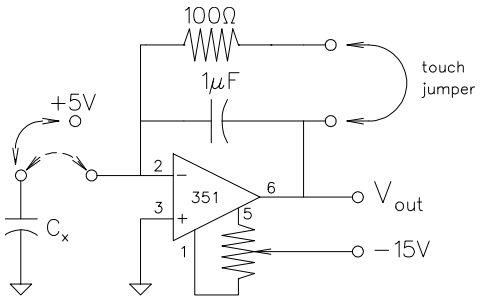
Using a capacitor as the feedback element in the inverting amplifier circuit, wire up the op-amp integrator.
Use a 1 μF low–leakage capacitor (10% tolerance or better), $R_{\rm in} = 1$ MΩ, and set $V_{\rm in} = 100$ mV.

(!) Measure the times required for the output to change by 1V, 3V, 5V, and 8V. Begin the timing when the touch jumper is removed. Use the jumper to discharge the integrating capacitor, *i.e.* to restart the integrator. Repeat 1V measurement at least 3 times, estimate the precision of your measurements (standard deviation).

The above measurements require that the op-amp be well-balanced. To test, restart the integrator, and quickly remove $V_{\rm in}$ when $V_{\rm out} \simeq 1$V. Does $V_{\rm out}$ remain constant after that? If not, re-balance the op-amp.

(!) Connect the input to ground, reset the integrator, and observe $V_{\rm out}$ on the most sensitive DMM scale. Record your observations.

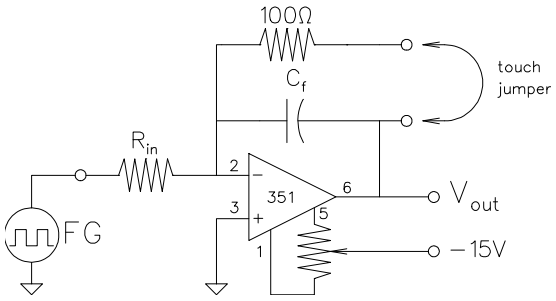
(!)



Modify the circuit as shown, turning it into a charge–to–voltage converter. The circuit will be used to measure the capacitance of another capacitor, C_x.
Discharge C_f ($1\,\mu$F), disconnect the touch jumper, then carefully move the input jumper from +5V to the negative input of the op-amp, and observe changes in $V_{\rm out}$. Repeat several times.

[?] Compare the measured value of the ratio C_f/C_x with that obtained by a direct reading of the capacitance meter.

(!)



Use FG to provide a square–wave input to the integrator. Use $R_{\rm in} = 1\,{\rm M}\Omega$ and set the frequency to about 1 kHz. Choose the C_f value appropriate for this frequency. Monitor both the input and the output with the scope. Make sure you adjust FG to have a zero DC offset. Alternatively, you may want to use a small capacitor ($\simeq 1\,\mu$F) in series with FG, to remove the dc component from the input.

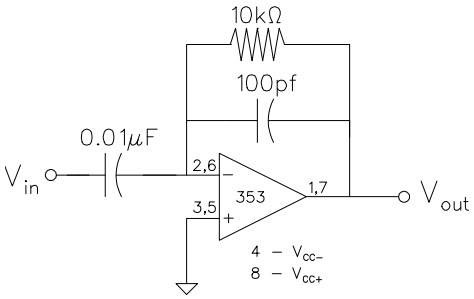
(!) Sketch and explain the observed waveforms.

2.2 Op-amp differentiator

By interchanging the resistor and capacitor of the op-amp integrator, we obtain an op-amp differentiator. We will analyze its response to various waveforms of the FG.

Do not remove the circuit of the previous section; you may want to re-use it in Section 2.3.

(!)



Wire up an op-amp differentiator as shown. In a dual-353 package you may choose either of the two op-amps (pins 2,3,1 or 6,5,7). The 100 pF capacitor is included to provide noise stability. For this circuit,

$$V_{\rm out} = -RC\frac{dV_{\rm in}}{dt}$$

Set the FG to 5V peak–to–peak 1 kHz triangular wave and connect it as $V_{\rm in}$.

(!) Sketch the input and output waveforms, including the proper scales. Make sure your scope is on a calibrated setting.

Calculate and record the slope of the input triangular wave. Also, record the amplitude of the square wave at the output.

? Calculate the expected theoretical value for the differentiator output and compare it to the experimental value.

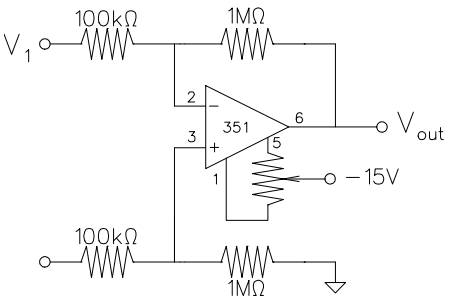
! Change the FG to square wave setting. Sketch the observed waveforms.

2.3 Difference amplifier

The purpose of this section is to introduce precision amplifiers and to learn to distinguish differential and common mode signals.

Ref: Simpson, Ch. 9–10, esp. Sec. 9.8.7, 10.4; Faissler, Ch. 31 (review); Malmstadt *et al.*, Ch. 8.1.

!



Wire up the difference amplifier as shown: Balance the op-amp by connecting both V_1 and V_2 to ground and adjusting the offset potentiometer until $V_{\text{out}} = 0$.
Leaving V_2 grounded, vary V_1 (several values between +1V and −1V) and measure V_{out}.

? Calculate the average gain of the amplifier. In this measurement, which components determine the gain of the amplifier? How does the measured value compare with the theoretical one?

! Connect V_2 to a constant +1V source and repeat the above two steps.

! Connect *both* V_1 and V_2 to the same variable voltage source; measure V_{out} for several values of $V_1 = V_2$ between +1V and −1V.

? Plot V_{out} *vs.* V_1 and determine the value of the **common mode gain** from the plot.

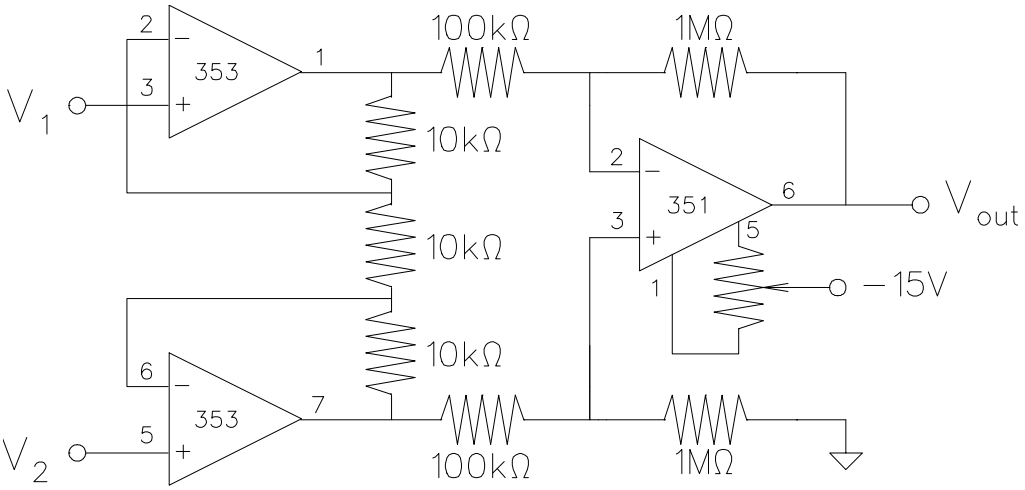
? Interpret your data in terms of the imbalance of the resistance ratios of the two pairs of resistors determining the gain, for the inverting and for the non-inverting input. Which pair has the higher gain and by how much? How could this common mode gain be reduced?

? Calculate the **common-mode rejection ratio** (CMRR) for your difference amplifier. Calculate the maximum common-mode signal the amplifier can accept if a 100 mV signal is to be amplified with an error of less than 0.1%.

2.4 Instrumentation amplifier

The purpose of this section is to combine the advantages of a difference input with the high input resistance of the voltage follower in a complete instrumentation amplifier.

(!) Wire up the instrumentation amplifier as shown (add the input voltage followers to the existing circuit).



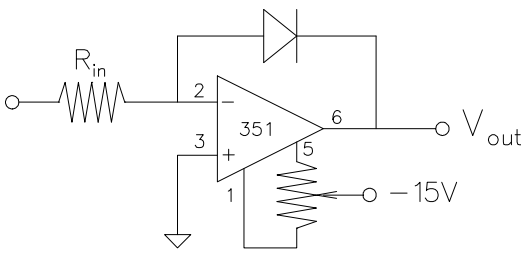
Check the offset of the instrumentation amplifier and adjust the difference amplifier offset potentiometer if needed. Measure V_{out} for various values of V_1 and V_2 so that you will be able to determine the difference gain and the common mode rejection ratio of the instrumentation amplifier. Be sure you have taken sufficient data to perform your calculations.

[?] Describe the reasoning you used in selecting the values for V_1 and V_2. From these data, determine the gain and the CMRR. Explain your interpretation of the data. Compare your results with the expected values.

2.5 Logarithmic amplifier

Using a non-linear feedback element with an op-amp (e.g. a pn–junction diode) produces startlingly different transfer functions. Logarithmic amplifiers serve as the basis for circuits such as analog multipliers studied in Section 2.6.

(!)



Carefully balance a 351 op-amp. Then wire the logarithmic amplifier (log amp), using a signal diode as the feedback element.

(!) Measure V_{out} as a function of V_{in} and R_{in}:

V_{in}	R_{in}	I_{in}	V_{out}
10.0mV	1MΩ		
10.0mV	100kΩ		
100.0mV	100kΩ		
1.0V	100kΩ		
10.0V	100kΩ		
10.0V	10kΩ		

? Plot $\log I_{\text{in}}$ *vs.* V_{out}.

For all but very small forward bias voltages, the current through a diode varies exponentially with the applied voltage:

$$I \simeq I_i e^{eV/\eta kT}$$

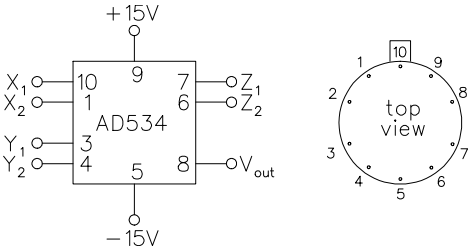
where η is an empirical parameter (~ 2 for Si, 1 for Ge diodes), and I_i is the intrinsic current at zero bias.

Apply circuit analysis (Simpson, Sec. 9.7) to your logarithmic amplifier and verify that the same relationship holds for the measured I_{in} and V_{out}.

Fit your data to the above equation and determine the parameters η and I_i for your diode. Can you tell if this is a Si or a Ge diode?

2.6 Analog multiplier

Combining log amps with adding amps allows one to build analog multipliers and other components of analog computers (for a review, see Faissler, Ch. 30). Here we examine the transfer functions of one such commercial device, AD534.



AD534 is internally trimmed and does not require external trimmer potentiometers. Its pinout is shown on the left.

! For multiplication, use the fixed +10V supply from the job board as the X_1 input and use several fixed voltages from the reference job board as the Y_1 input (+10V, −10V, −1V, +1V). Connect the Z_1 input to the output. Connect the X_2, Y_2, and Z_2 inputs to common. Test the multiplier in all four quadrants by applying voltages of both polarities in the range of ±10V. The multiplier transfer function should be $V_{\text{out}} = (V_x \times V_y)/10$. Include in your data set (X_1, Y_1) values of $(+10, 0)$, $(0, 0)$, and $(0, +10)$.

? Offsets modify the multiplier equation:

$$V_{\text{out}} = {V_{\text{out}}}^{(0)} + 0.1 \times \left[V_x - V_x^{(0)}\right] \times \left[V_y - V_y^{(0)}\right]$$

where $V_x^{(0)}$, $V_y^{(0)}$, and ${V_{\text{out}}}^{(0)}$ are the X, Y, and output offsets, respectively. Use your data to evaluate each of the offsets. Explain how magnitude of offset–induced errors changes with X and Y input levels.

(!) To obtain an output voltage proportional to the square of an input voltage, connect both X_1 and Y_1 inputs to the same voltage source and the X_2 and Y_2 inputs to common. The Z_1 input remains connected to the output. Test the circuit over a ±10V range of voltages and compare to the expected $V_{\text{out}} = 0.1 \times {V_{\text{in}}}^2$.

(!) The "squared voltage"output can be plotted against the input with the xy–mode of the oscilloscope. Substitute the output of the FG set in the sine wave mode as the source in the squaring circuit wired above. Connect the multiplier output to the vertical scope input and the FG output to the horizontal. Use a 10 Hz sine wave signal. Sketch the resulting display.

(!) Now use the dual-trace mode to observe the waveforms of the input and output signals. Sketch a representative display and indicate the position of OV for each waveform.

[?] Explain the relationship of the frequencies and the DC components of the input and output waveforms.

Optional: analog division

(!) To obtain division, connect the multiplier output to the Y_2 input. Now Z_1 is no longer connected to the output, and Z_2 is no longer grounded. In this configuration:

$$V_{\text{out}} = 10 \times \frac{V(Z_2) - V(Z_1)}{V(X_1) - V(X_2)} + V(Y_1)$$

Measure V_{out} for several values of $V(Z_2) - V(Z_1)$ and $V(X_1) - V(X_2)$. For simplicity, you may want to ground Z_1, X_2, and Y_1. Make sure you keep $V(X_1) - V(X_2)$ positive (see the spec sheets of AD534).

[?] The output limits of AD534 are ±11 V. Calculate and plot the minimum value for $V(X_1) - V(X_2)$ as a function of $V(Z_2) - V(Z_1)$ over the $V(Z)$ range of ±10 V.

Experiment 3

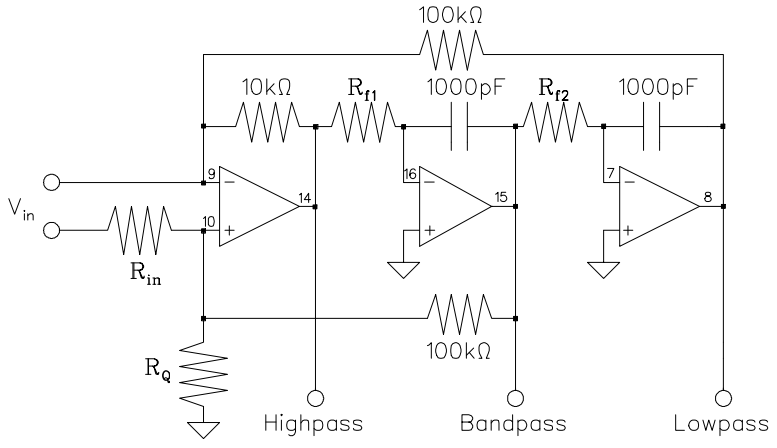
Active filters and tuned amplifiers

3.1 Active filter

Weak signals require special attention. The techniques of separating signal from noise vary depending on the nature of the signal and of noise. There are no general easy prescriptions.

When the frequencies of the signal and of the noise differ, one way to increase the signal–to–noise (S/N) ratio is to restrict the bandwidth of the amplifier in such a way that only the signal frequencies are transmitted. This principle is illustrated using an active filter device.

The AF100 universal active filter is a versatile active filter device. It has high–pass (HP), low–pass (LP), and band–pass (BP) outputs simultaneously available and an uncommitted summing amplifier for making notch filters. The centre frequency is tunable from 200 Hz to 10 kHz with two resistors. The quality factor (Q) is variable from 0.01 to 500 by changing two additonal resistors. The AF100 can be used in either an inverting or a non–inverting configuration.

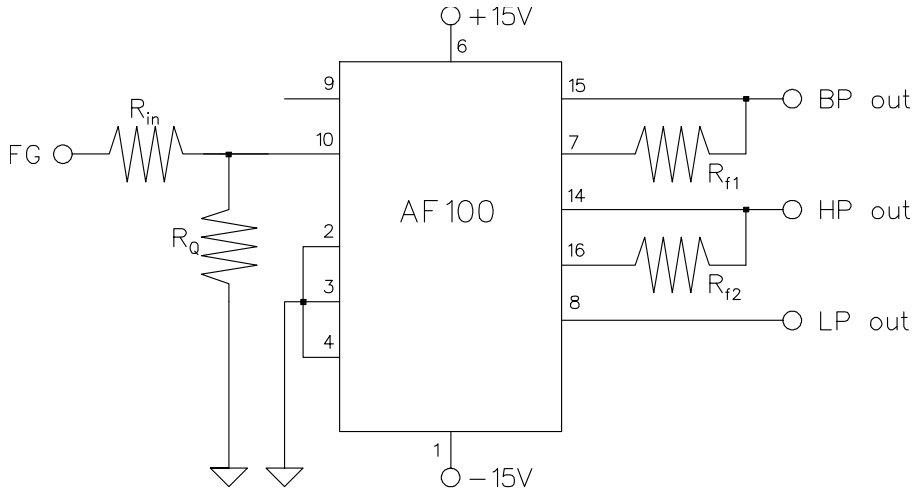


The pinout and the schematic diagram of an AF100 are as shown. $R_{\rm in}$, $R_{\rm Q}$, $R_{\rm f1}$, and $R_{\rm f2}$ must be supplied externally to AF100 (see below). All other components are internal.

The gain and Q value of the filter are determined by $R_{\rm in}$ and $R_{\rm Q}$. The centre frequency of the filter is determined by identical resistors, $R_{\rm f1}$ and $R_{\rm f2}$, according to

$$f_0 = \frac{50.33 \times 10^6}{R_{\rm f}}, \qquad \text{in Hz}$$

(!)



Wire up the non–inverting mode filter using external resistors of $R_{\rm f1} = R_{\rm f2} = 100{\rm k}\Omega$ and $R_{\rm in} = R_{\rm Q} = 100{\rm k}\Omega$. Use precision resistors if possible.

(!) These external resistor values should give a centre frequency of ~ 500Hz and a Q of slightly greater than unity. Connect the FG output to $R_{\rm in}$ and use the scope in the two–channel mode to observe both the FG output and the bandpass output of the filter. Connect the FG TTL output to the digital counter for a readout of frequency. Set the FG for a 1V peak–to–peak sine wave. Observe the bandpass output as the FG frequency is varied through the centre frequency, f_0.

[?] What happens to the bandpass output at f_0?

(!) To measure f_0 accurately, switch the scope to produce an xy–plot (Lissajous figure) of filter output *vs.* filter input. At the centre frequency the bandpass output should be exactly 180° out of phase with the input signal. Use the Lissajous figure to adjust the FG exactly to the centre frequency (see Experiment 1, and/or *Malmstadt* p.43 or *Brophy* p.63, for a discussion of Lissajous figures).

(!) Now switch the scope back to the dual trace mode and measure the peak–to–peak output voltage of the bandpass filter as a function of FG frequency over a range of 20 Hz to 20 kHz. Record 10–15 values in this range including several near f_0.

(!) Calculate and plot the filter gain in dB *vs.* log frequency.

[?] From the graph, determine the rolloff rate of the filter in dB/decade, on both sides of f_0.[1] Comment on the values you obtain.

(!) Now connect the scope to the low–pass filter output. Convince yourself that the device acts as a low–pass filter. Accurately measure and record the 3 dB frequency where gain $G = 0.707 \times G(\text{lowfrequency})$, and the phase shift at the 3 dB frequency.

(!) Repeat for the high–pass filter output.

(!) To get a filter with a higher Q, use $R_{\rm in} = 20\,{\rm k}\Omega$ and $R_{\rm Q} = 1\,{\rm k}\Omega$. Set the FG to give a sine wave with $V_{p-p} \simeq 0.5$V. Observe the bandpass output.

[1] A useful `physica` trick: `fit G=(a*f+b)*(f<=480)` will only fit $G(f)$ for values of $f = 480$ and below.

(!) Measure and plot the gain in dB *vs.* log frequency for the high–Q bandpass filter.

[?] Estimate the Q of the two bandpass filters you have investigated. Q can be measured as the ratio of the centre frequency f_0 of the bandpass output to the bandwidth (the difference in frequency between the upper and the lower 3 dB points).

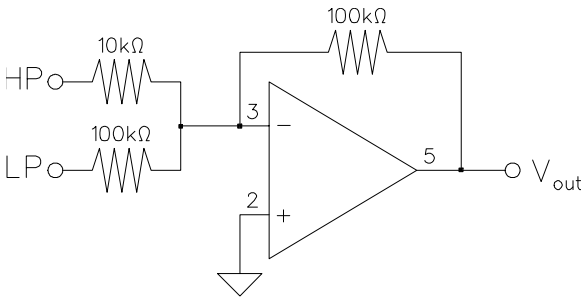
(!) Return the AF100 to the low–Q state, ($R_{\text{in}} = 100\text{k}\Omega$, $R_{\text{Q}} = 100\text{k}\Omega$). Vary the feedback resistors and measure the centre frequency of the bandpass output.

$R_{\text{f1}} = R_{\text{f2}}$	f_0, predicted	f_0, measured	% error
10kΩ			
50kΩ			
200kΩ			

3.2 Notch filter

A special form of active filtering can be thought of as the reverse of bandpass filtering. In analyzing a notch filter *we concentrate on the noise rather than the signal.*

(!)



AF100 has one additional, uncommitted, summing op–amp. It can be used to construct a notch filter by summing the low and high–pass outputs, as shown.
Wire up the above circuit. **Make sure you disconnect the grounding wire from pin 3!** Set the AF100 to $f_0 =$ 500Hz.

(!) Measure the frequency response of the notch filter. Choose the frequencies of the FG wisely: take a sufficient number of measurements to resolve the shape of the filter's transfer function.

(!) Plot the gain *vs.* log frequency.

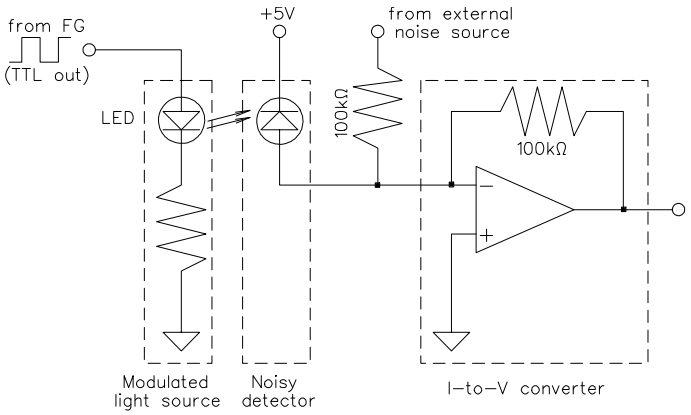
[?] What type of noise could be reduced using the notch filter?

[?] Determine f_0 from your plot. How does it compare with the expected value?

3.3 Lock–in amplifier

One of the best ways to dicriminate against noise is to use a lock–in amplifier. It combines the techniques of signal modulation at the source, band–pass limitation, and phase–lock demodulation to provide ability to distinguish weak signals "buried" in the noise. Because they actively modulate the source signal, lock–in amplifiers are capable of distinguishing signal and noise that have overlapping frequency spectra.

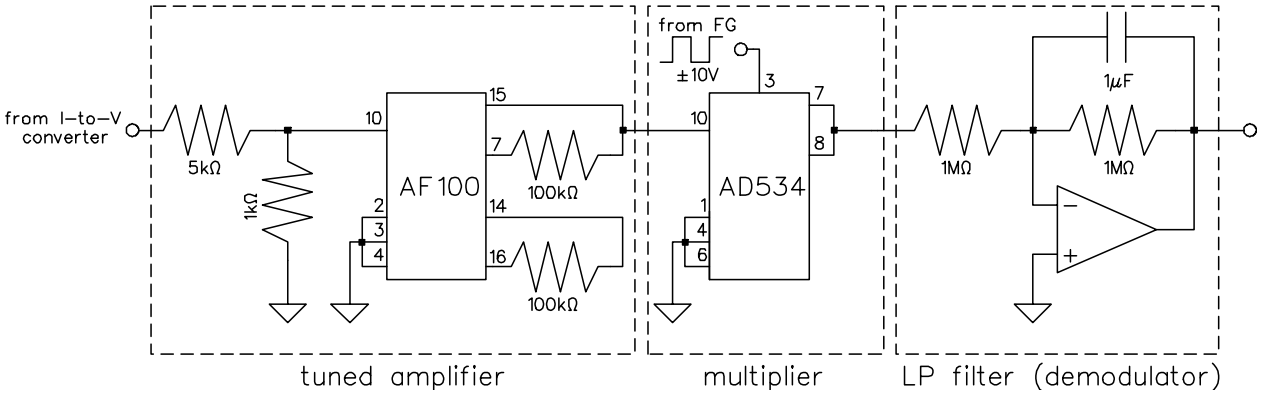
(!)



Connect a biased photodiode or a phototransistor to a I–to–V–converter. Connect an LED to the TTL output of the FG set at about 500 Hz. Do not connect the external noise source (NM on the job board) yet.
Use the scope to observe the output of the converter and adjust the position of the photodiode and/or the gain of the amplifier until the square–wave component of 50 to 100 mV is obtained at the output.

Make note of the DC level, the square–wave amplitude (p–p), and the approximate noise amplitude (p–p) in the output signal.

[?] Why is there a DC component in the output of the I–to–V–converter?



(!) Now connect the AF100 tuned amplifier circuit to the output of the I–to–V converter. Observe the tuned amplifier output with the scope. Adjust the FG frequency to get the maximum output from the tuned amplifier. Record your values of tuned amplifier output voltage (p–p), waveshape appearance, DC component of the output voltage, and the FG frequency setting.

[?] You should have observed a sine wave at the tuned amplifier output. The input, however, was a noisy square wave with a DC component. Explain the difference in input and output waveforms.

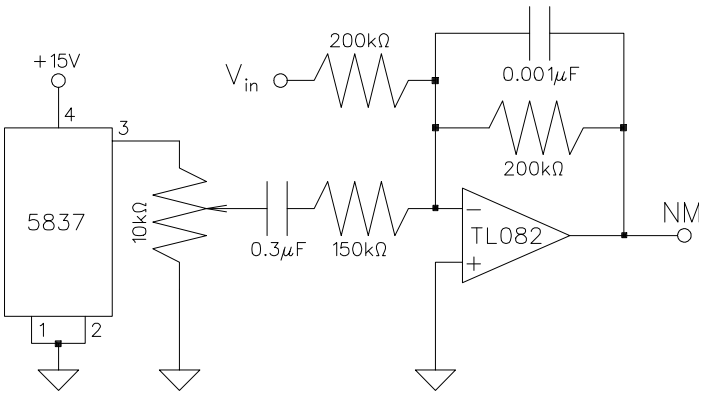
(!) The analog multiplier and low pass filter (phase–locked demodulator) should now be connected. The tuned amplifier output is multiplied by a square wave that is synchronous with the LED modulation. Adjust the FG square wave output to supply a ±10 V reference signal to the multiplier. Observe the multiplier output. Adjust the FG frequency carefully to obtain a waveform that most closely approximates a full–wave rectified sine wave. Draw the observed multiplier output waveform. Label the axes.

(!) Connect the active low–pass filter to the multiplier output. Observe the DC output with the scope. Record the DC level observed with the modulated LED on and off.

(!) Look again at the I–to–V converter output and measure the ratio of the square–wave amplitude to noise amplitude.

[?] Calculate the signal–to–noise (S/N) ratio improvement obtained with the lock–in amplifier.

(!) To better demonstrate the noise rejection capabilities of the lock–in amplifier, still more noise will be intentionally added to the signal. This noise will be obtained from the noise generator circuit available on the reference job board.



The relevant part of the reference job board circuit is shown. The 5837 digital noise generator IC produces 10 V pulses that have varying durations. The pulse durations are random integer multiples of 20 μs. The 10 kΩ potentiometer selects a fraction of the noise generator output amplitude. The noise signal is AC–coupled into a summing amplifier that also serves as an active low–pass filter.

The additional input to the summing amplifier allows the noise generator signal to be added to another signal: $V_{\mathrm{NM}} = V_{\mathrm{in}} + \mathrm{noise}$

Vary the 10 kΩ potentiometer to obtain maximal noise amplitude. Sketch the waveform observed on both sides of the coupling capacitor and at the output of the summing amplifier. An oscilloscope time base of 20 μs/div is recommended.

Also observe the output of the summing amplifier at a sweep speed of 500 μs/div. This output is labelled NM on the job board, for Noise Mixer output.

[?] Calculate the cut–off frequency of the low–pass filter in the NM. What is the attenuation of this filter for the frequency component that results from transitions every 20 μs (25kHz)?

(!) Connect the NM output through a 100 kΩ resistor to the summing point of the I–to–V converter (in the noisy signal source). Observe the converter output with a scope and adjust the noise generator output from zero until the square wave becomes difficult to see. (Trigger the scope from a clean square–wave or TTL output of the FG to avoid loss of synch.) Measure the DC output voltage of the lock–in amplifier, with the modulated LED on and then off. Compare again the signal–to–noise (S/N) ratios at the input and output of the lock–in amplifier.

[?] Explain why it is necessary to modulate the signal in order to obtain the improvement in S/N through the lock–in technique.

Experiment 4

PICLab project board

In this lab you will build your own microcomputer, Brock's own PICLab microcontroller project board. Based on a capable high-integration microcontroller (Microchip PIC16F8xx series IC), it features all the basic computer parts: CPU, on-board memory, ALU, etc., but also programmable input/output ports, hardware timers, analog-to-digital converters, etc. This makes it an excellent platform to base experimental Physics projects on.

4.1 Introduction

The vast majority of computers in the world do not run Windows, Unix or Linux. They do not execute word processing or multimedia applications. These are the computers that run appliances such as your television, VCR, microwave, and cell phone. These intelligent devices are known as embedded processors, microcontrollers or peripheral interface controllers (PICs). They are used to perform specific repetitive tasks that require low computational resources such as disk space or high throughput video processors, and little or no human intervention.

In contrast to the typical number crunching desktop computer, these devices excel in their ability to communicate with the world around them. To this end, a microcontroller IC not only implements the basic arithmetic and logical functions of a typical microprocessor, but also includes a variety of programmable input/output ports, hardware timers, analog-to-digital converters, and a fast and efficient means of interrupting the execution of the microcontroller program to service a variety of external or internal events.

A very capable example of a microcontroller is the Microchip PIC16F877. This 40-pin IC includes an 8-bit reduced instruction set (RISC) processor with 35 instructions, 8k words of re-writable (flash) program memory, 512 bytes of scratchpad (RAM) memory and system registers, 256 bytes of electrically-re-writable (EEPROM) data memory. There are 33 programmable input/output pins, an 8-channel analog to digital converter (ADC), three event counters/timers, and a universal synchronous/asynchronous receiver/transmitter (USART) capable of communication at up to 1.25Mbits/s. With a 4 MHz clock oscillator, each instruction requires 1 μs to execute. The device will operate at up to 20 MHz and execute five million instructions per second. This microcontroller can be programmed in circuit with an in-circuit serial programmer (ICSP) or it can reprogram itself by downloading a new program via the serial (COM) port of a PC or terminal.

Brock's *PICLab microcontroller project board* is compatible with the Microchip PIC16F8xx se-

ries of ICs. This family includes two 40-pin versions, PIC16F874/877, and two 28-pin versions, PIC16F873/876. These chips are functionally identical but differ in the number of input/output pins, and the size of the program and data memory.
The PICLab project board includes a variety of peripheral circuits intended to simplify the development of a microcontroller based project. Included are the circuits required to drive a 7 segment LED display, an interface to an LCD display, a keypad, a serial RS232 or USB interface, relays and current drivers for the control of external devices and an in circuit programming interface. There is also a small prototyping area for the inclusion of extra components. The PICLab can be powered from a 9 V DC "wall wart", a battery, or it can extract power from a computer's USB port.
A fully assembled PICLab board can operate as *a stand-alone device*. A five button expandable keypad can be used to input data and control the operation of the project board. For the display of output data, a four digit seven-segment LED display can be utilized. Alternately, a more elaborate LCD alphanumeric display of 2 lines of 16 characters each can be used. This "intelligent" display has its own character memory and is programmed with a set of commands, much like the microcontroller chip itself. This device might be used as a programmable thermostat, an alarm clock, or a battery powered portable instrument such as a digital voltmeter.
A PICLab board also can operate as *a remote device*. Connected via a serial RS-232 port, or a much faster USB port, a computer or terminal can accept and display the PICLab's output data, send the PICLab commands, and even change the program that the microcontroller is executing. Connected to a modem (modulator/demodulator), the PICLab could send an alert via the telephone to inform that the system needs attention. This device might be interfaced to several motion detectors and used as an intrusion alarm system or other household monitoring device, or as a remote data acquisition module.
The PICLab project board was designed at the Physics Department specifically as a convenient platform for several experiments in this course. Later on you will learn the basics of Assembly language programming, A/D and D/A conversion, and other aspects of computer assisted data acquisition and control. In this experiment, you will build your own PICLab workstation, by assembling (soldering) a project board of your own.

4.2 Pre-assembly review of parts and tools

Be sure to examine the schematics diagram of the project board, provided separately. You are not expected to understand all of the details yet, however, you need to learn to recognize the overall relationship between what is on the schematics, and its physical implementation on the project board. The locations of various components on the printed circuit board (see below) are well marked.
Examine, in particular, the keypad part of the circuit diagram. What should happen when you press various normally open (N/O) momentary switches? Note how instead of multiple binary logic lines to the PIC, multiple switches are connected to a single ADC input. Measuring the voltage on this line, the PIC can determine which of the switches is pressed.
The project board is a high quality double sided **printed circuit board**. The conductive traces on the fiberglass substrate are Pre-tinned for ease of soldering and both sides of the board are covered with a solder mask to minimize the possibility of solder connections between adjacent traces. To simplify parts placement, the top or component side of the board is silk-screened with the various

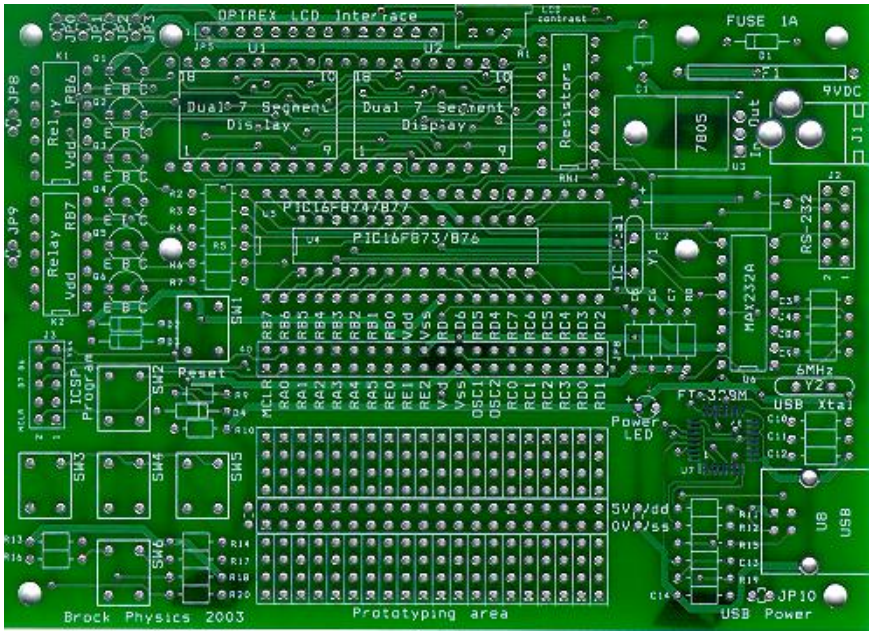


Figure 4.1: A PICLab printed circuit board, version 1.0, the component side

part outlines and corresponding part IDs. All soldering is done on the opposite, or bottom side of the board.

You will be using version 2.0 of the PICLab board, an updated of version 1.0 that no longer supports the RS232 interface or 28-pin PICs but includes a prototyping area with several types of surface-mount pads as well as a TLV431 voltage reference for the A/D converter. Space for two user-configurable trimmer potentiometers is also included.

Table 4.1 lists the components required to assemble a USB-powered printed-circuit board. These component are through-hole parts, inserted and then soldered at their proper location. The PIC itself, and the two 7-segment LED displays are socketed: the components are inserted into the socket in the final step of the assembly. Several other components, such as a voltage regulator and power jack, are required if the board needs more than 250mA of current to operate. In this case, a battery or AC adapter can be used.

Soldering

You will be using a variable temperature soldering station for all your soldering. Turn on the power and set the temperature so that the green LEDs light up but not the red ones. The soldering station may take a minute or two to reach the selected temperature. While you are waiting, moisten the tip-cleaning sponge.

The reliability of your project depends greatly on the quality of your solder connections. Please review the reference materials on soldering techniques provided on the course web site; they contain illustrations that may give you a good idea of what is expected. The following guidelines are a brief summary.

- Each time that you make a solder joint, begin by cleaning the tip of the soldering iron with the moistened sponge, then "tin" the iron by applying a small amount of solder to the tip.

Table 4.1: PICLab board v2.0 basic parts list

	#	item	part ID	circuit	function
	1	100Ω resistor	R9	Reset	current limiting resistor
	1	47KΩ resistor	R10	Reset	current limiting resistor
	1	1N914 diode, glass	D4	Reset	blocking diode during programming
	1	N/O mini switch	SW1	Reset	normally open reset switch
	1	40 pin IC socket	U5	PIC	for PIC 16F877 controller
	1	20.00MHz crystal	Y1	PIC	microcontroller oscillator crystal
	2	22 pF capacitor	C5,C6	PIC	oscillator capacitors
	1	0.1μF capacitor	C7	PIC	decoupling capacitor
	1	10 pin header	J3	PIC	ICSP Program interface
	1	40 pin IC socket	U1,U2	Display	for RT-DDC563DSA 7-segment displays
	8	330Ω resistor	RN1	Display	segment current limiting resistors
	4	2N4401 transistor	Q1-Q4	Display	7-segment digit driver transistors
	4	2.2KΩ resistor	R2-R5	Display	transistor base current limiting resistors
	5	N/O mini switch	SW2-SW6	Keypad	normally open keypad switches
	1	10KΩ resistor	R17	Keypad	voltage divider pullup resistor
	1	4.7KΩ resistor	R18	Keypad	SW2 voltage divider resistor
	1	8.2KΩ resistor	R20	Keypad	SW3 voltage divider resistor
	1	13KΩ resistor	R13	Keypad	SW4 voltage divider resistor
	1	22KΩ resistor	R14	Keypad	SW5 voltage divider resistor
	1	47KΩ resistor	R16	Keypad	SW6 voltage divider resistor
	1	500mA solid state fuse	0.5A	PSU	yellow disc circuit breaker
	1	red LED	LED	PSU	power on LED, longer lead is + anode
	1	470Ω resistor	R8	PSU	power LED current limiting resistor
	1	6.00MHz crystal	Y2	USB	USB interface oscillator crystal
	1	0.033μF capacitor	C10	USB	decoupling capacitor
	2	22 pF capacitor	C11,C12	USB	oscillator capacitors
	1	0.01μF capacitor	C13	USB	decoupling capacitor
	1	0.1μF capacitor	C14	USB	decoupling capacitor
	2	27Ω resistor	R11,R12	USB	current limiting resistors
	1	1.5KΩ resistor	R15	USB	pull-up resistor
	1	470Ω resistor	R19	USB	resistor
	1	USB B-type connector	USB	USB	USB cable connector

Table 4.2: PICLab board v2.0 optional on-board power supply parts list

	#	item	part ID	circuit	function
	1	1.0A solid state fuse	1A	Power	yellow disc circuit breaker
	1	100μF/10V capacitor	C1	Power	output filter capacitor
	1	100μF/25V capacitor	C2	Power	input filter capacitor
	1	1N4004	D1	Power	polarity reversal diode
	1	3 pin header		Power	USB/VDC power select
	1	7805	7805	Power	5V regulator
	1	Power jack 2.1mm	J1	Power	external 9VDC voltage input

This procedure will result in better transfer of heat from the iron to the parts to be soldered.

- Apply the tip of the iron where the component lead and the PC board copper trace meet so that *both* are heated at the same time. Apply the solder to the side *opposite* the tip. Do not touch the solder with the iron tip. When both the lead and the trace are sufficiently hot, the solder will melt and form a connection. This may take one or two seconds. Apply only sufficient solder to cover the joint.
- Withdraw the tip without disturbing the solder joint and let the joint cool. A good joint will be smooth and shiny and show a visibly solid connection between the copper trace and the component lead. When insufficient heat is applied to a joint, the solder will fail to flow around the connection and will bead and form globules, resulting in a "dry" joint. To correct this, reheat the joint until the solder melts, apply a touch more solder and let cool.
- When soldering a two lead component such as a resistor or diode, insert the component into the PC board so that it rests flush with the board's surface, then slightly bend the leads outward where they meet the board. Solder both leads and when cooled snip off the excess lead where it meets the solder joint.
- When soldering a component with several connections such as an IC socket, insert the component flush with the board and hold it in place as you solder the corner pins to the board. If the socket is not properly seated, apply some pressure to the raised region and heat the solder joint. After you are satisfied that the part is flush with the board, solder the remaining pins.

4.3 Assembly of a PICLab project board

The parts IDs are laid out on the board as text on paper, with the lowest index at the top left corner and ID numbers progressing in rows to the lower right corner of the board. To attach the various components to the printed circuit board, adhere to the following assembly sequence. Check off each step as it is completed. Note that many components are polarized and require to be placed on the board in a particular orientation. Follow the philosophy of checking component placement twice and soldering once. The removal and replacement of improperly installed components can be a tedious, time consuming process and if improperly carried out, can lead to board damage.

Note: before proceeding with the assembly, thoroughly read the following instructions in their entirety.

Before soldering any components to the project board, familiarize yourself with the proper *location and orientation* of all the components. Verify that you are installing the correct parts as specified in Table 4.1. If you are uncertain as to the value of a particular resistor, measure it with a multimeter. As you go along, you may find it useful to mark off the steps already completed.

- (!) Locate the 100 Ω reset circuit resistor R9 and verify the value with an Ohmmeter. Bend the leads at a right angle where they meet the body of the resistor. You can do this by applying pressure to the end of the resistor body with the tip of your finger. Be sure to make a tight angle otherwise the part will not fit into the board. Avoid bending the leads many times as they will likely break off. With the PC board component side up, install resistor R9 flush with the PC board, then bend the leads outward to hold the part in place. Turn over the PC board and solder the resistor leads. Snip the excess lead lengths *after* the solder joint has cooled.
- (!) Repeat the procedure to install the 47 KΩ reset circuit resistor R10.
- (!) Install diode D4. The diode has a glass body with a thin black band at one end to indicate the negative cathode. The band should be oriented in the same direction as the part outline on the PC board. Solder and trim the leads.
- (!) Install the reset switch SW1. The pins have an S shaped bend designed to hold the part in place during the automated assembly process. You will need to carefully straighten the pins of all the switches with pliers so that they can be inserted into the PICLab board. Be sure that the switch is flush with the PC board, then solder it in place.
- (!) Install the 20 MHz PIC oscillator crystal Y1 and oscillator capacitors C5 and C6.
- (!) Install the power LED, noting that the negative side of the diode is identified by the notch at the base, the LED current-limiting resistor R8 and 500mA solid-state fuse, a small flat yellow disk, at the location marked 'Fuses, 0.5A'.
- (!) A three-pin jumper, next to the fuse, can be installed to select the source pf power to the board, either from the USB connection (USB) or from an on-board power supply (VDC). To power the board only from the USB interface, connect a piece of wire from the middle hole to the end hole labeled USB.
- (!) The location RN1 for the display segment current limiting resistors can accept a resistor network, an IC that integrates eight resistors in one package, or discrete resistors. You will use eight discrete 330 Ω resistors for this purpose. Install side by side and solder the eight resistors at location RN1.
- (!) Install the four 2.2 KΩ resistors R2 to R5 that limit the base current of the transistors Q1 to Q4.
- (!) The 2N4401 (or 2N3904) transistors Q1 to Q4 must be properly nstalled. Hold the transistor upright with the part number facing you and the three legs facing downward. From left to right, the legs are identified as E-B-C. The placement of these legs should conform with the markings on the PC board. Generally, the three legs are clearly marked on the transistor

body. Add these transistors to the PC board. If you *are not sure* as to the proper orientation of the transistors, ask the instructor.

(!) Install the five keypad switches SW2 to SW6.

(!) Install keypad resistors R17, R13, R14, R16, R18, R20. Be careful to place these resistors at their proper location, otherwise the keypad will not function properly. Check their resistance with an Ohmmeter. The reading should be within a couple of percent of the required value.

(!) With the project board component side up, insert a 40-pin IC socket for the PIC controller at location U5 on the board. One end of an IC socket is usually indexed with a cutout or some other identifying mark to properly orient the removable IC in the socket. Be sure that the socket orientation corresponds with the indexed outline on the board.

(!) With the board solder side up and the IC socket flush with the surface of the board, solder the four corner pins to the board. Check that the socket is properly seated. If it is not, gently apply pressure to the socket and apply some heat to the pin to melt the solder and seat the socket. Solder the remaining pins, being careful to not apply too much solder and short out adjacent pins.

(!) Repeat the above procedure to mount the 40-pin socket for the seven-segment displays U1 and U2. Here we are using a socket to allow for the displays to be removable. Orient this IC socket with the index mark next to pin 1 of U1.

(!) Locate the 10-pin header J3 required for in-circuit serial programming. The location is labeled "ICSP Program". Insert the shorter end of the header strip flush with the board and solder it in place.

(!) The FT232BM USB interface chip has been pre-soldered to the board. Typically, for proper installation, a surface-mount component requires a fine-tipped soldering iron and very thin solder as well as the aid of a magnifier. Install the other USB-related components: the resistors R11, R12, R15 and R19, then the capacitors C7 and C10-C14, and finally the 6 MHz crystal Y2, making sure that the metal case does not contact the pads of C10.

(!) Install the silver USB-B connector by gently pressing the jack into the mounting holes while being careful to make sure that the four small signal wires are properly inserted and protruding from the other side of the board.

Carefully check over the entire board. You can use the illuminated magnifier to verify that all of the solder joints are of good quality and that the components are installed at the correct locations and in the proper orientation. Fig.4.2 shows you what your completed PICLab project board should look like.

4.4 PICLab basic functionality tests

Before putting your project board to a practical use, you must verify that all of the board's components are functioning as expected. To begin with, always establish that the correct voltage is present and distributed throughout the board.

(!) Have the instructor check your board before you perform the following tests.

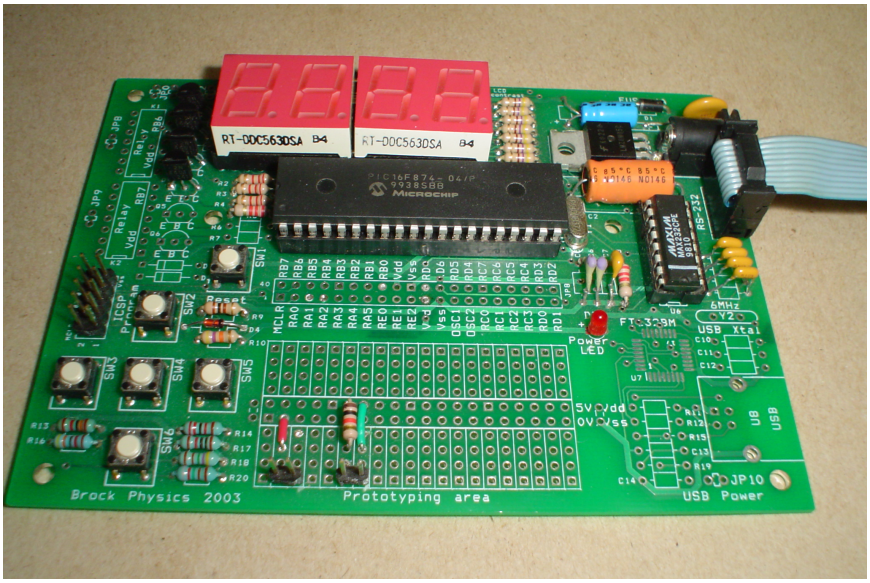


Figure 4.2: A completely assembled PICLab project board

(!) Connect a USB cable from the board to the host computer. The LED should light up. If it does not, the LED may have been inserted backwards. With a voltmeter, verify that there is 5V DC at the Vdd pin of the PIC expansion bus.

(!) Test the reset circuit. This circuit sets the voltage at the MCLR pin of the controller. A low voltage at this pin resets the processor while a level of +5V puts the processor in run mode. With the Reset switch released, there should be +5V DC present at the MCLR pin of JP8. Press the Reset switch SW1. The voltage should drop to OV. Release the switch to return the reset line to +5V.

(!) If the above tests have been successful, remove power from the project board. Ground yourself by touching the metal case of an instrument, then install the PIC microcontroller chip on the board. Be sure to properly orient the chip in the IC socket. Without touching the pins, carefully line up the PIC chip with the socket so that all the pins are lined up with the socket below. Gently and evenly press the chip into the socket, being sure that none of the pins are out of alignment and being bent, until it is fully seated in the socket. If the chip resists installation, see the instructor.

(!) Install the two dual 7-segment display ICs. Note the correct orientation. The decimal points of the display should be at the bottom of the display, toward the PIC. Install the first display IC flush with the right side of the socket. The second display is installed flush with the first. The two leftmost pins of the socket will remain empty. That is to say, the displays should appear offset slightly to the right on the socket.

(!) Noting the correct orientation, install the MAX232 serial interface chip into the 16-pin socket following the directions outlined above.

(!) Reconnect power to the project board. If the PIC circuit is functioning properly, the PICLab will test the 7-segment display by displaying the number 8888. The PICLab will then blank

the display and wait for user input. Press the Reset button. The PICLab should once again display 8888, then blank the display. If this happens, the PICLab microcontroller and display circuits are functioning as expected.

The keypad now needs to be tested. The state of the keypad is encoded as specific voltage levels at ADC input channel 0. The switches are organized as follows:

```
                SW2 = '2'

SW3 = '3'   SW4 = '4'   SW5 = '5'          SW2 + SW3 = '1'

                SW6 = '6'                        none = '7'
```

The keypad test routine verifies that the keypad resistors were correctly installed by displaying the switch number on the LED display when a switch is pressed. The following procedure causes PICLab to enter a diagnostic mode that displays keypad data.

(!) Press and hold one of the keypad switches. Press and release the reset button. The number 8888 should be displayed, followed by a number that corresponds to the keypad button currently pressed.

(!) Release the switch. The number '7' should appear. Press each of the keypad switches in turn and verify that the number corresponding to the switch is displayed. If the number output does not match the switch pressed, an incorrectly valued resistor has been installed. Simultaneously press SW2 and SW3; the digit '1' should be displayed. Press the reset button to exit the diagnostic routine.

Now you can test the operation of the PICLab USB serial interface. The PICLab board is controlled and programmed via a connection to the `picl` software running on a host computer. `picl` can automatically detect the presence of the PICLab board. More on this later...

(!) With the PICLab board connected to the host computer, login to your workstation and type 'picl' at the command prompt. The `picl` software should start by opening a 'PICL' window on your desktop, as well as a 'PIC simulator' window. At the top left corner of the 'PICL' window, an icon displaying a single plug shows that the PICLab board is not currently communicating with the `picl` software and `picl` software is running as a PICLab simulator. In this mode, your programs are executed on a virtual duplicate of the PICLab hardware.

(!) Check that the port is set to '/dev/ttyUSB0'. Click on the connection icon; it should change to a connected pair of plugs and a message 'Connected to PICLab at 57600 Baud' should be displayed in the status box. The 'PIC simulator' window disappears.

(!) From the 'Options' menu, click the 'Reset PIC' button. The PICLab board should momentarily display the '8888' and then blank, just as if you had pressed the Reset button on the board.

Once all of the above tests have been successfully carried out and all problems have been resolved, your PICLab is ready for use.

Experiment 5

PICLab programming

Brock's own PICLab is the development board package that will be used in several experiments in this lab. The microcontroller used in PICLab is the Microchip PIC16F877-20 or PIC16F887. With a 20-MHz oscillator, most instructions require 200 ns (4 clock cycles) to execute. In addition to the PIC itself, the PICLab board contains power supply, display, and interface circuits necessary to communicate with the board via a USB port of a Linux workstation or PC. The goal of this lab is to learn how to add operating softwaer to this hardware platrform.

PICLab bootloader

A small bootstrap utility program has been pre-loaded into the memory of your PICLab, and a computer program called `picl` has been written to provide transparent communications with the PICLab board.

`picl` IDE

`picl` is an Integrated Development Environment of the PICLab board. It allows you to write assembler programs, compile and download them to the memory of the PIC, and to examine the state of the PIC memory or registers during the debugging process. `picl` can also provide a real-time text and graphical display of data sent by your running program.

`picl` also includes a PICLab simulator. Implemented in software is most of the functionality of PICLab, including the internal hardware of the microcontroller and the external hardware elements such as the LED display, keypad, LCD display and serial port. The user subroutines that are preloaded on PICLab are also simulated in software. Hence, your code developed with the simulator should run as expected on the real PICLab.

With the simulator you can easily single step through your code and monitor the outcome of each instruction. Further, Virtual Pic allows you to view how an instruction is executed inside the PIC and the path that your data follows on every cycle of the PIC clock.

You can easily switch execution of your code between the simulator and the PICLab board by making or breaking the PICLab connection.

PICLab schematic

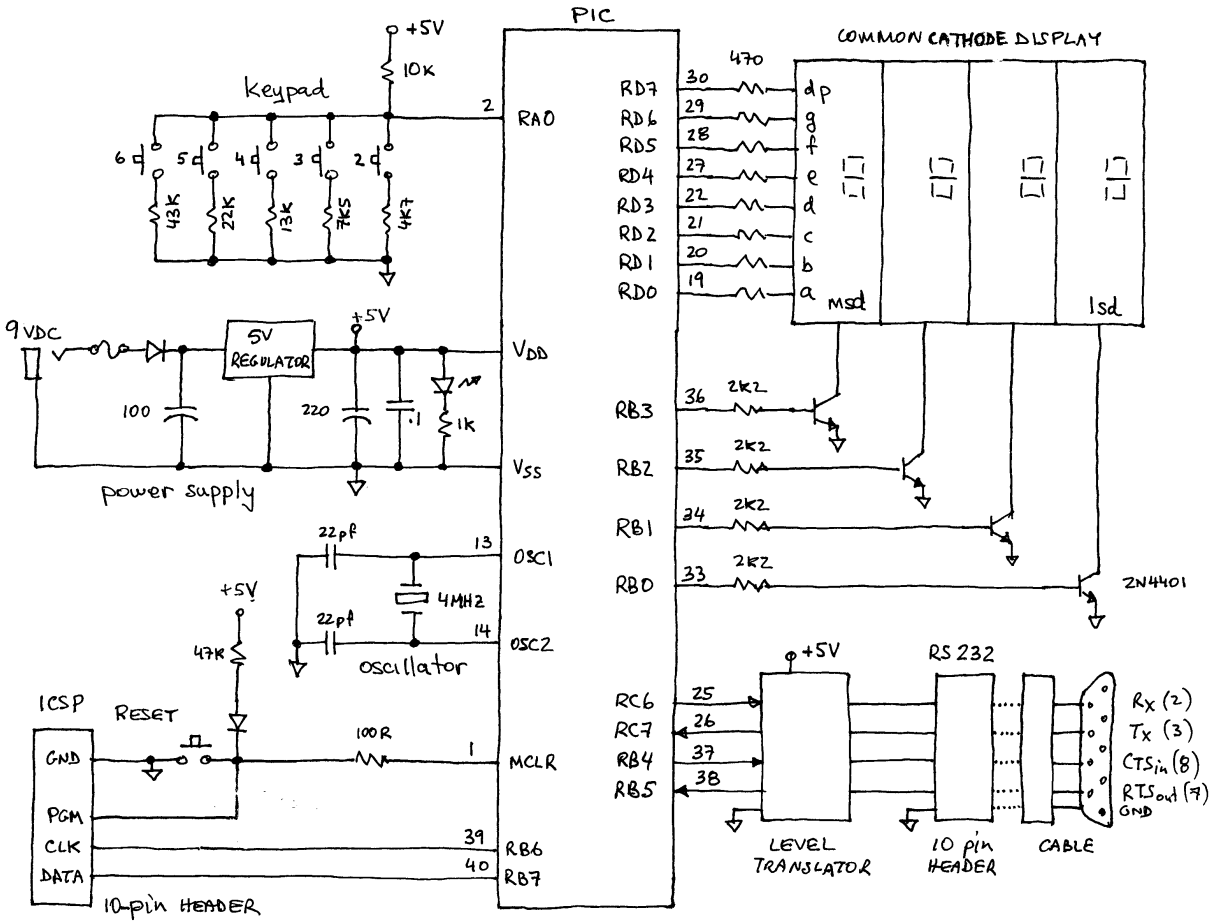


Figure 5.1: Block diagram of a PICLab board

Fig. 5.1 provides an overall block diagram of the PICLab board, showing the essential connections between the PIC itself and the other components of the PICLab board. Together with the `picl` help menus and the PIC reference documentation (see the References section of the class website) this information should be sufficient for you to get your PICs to work.

Connecting PICLab

By default, `picl` enters the PIC Simulator mode on start-up with the 'Connect' icon showing a single plug meaning that PICLab is not currently connected to PICL. With your PICLab board connected to the USB port of your Linux workstation and the port set to `/dev/piclab`, click the connection icon. The icon changes to two connected plugs, a message 'Connected to PICLab at 57600 baud" appears in the status box, and `picl` is ready to communicate with PICLab. Check the 'Auto connect' box in the 'Settings' menu to detect and connect to PICLab on start-up. [1]

PICLab interface

On your Linux workstation, invoke the graphical user interface to PICLab by typing:

```
picl &
```

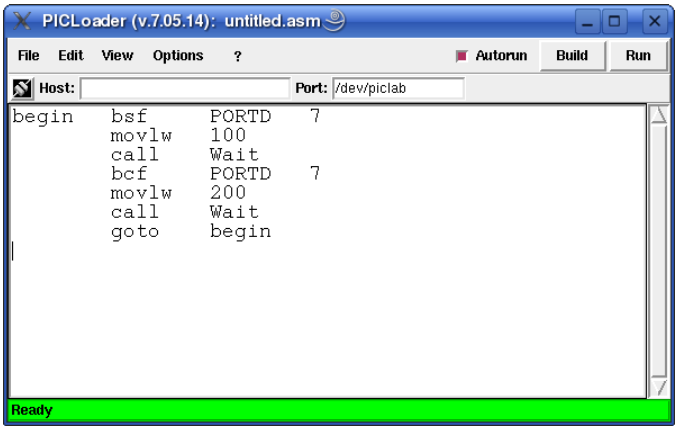


Figure 5.2: `picl` application window, connected to PICLab

Fig. 5.2 shows what `picl` window should look like on your screen. Typically, you enter one or more opcodes in the entry window and click **Build**. If the assembly completes without errors, PICLab loads the given instruction(s) to the Flash program memory and executes them.[2] This code will not erase if the PIC is reset or the power is turned off. You can execute the program currently stored in the PIC memory by clicking the **Run** button.
As the program runs, PICLab sends a variety of data back to the user. You can open several windows in the **View** menu to keep track of how the values in the PIC registers and memory change as a result of your instructions being executed.
Click the **?** button for help on the PIC opcodes, assembler directive commands and a list of the utility subroutines pre-written for you to use in your programs. Click 'Exit' in the 'File' menui when done, your working environment will be saved.

[1] To program PICLab from your laptop or PC, you need `picl` and the Tcl/Tk 8.4 interpreter installed; it is freely available for all platforms at `www.activestate.com`. You may also need to specify the connection: e.g. /dev/piclab or /dev/ttyUSBn under Linux or COMn under Windows, where n is the desired port. Under Windows, a USB serial device is identified as a COM port.

[2] By default, the loader chooses 0x0400 as the starting address of the user program, out of the total program address space of PIC of 0x0000–0x1FFF; the loader itself is using 0x0000–0x03FF.

5.1 Assembler instructions and code development

You are now ready to begin programming. As you progress through the exercises, be sure to understand the function of each instruction (see the help menu) and the overall logic of the program code. Familiarity with the PIC instruction set and with these basic technoques of interacting with PICLab will make programming the PICLab board a more pleasurable experience.

(!) Start the `picl` application. Connect to PICLab. In the following steps, the comments shown in brackets do not need to be entered; the text can be formatted using the tab key.

Begin by sending to the PIC an instruction to turn on bit 7 of its Port D. This pin is connected to the decimal point of the seven-segment displays. In the entry window, input the following opcode:

```
bsf       PORTD,7      ; set bit 7 of file register PORTD
```

Click the **Build** button. The LED attached to bit 7 of Port D on the PIC should turn on. Modify the above instruction to read as follows:

```
bcf       PORTD,7      ; clear bit 7 of file register PORTD
```

Clicking **Build** should turn off the LED.

(!) You will now implement a loop. Enter the following code:

```
begin     bsf       PORTD,7      ; set bit 7 of file register PORTD
          bcf       PORTD,7      ; clear bit 7 of file register PORTD
          goto      begin        ; branch to label called "begin"
```

Since the program is toggling the Port D bit on and off a couple of times every microsecond, the LED should appear continuously lit, but somewhat dimmer. The PICLab board is now executing an infinite loop and will not respond to commands.

Connect an oscilloscope to pin 7 of `PORTD` on the expansion connector. Sketch and label the output waveform.

[?] Explain the timing in terms of the PIC instruction execution times. Is the timing in agreement with your expectations?

(!) Press the **Reset** button on the PICLab board to interrupt the program and regain control of the hardware.

You can slow down the LED flashing rate by introducing a long, in PIC terms, delay after each of the bit operations. A utility subroutine called `Wait` is available to implement such a delay. The `W` register is loaded with a delay value to be passed to the subroutine. Try the following code:

```
begin     bsf       PORTD,7      ; set bit 7 of file register PORTD
          movlw     250          ; pass delay count to Wait subroutine
          call      Wait         ; execute a delay of 250*150us
          bcf       PORTD,7      ; clear bit 7 of file register PORTD
          movlw     250          ; delay value for Wait subroutine
          call      Wait         ; execute a delay of 250*150us
          goto      begin        ; branch to label called "begin"
```

The flashing of the LED is now clearly noticeable. Vary the value in the `movlw` instructions to observe how the flashing rate varies.

5.2 Loops, conditional branching, and calls to subroutines

The next step in our exploration of PIC programming is to add some flow control to the program's execution. One possibility is to have the algorithm flash the LED a set number of times, then terminate and return control to the user. You can use labels to make the program more readable. The `equ` directive assigns a value to a label. The following code will flash the LED `COUNT` times and terminate.

```
; Flash.asm: Program to flash LED on and off a specified number of times
COUNT_REG equ       0x20         ; use register 0x20 to count, 0..1F are reserved
COUNT     equ       0x10         ; count value to be put into the count register
DELAY     equ       0xff         ; "Wait" this many tics, ~150us ea

          movlw     COUNT        ; put a count value into the accumulator
          movwf     COUNT_REG    ; put accumulator into the count register

flash     bsf       PORTD,7      ; turn on LED segment
          movlw     DELAY        ; pass DELAY count to function Wait
          call      Wait
          bcf       PORTD,7      ; turn off LED segment
          movlw     DELAY        ; pass DELAY count to function Wait
          call      Wait
          decfsz    COUNT_REG    ; decrement count, skip over the next...
          goto      flash        ; ...instruction when file register=0
```

Another way to terminate a loop is to check for a certain condition and run until it is satisfied, such as when the user presses a button. The `Getkey` subroutine reads the keypad and returns in `W` a value of 2-6 if a button is pressed, otherwise a value of 7 is returned. The `STATUS` register maintains the state of several flags that can be tested to alter the program flow. The zero flag `Z` is set when the result of an operation is zero, and is cleared otherwise.

(!) Document the following code and test the algorithm by running the program. What does the program do? Does the program behave as expected? Consult the help menu (?) to obtain more information on the PIC opcodes and utility subroutines that are available for you to use.

```
; Showkey.asm:      Program to ....................................
KEYSAVE   equ       0x20          ; ....................................
          clrf      PORTD         ; ....................................

readkey   call      Getkey        ; ....................................
          movwf     KEYSAVE       ; ....................................
          sublw     7             ; ....................................
          btfsc     STATUS,Z      ; ....................................
          goto      readkey       ; ....................................

          movf      KEYSAVE,W     ; ....................................
          movwf     PORTD         ; ....................................
          sublw     2             ; ....................................
          btfss     STATUS,Z      ; ....................................
          goto      readkey       ; ....................................
          return                  ; required if code follows main program .
```

In the above example, a simple return from a subroutine (instruction `return`) is being used. In the `picl` convention, the entire user code is assumed to be a subroutine of the PICLab loader, and so at the very end of the code, an automatic return is always inserted for you. This is why your one-line "programs" like `bsf PORTD 7` worked just fine even though they did not have a `return` operation. However, you must insert an explicit return at the end of every subroutine that you yourself write, and at the end of your main program if any code follows it.

An extra `return` somewhere in the middle of the code can also be used as a simple debugging tool. Upon encountering such a "premature" return, the program will terminate, pass the control to the PICLab loader, and it in turn will update all of the open windows of `picl` with the current values of various registers, memory contents, *etc.* You will then be able to examine the current status of your PIC and decide if the code you wrote is doing exactly what you intended it to do.

You may also execute a `call Break` instruction to update `picl` with the recent values from the PIC, and to continue program execution. This subroutine is not a part of the PIC instruction set, but is made available through the utility loader function set. You can use the `!` symbol in a blank line as a short form for `call Break`.

Note: The `call Break` and `!` instructions may cause unexpected program behaviour when placed in your code following a conditional branch instruction or as part of a jump table.

In addition to the simple returns, the PIC instruction set has other flow control instructions that allow one to take some programming shortcuts. For example, `addwf PCL F` instruction increments the current program counter (PC) by a value stored in the `W` register before proceeding to the instruction stored at that location. In this way, an indexed goto statement is implemented. The `retlw` is a combined load-and-return instruction; it first loads the `W` register with a literal value and then exits by executing a return-from-subroutine instruction.

? You can use a lookup table to convert binary data into bit patterns that corresponds to a decimal digit on the seven-segment display. Determine from the PICLab schematic the mapping of `PORTD` pins to the display segments and write down the bit patterns that represent decimal digits 0 through 7 on the seven-segment display.

(!) Append the following code to the above program (that is why you needed the explicit `return` at the end of it) and convert your keypad data by calling the `Convert` subroutine at an appropriate time in your program. Explain the program flow of this code. What is the purpose of the `andlw` instruction?

```
Convert   andlw     7           ; ....................................
          addwf     PCL,F       ; ....................................
          retlw     %00111111   ; seven-segment bit pattern for digit "0"
          retlw     %________   ; ....................................
          retlw     %________   ; ....................................
          retlw     %________   ; ....................................
          retlw     %________   ; ....................................
          retlw     %________   ; ....................................
          retlw     %________   ; ....................................
          retlw     %________   ; ....................................
```

Each seven-segment display consists of eight LEDs connected together at the cathode (-). The `PORTD` pins connect to the individual LED anodes (+). With the common cathode pins of each display connected to ground, it would require 32 bits to control the four displays of the PICLab board.
A more efficient use of resources employs the technique of time multiplexing to generate an output on the four displays. Here, the digits are displayed sequentially with only one of the four digits enabled at anytime. If the switching between digits is sufficiently rapid, the persistence of the human eye creates the illusion that all the digits are on at the same time.
Four output pins (`PORTB` pins 0–3) control the voltage at the display cathodes via current-driving transistors. The corresponding anodes of the four displays are connected in parallel (refer to the PICLab schematic, Fig. 5.1). A software loop then enables each of the displays in turn while the bit pattern corresponding to that digit is presented on the `PORTD` pins. With multiplexing, the pin count has been reduced to 12 from 32, a saving of 20 input/output pins.

(!) Develop a flowchart to multiplex four different digits of data on the PICLab display. Convert the flowchart to PIC instructions and test your code.

(!) Vary the loop timing to determine the minimum refresh rate necessary to prevent the display from flickering.

5.3 Macros and subroutines

A Macro is a group of instructions that are referred to as a single new instruction. During assembly, every time a Macro instruction is encountered, the original group of instructions is assembled into the program.
A subroutine consists of a group of instructions within the user program that begin with a subroutine name tt label and end with a `return` statement. A `call label` instruction branches the program to `label` and executes the subroutine code until the `return` instruction restores the program flow to the instruction following the call.

The following code incorporates some practical programming techniques to implement a display multiplexing scheme. A macro definition is shown as well as an efficient method of implementing a jump table to select one of several branch possibilities.

(!) Analyse and document the code; explain clearly the functionality of the subroutines, then compare the functionality of this program with your version. You may want to save your code into a file (say, `Show4.asm`) before you build and run it. Be sure to add some of your own code to provide meaningful values to `7seg_0` ... `7seg_3`, then run the program.

```
; ..... Show4.asm: multiplex the four-digit seven-segment display ........

7seg_0    equ       0x20         ; least significant display digit .......
7seg_1    equ       0x21
7seg_2    equ       0x22
7seg_3    equ       0x23         ; most significant display digit ........
7seg_ptr  equ       0x24         ; pointer to current digit displayed.....

Move      macro     src,dst      ; register to register move, modifies W
          movf      src,W        ; the macro defines a new Move
          movwf     dst          ; instruction that is not available
          endm                   ; as part of the PIC instruction set

;         your code goes here, with a call to Scan7seg

          return

Scan7seg                         ; ....................................
          incf      7seg_ptr,F   ; ....................................
          movlw     3            ; ....................................
          andwf     7seg_ptr,F   ; ....................................
          movlw     0xf0         ; ....................................
          andwf     PORTB,F      ; ....................................
          call      Show7seg     ; ....................................
          iorwf     PORTB,F      ; ....................................
          return                 ; ....................................

Show7seg                         ; ....................................
          movf      7seg_ptr,W   ; ....................................
          addwf     PCL,F        ; ....................................
          goto      Show0        ; ....................................
          goto      Show1        ; ....................................
          goto      Show2        ; ....................................
          goto      Show3        ; ....................................
Show0     Move      7seg_0,PORTD ; ....................................
          retlw     %00000001    ; bit 0 selects display 0, active high ..
Show1     Move      7seg_1,PORTD ; ....................................
          retlw     %00000010    ; ....................................
Show2     Move      7seg_2,PORTD ; ....................................
```

```
            retlw     %00000100    ; ....................................
Show3       Move      7seg_3,PORTD ; ....................................
            retlw     %00001000    ; ....................................
```

The bits of a port are generally assigned various functions so you must take care to modify only the pertinent bits when using byte size instructions. This masking process requires reading the current port value, modifying only specific bits, then writing back the data to the port. Bit manipulation instructions are not useful when the bit to be modified varies, as in the selection of the digit to be displayed.

5.4 Interrupts

You will note that depending on the code that you added, the above program will initialize the digit values and display them indefinitely, or you will have implemented a loop that modifies the digit variables and calls the `Scan7seg` subroutine. In the first case, the PIC is fully occupied scanning the display and can perform no other function; in the second case the refresh rate is determined by repeated calls to a subroutine.
The ideal way to execute a periodic sequence of events is to use an interrupt. A hardware timer on the PIC interrupts the program flow every 5ms and executes a call to a user interrupt service routine (ISR). The ISR code runs in the background independently of the user program. You can, with an ISR, update the display variables or wait for input while the `Scan7seg` ISR scans the display at a constant rate. Note that the execution time of the ISR must be less than the time between interrupts or your program will hang. With this in mind, your ISR is disabled when the PIC is reset.
To define an interrupt service subroutine that will be remembered by the PIC until redefined, add the `#UserISRon` directive following your ISR subroutine code:

```
#UserISRon Scan7seg       ;set Scan7seg as user ISR and enable interrupt
```

The ISR routine will only execute while your program is running. To test some ISR code, you can program a one-line instruction (e.g. `here  goto here`) to execute an infinite loop; the ISR routine will execute until the PICLab board is reset. The user ISR routine can be turned on and off from within your program with the following instructions:

```
bcf       Flags,USERISR   ;reset user ISR flag, disable user ISR
bsf       Flags,USERISR   ;set user ISR flag, enable user ISR
```

5.5 Analog-to-digital conversion

The PIC can sample one of eight input channels with a 10-bit resolution. To perform an analog-to-digital conversion (ADC), an input channel is selected. A delay follows, to allow the input voltage to be sampled. A start of conversion flag is set to begin the ADC and another flag is set when the conversion is completed. The data is then ready to be used.
The `ReadAD` subroutine performs all of the above tasks. Load the `W` register with the number of the input channel and call the routine. After 50μs, the lower eight bits of data are returned in the `WL`

file register and the two most significant bits are in WH. The Getkey routine reads A/D channel 0 and uses the three most significant bits of the converted value to determine which key was pressed.

(!) picl has predefined pointers to the 7-segment LED displays called Digit0..Digit3, a subroutine equivalent to Scan7seg called Refresh and a routine LedTable, similar to your Convert routine, that converts a value 0-0x0F contained in W to the 7-segment pattern for the corresponding hex digit. For convenience and to make your code more compact, use these as part of your programs.

The following code reads a 10-bit value from the A/D converter channel connected to the keypad and displays it as three hexadecimal digits on the LED display. Complete the missing code and verify that the program functions as expected:

```
            #UserISRon  Refresh       ;define LED display scanning routine as user ISR

            bsf         Flags,USERISR  ;enable execution of user interrupt routine

begin       .....       .....          ;select the keypad channel
            .....       .....          ;read 10-bit A/D value, store in WH:WL
            .....       .....          ;place lower 8 bits of 10-bit A/D value in W
            .....       .....          ;set bits 4-7 to zero, bits 0-3 are A/D bits 0-3
            call        LedTable       ;convert value in W to 7-segment hex digit
            movwf       Digit0         ;display hex digit for A/D bits 0-3
            swapf       WL,W           ;place swapped nibbles (hex digits) from WL into W
            .....       .....          ;set bits 4-7 to zero, bits 0-3 are A/D bits 4-7
            .....       .....          ;convert value in W to 7-segment hex digit
            .....       .....          ;display hex digit for A/D bits 4-7
            .....       .....          ;place upper 2 bits of 10-bit A/D value in W
            .....       .....          ;convert value in W to 7-segment hex digit
            .....       .....          ;display hex digit for A/D bits 8-9
            .....       .....          ;loop code
```

5.6 Utility subroutines and data output

There are several pre-loaded subroutines available for use as part of your programs. Click on the help menu '?', and browse the 'Routines' subdirectories. You should become familiar with these routines; they are bug-free and will make your programming task much easier.
Several of these routines make the output and conversion of data a simple matter. For example, the Bin2BCD routine takes the 16-bit binary value stored in the file registers WH and WL and converts it to a five-digit signed or unsigned decimal value stored in registers Dec0-Dec4. The BCD2LED routine converts and outputs this result to the 7-segment display. Alternately, the contents of Dec0-Dec4 can be sent to the 'PICLab output' window in picl as an ASCII [3] string by calling the BCD2TCL routine, or to the LCD display by calling the BCD2LCD routine. The PICLab LCD display uses the ASCII character set. These routines require parameters to be set prior to execution.

[3] ASCII refers to the American Standard Code for Information Interchange, where an 8-bit value is used to represent the character set of alphanumeric characters a-z, A-Z, 0-9 as well as other symbols typically found on a keyboard, such as ?, +, !. Coded in the range of ASCII=0-0x1F, are non-printed control characters.

To send to `picl` a single ASCII character stored in `W`, use the `TxByte` routine. To send a value in `W` in the range of 0-9 as the corresponding ASCII decimal character, use the `TxDigit` routine. The `Hex2TCL` routine outputs the value in `W` as a 2-character hexadecimal string, while the `Dec2TCL` routine outputs the value in `W` in the valid range of 0-99 as a 2-character decimal string.
The characters sent to `picl` are stored in a buffer until a newline character ASCII=0x0A, is received. This is handy when several columns of data need to be sent; they will be displayed on the same line until terminated by a newline character. To separate your data values with a space, send the 'space' character, ASCII=0x20.
The following code segment outputs the value in `WH:WL` as a decimal string to the PICLab output window:

```
call      Bin2BCD   ;convert 16-bit value in WH:WL to decimal string
movlw     6         ;set field width for BCD2TCL to 6 characters
call      BCD2TCL   ;send string to TCL buffer
movlw     '\n       ;load W register with ASCII newline character
call      TxByte    ;send character, flush buffer to display contents
```

For the following exercises, begin by sketching a flowchart of the logical steps required to perform the given task, then convert each step to one or more PIC instructions that will define your program. Be sure to thoroughly document your code. Test your code initially on the PIC simulator, then execute your program on PICLab:

1. write a program that reads the keypad channel and displays the result as a decimal value 0-1023 on the 7-segment LED display. The value should change as the various keypad switches are pressed;

2. write a program that outputs three pairs of coordinate points (1,2), (2,4), (3,8) to the PICLab output window. The data should appear as an array of three rows by two columns. Click the **Graph** button to generate a Gnuplot graph of your data. Check the Lines box to interpolate the data with line segments;

3. write programs that implement in software the summing and shift/add algorithms used to multiply two 4-bit numbers that were implemented in hardware as part of Experiment 6. Begin by reviewing the arithmetic instructions available to the PIC and their effect on the carry `C` and zero `Z` flags contained in the `STATUS` register.

Experiment 6

Building and using a digital thermometer

Some things happen so fast that one simply cannot monitor them without help from a fast computer in the role of a data taker. One example is a rapid quench which occurs when a hot body is immersed into a cold fluid. Is the rate of cooling still proportional to the temperature difference?

(!) Assemble the thermistor circuit as shown in Fig.6.1.

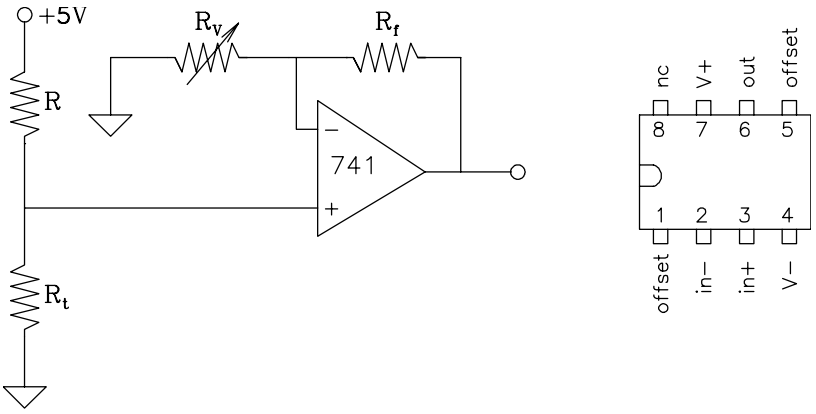


Figure 6.1: **Thermistor circuit.**
R, $R_f \sim 10\text{k}\Omega$; $R_V = 10\text{k}\Omega$ potentiometer; R_t = thermistor, R_t decreases as temperature increases

(!) Immerse thermistor and mercury thermometer into a beaker of ice-water at $t = 0°\text{C}$. Adjust R_V so that $V \approx 9.5\,\text{V}$ at 0°C.

Calibrate V as a function of t. Slowly heat the water — so as to maintain thermal quasi-equilibrium — and measure V and t at $\sim 5\,\text{s}$ intervals. Heat to $\sim 60°\text{C}$. Use little water for speed. Use the program you have written to measure V; record temperature readings from the mercury thermometer by hand.

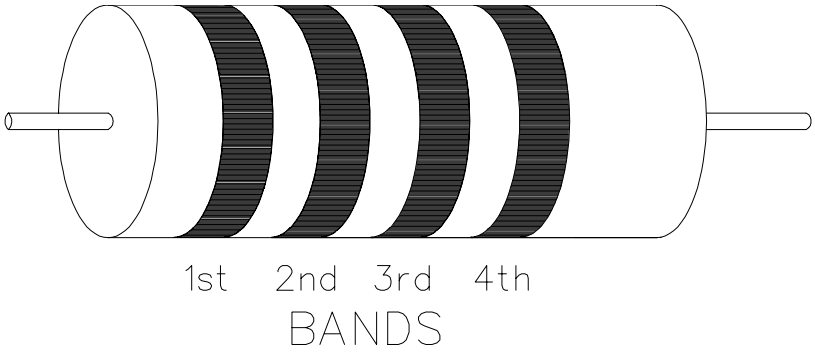
(!) Plot and analyze your data and create the temperature calibration plot of the thermistor circuit.

(!) *Optional:* modify your program to report true temperature in °C.

(!) Prepare two beakers, one with ice-water, one at a moderately high temperature. Immerse the thermistor/mercury thermometer assembly into the hot one. Modify your program to perform a frame grab of a large number of points. Start the program and rapidly transfer the assembly into the ice-water beaker. Use your calibration data to plot true temperature as a function of time and analyze your data, attempting to verify Newton's law of cooling (the rate of cooling is proportional to the temperature difference). Note and comment on the deviations from the exponential behaviour.

Appendix A

Resistor colour code



Colour	First Band	Second Band	Third Band	Fourth Band
Black	0	0	10^0	-
Brown	1	1	10^1	-
Red	2	2	10^2	-
Orange	3	3	10^3	-
Yellow	4	4	10^4	-
Green	5	5	10^5	-
Blue	6	6	10^6	-
Violet	7	7	10^7	-
Gray	8	8	10^8	-
White	9	9	10^9	-
Gold	-	-	10^{-1}	5% tolerance
Silver	-	-	10^{-2}	10% tolerance
No band	-	-	-	20% tolerance

For example, the resistance of a resistor whose bands are red, red, red, silver is

$$22 \times 10^2 \longrightarrow 2.2\,\mathrm{k}\Omega \pm 10\%$$