Brock University Physics Department
St. Catharines, Ontario, Canada L2S 3A1

Phys 2P32: Electronics (II)
Laboratory Manual

E. Sternin and F.Boseglav

Contents

Preface

Introduction

Mastering Electronics is not an easy task. While many concepts are straightforward, their application to a real-world device are often non-trivial. Part of the difficulty is that in addition to new concepts one often has to learn new numerical and algebraic tools that enable us to predict the values of various components to use, to select their settings and operating points for optimum performance. Putting it all together can be quite daunting.

In this laboratory you will use a variety of tools to achieve just that:

- hands-on experiments, where you will assemble real circuits using real components, meters, wires, and devices — workstations with multi-meters, function generators, oscilloscopes, programmable power supplies, and bread-boarding stations are provided for this purpose;
- computer-based tutorials using software called `Electronics Workbench`, where virtual circuits are assembled, tested and analyzed using the common graphical "drag-and-drop" skills;
- graphing and numerical analysis of the results of your real or virtual experiments, with the help of the `physica` software or its graphical version `physicalab`.

A typical lab experiment may consist of simulating a circuit, choosing the optimal value for some component, then assembling the very same circuit on the breadboard in the lab, testing it, and finally, analyzing your measurements and comparing them to the predictions of the theory learned in the lectures.

Lab books, reports and marking

Some lab sessions are devoted to the computer-based exercises using `Electronics Workbench`. As you go through the exercises, be sure to answer all the questions in your lab book and record pertinent observations. Screen capture a copy of all the circuits that you simulate and import them into your lab report. Be sure to save a copy of all the working circuits to your file space *before* you begin the simulation.

Other experiments involve actual electronic components and circuits. Sometimes you will assemble exactly the same circuits that you had simulated in an earlier experiment. A similar step-by-step write up in the lab book is expected. All of your individual observations and measurements must be included. Here you can capture the screen output and settings from the digital oscilloscope and save it to your lab report.

Following every lab you are required to submit a lab report analyzing and summarizing the data and the experimental procedures. The lab report should be typed, single-sided, and submitted in a clear report cover/document folder. The grading is based on the following:

- overall neatness and coherence in the structure of the report;
- completion of all the required simulated and experimental steps;
- inclusion of printouts, data tables, circuit and waveform sketches;
- thoughtful and understandable responses to the guide questions;
- adherence to the designated lab format.

  You will find it most efficient to open a wordprocessor document at the start of the lab and then enter observations, data, screen captures of circuits, graphs and oscilloscope traces as you proceed with the experiment. This way, the overall structure of the lab report will have been created and can be easily enhanced with further details and insights.

A lab report should start with an overall statement of purpose of the experiments. Then *for each exercise* include a schematic diagram of the circuit and graphs of the waveforms observed, formula derivations, a description of the theoretical behaviour of the circuit and comparison with your actual observations, and answers to the pertinent questions. The presentation of your results should be organized and complete, your diagrams titled and referenced, so that someone who is not familiar with the experiments would have no difficulty understanding what was done.

At the end of the lab report, include a brief Conclusions section that summarizes the results and discusses problems encountered and insights gained. The purpose is to reach the synthesis stage, to give you a chance to establish intermediate milestones in your learning.

Completed lab reports will be collected at the beginning of the next lab; thus you have a full week to complete your lab reports. However, you will find it easier to do the write-up within one or two days of the end of the lab, while the details are still fresh in your mind. There will be no time extensions given for late submissions. Late labs receive a zero grade; all labs must be completed with a passing grade to satisfy the lab component for the course.

Be prepared!

Some of the experiments will require the full 3-hour lab period; therefore, it is essential that you are fully prepared *before* attending your lab session.

- Arrive on time. Your lab starts at 2:00 pm sharp!
- Be sure you have read through the experiment *in its entirety* at least once before arriving.

Your partner cannot be expected to wait for you. Failure on the above two points can result in you working alone (time permitting), or being asked to make up the experiment some other time (depending on the availability of equipment).

Plagiarism

When you are working closely with your partner, it is understood that parts of the lab reports will appear similar; however, plagiarism will result in a *zero* for that report. It is your responsibility to consult the section on Academic Misconduct of the University Calendar prior to the first lab session.

Conventions used in this manual

(!) Whenever you see a paragraph marked off with this symbol, it indicates an experimental step. You are expected to perform one or several operations and write down your results and observations in the lab book.

[?] When you encounter this symbol, it indicates a question or a problem. You are expected to perform the necessary calculation (using pen and paper) and to provide a written answer and, possibly, a brief explanation in your lab book *before* you proceed to the next stage of the experiment.

Plotting and fitting with `physica`

An integral part of every lab is an analysis of the results, and it is best done with the help of a scientific visualization/plotting/fitting computer program. The Physics Department uses a plotting and fitting package called `physica`, written at the TRIUMF accelerator in Vancouver, BC. This is the recommended software for use in the analysis of experimental data and in the preparation of lab reports, theses, and scientific articles.

The main `physica` "engine" is an "old-fashioned" piece of software in the sense that it has a command language and requires typing of commands at the prompt, and not clicking a mouse and using visual widgets. On the other hand, it is easy to learn, its numerical engine is an extremely powerful one, and a macro language allows you to automate many tasks using only a text editor.

- A simple to use interface to Physica available only on the Physics Department computers is the `Physicalab` data acquisition and plotting software used in the first-year Physics labs. Open a terminal window and type `Physicalab` at the command prompt to invoke the program.

- In addition, `Physica Online` is a web-based interface into `physica` which may be accessed from any web browser. It is fairly self-explanatory and can be invoked by pointing a web browser to

  `http://www.physics.brocku.ca/physica/`

- For more advanced tasks, `Physicalab` and `Physica Online` provide an "expert mode" which allow access to full capabilities of `physica`. In order to harness the full power of `physica` you may need to spend some time learning its command language. i

References

In addition to your course textbook, numerous excellent introductory electronics books exist, and you are encouraged to refer to them often. Some selected titles are listed below, with Brock Library calling numbers shown where appropriate.

Other references such as manufacturers' data books and the equipment manuals should be consulted as needed; most of them are available online. The web page of the course has some select pointers in the section References and is a good place to start.

1. D. Barnaal, *Analog and Digital Electronics for Scientific Applications*. Waveland Press, 1982. TK7816 B34.

2. J.J. Brophy, *Basic Electronics for Scientists.* McGraw–Hill, 1990. TK7815 B74.
3. M.M. Cirovic, *Basic Electronics: Devices, Circuits, and Systems.* 1974. TK7815 C53.
4. A.J. Diefenderfer and B.E. Holton, *Principles of Electronic Instrumentation.* Saunders College Pub., 1994.
5. W.L. Faissler, *An Introduction to Modern Electronics.* J.Wiley & Sons, 1991.
6. P. Horowitz and W. Hill, *The Art of Electronics.* Cambridge University Press, New York, 1989. TK7815 H67.
7. H.V. Malmstadt, C.G. Enke, and S.R. Crouch, *Electronics and Instrumentation for Scientists.* Benjamin/Cummings Publishing Co., 1981.
8. R.E. Simpson, *Introductory Electronics for Scientists and Engineers.* Allyn and Bacon, Boston, 1987.

NAME (PRINT)	STUDENT ID	LAB DATE	GRADE

Experiment 1

Introduction to `Electronics Workbench`

1.1 Preliminaries

You may be quite familiar with using a general-purpose computer, such as a Windows PC or a Mac, but the computer environment you will find yourself in in the Electronics lab is more specialized, and may present additional challenges. Be sure to attend the introductory lecture offered by the Brock Information Technology Services (ITS) early on in the semester. The computing environment around Brock evolves quickly; your lab instructor should have the most up-to-date information.

Currently, the ITS operates several computer labs within the Department of Physics (B203 and H300 are the two largest ones), all of them deploying identical Linux-based workstations. Logging on to any of them with your Brock assogned Campus username/password will bring you into the exact same environment, and all the files in your home directory will be available to you (they are stored on one central network server, accessible from all workstations equally).

In addition, this course makes a heavy use of a circuit simulation program called `Electronics Workbench` (EWB, also known as the National Instruments Multisim due to a recent corporate change). Unfortunately, there is no Linux-compatible version of this software, and it has to be run from a separate central Windows server. You access this server using the same Campus username/password.

(!) Log on to the Linux workstation using the Campus username/password.

(!) Click on the OpenSUSE icon at the bottom left of your screen, then click on *Science* followed by *Electronics Workbench* to open login window to the Windows server. Enter the same Campus/username/password and a Windows Desktop will appear.

(!) Click on the EWB5 icon to start the program.

Move around and examine the menus and controls. Pausing a cursor over an unknown item should bring up a bubble of a description of what the item is. The simulation is controlled by the *on/off* switch at the top right corner of the screen. If you are lost, quit and restart the program.

1.2 Virtual circuits in `Electronics Workbench`

Electronics Workbench is essentially "an electronics lab in a computer", and the way it appears on your computer screen is shown in Fig. 1.1. The white field in the middle is the workspace into which you drag various components and devices found in the multiple parts bins divided into several categories, just above the workspace. When you then bring the mouse near the edges of

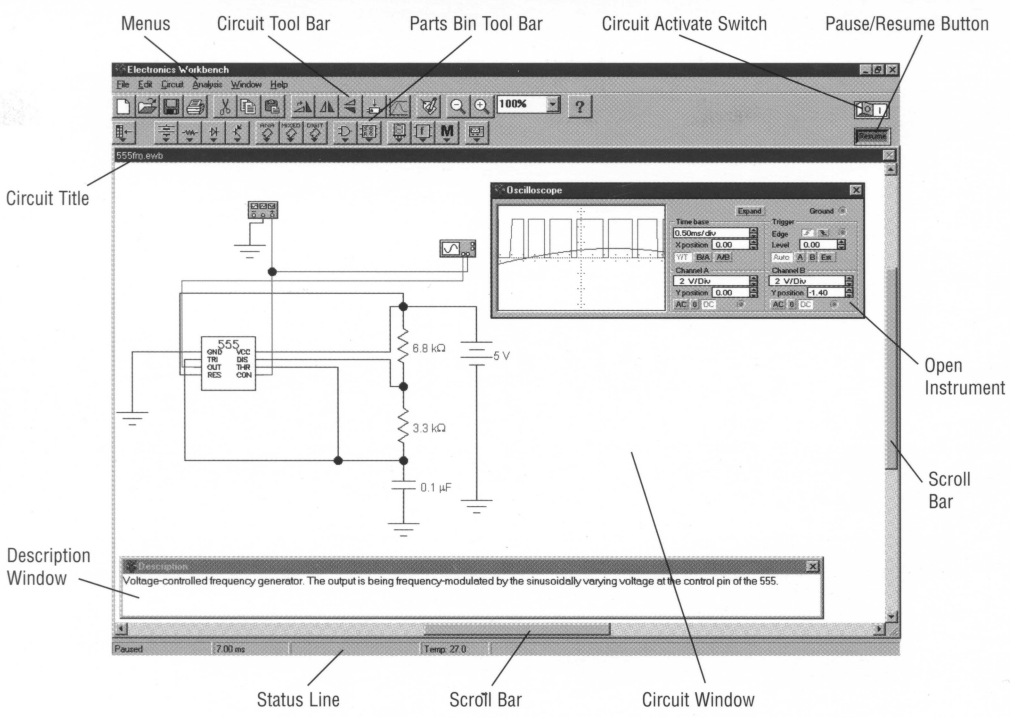


Figure 1.1: Electronics Workbench screen

each component, they turn into dark dots representing nodes of your future circuit. Click and drag until a line stretching out of a node reaches a node of another component, then release. You just connected a virtual "wire" between the components. The wires snap to a grid (which can be made explicitly visible through the Circuit menu), and as you move components around the wires stretch and follow as needed.

After a few mouse-clicks, you can assemble an entire virtual circuit that includes passive and active components, meters, oscilloscopes, and other virtual counterparts to the real devices and instruments found in an electronics lab.

There is one important difference to working with a virtual circuit. As you are putting it together, the program creates a set of mathematical equations that describe the circuit. As you then flick the virtual ON switch, the computer proceeds to solve these equations, quickly and with great precision, and reports and even plots the results. A variety of values can be swept through quickly and automatically, to discover the optimum ones; an entire frequency response curve can be obtained with a single click of a mouse.

What happens is that you are able to concentrate on the *physics* of the problem, and not on the sometimes tedious details of setting up and solving a fairly large system of coupled linear and differential equations. You do not need to be careful with the details of these calculations, and you concentrate instead on making sure you understood the behaviour of the circuit and how this behaviour relates to the underlying theory.

When you concentrate on the concepts and avoid applying by rote a memorized set of steps you are studying for mastery. When you understand what is going on behind the equations, you can apply that understanding to problems where the rote method is sure to fail. In our computer-assisted labs you will learn to test your understanding, to make up circuits and to predict the results mentally, then have the computer verify (or not!) your predictions. You will build up your intuition on the subject of Electronics. In some sense, your efforts will closely parallel what physicists do

every day in their research, something often called "the scientific method": organize your knowledge, develop a theory, make predictions, test them by experiment.

The results of this exploration need to be described in a complete, coherent fashion to allow a reader (or marker) familiar with the subject the opportunity to understand your methodology and attempt to reproduce your data. **Do not** assume that the reader has intimate knowledge of your experimental setup and goals.

For each of the following exercises, explain fully the procedures undertaken, your observations, followed by a short summary of your results. Include screen captures of all circuits, simulation output, data tables and meaningfully scaled graphs. Do not refer to the lab manual; include *all* pertinent details, in your own words, as part of the report. Support *all* statements and conclusions with experimental evidence or reasoned arguments.

Hint: You can screen capture Electronics Workbench output by going to the OpenSUSE start menu then clicking on *Accessories* followed by *Screenshot*. In the window that opens, select to capture a screen region, outline the area of interest by pressing the left mouse and moving the cursor to outline a rectangular region, the release the buttton and save the output to a file. You can conveniently include the schematic, instrument settings and output, as well as descriptive text (A, textbox from the Miscellaneous submenu) as one capture that can be saved and imported to a document. **Do not** append printouts to the end of the lab report.

1.3 Real and ideal meters

(!) Consult your notes or the Web to determine the internal configuration of a real (non-ideal) voltmeter and ammeter.

(!) Pull down a battery (in the sources bin, move the mouse over the icons and a bubble with the name appears) and a multi-meter (in the instruments bin) into the worksheet. Double-click the multi-meter icon for a close-up view. Verify the multi-meter is in voltage mode, *i.e.* that [V] is highlighted. Practice connecting/disconnecting the wires and moving the components around the worksheet. Move the components to verify that the wires are properly connected to them and that the electrical circuit is complete. Connections display a black dot.

[?] When do you see a positive reading on the meter? a negative one?

(!) While the meter is connected to the battery, switch it into the current mode by pressing [A].

[?] What happened? Why do you **never** do this to a real meter? Refer to the internal circuit for a real ammeter to describe what happened. Click the multimeter *Settings* button for further insight. What is the most common way used in real multi-meters to protect against errors like this?

(!) Switch the meter back to voltage mode. Pull down and insert a 1 kΩ resistor in series with the battery. (Position the resistor directly over an existing wire, and release; the resistor will insert itself.) Vary the resistance; is the voltage on the multi-meter changing? (*Hint:* you may have to go to pretty high R values!) Try to find the point where the meter reads exactly $\frac{1}{2}$ of the nominal battery voltage.

[?] In EWB the batteries are always ideal sources; however, the meters are not. The point of $\frac{1}{2}$-voltage is where the internal resistance of the (non-ideal) meter is exactly equal to the external R. Explain your result and provide a calculation to support your observation.

1.4 Review of basic electronic components

While logic gates are all that is required to construct a digital circuit, you will find that to interact with your digital system some familiarity with the behaviour of basic electronic components such as resistors, diodes and transistors, is necessary.

- The resistor is a linear circuit element in that it obeys Ohm's Law: V=I*R, where a voltage V across a resistance R causes a current I to flow through the resistor. The resistor is typically used in series with another component as part of a voltage divider or to limit the circuit current.

- The ideal diode can be visualized as a switch that is turned off ($R \approx \infty$) until a positive voltage (forward bias) $V \geq V_{on}$ is applied across the terminals. The negative diode terminal (cathode) is represented by the flat line. When turned on, a voltage V_{on} will be present ascoss the diode and $R \approx 0$. The value of V_{on} is $\approx 0.6V$ for a Silicon diode, $\approx 0.3V$ for a Germanium diode and $\approx 1.5V$ for a GaAs red light-emitting diode (LED). Continuous diode currents much greater than 10-20 mA will cause LEDs to burn out.

- The bipolar transistor is a three terminal device. The terminal with the arrow is the emitter, the opposite terminal is the collector and the centre terminal is the base. This device is a current amplifier in that the emitter-collector current $i_{CE} = \beta * i_{BE}$, the base-emitter current times the transistor current gain β. The base-emitter junction behaves like a diode so that $i_{BE} \approx 0$ and $R_{CE} \approx \infty$ until $V_{BE} \geq V_{on}$. As V_{BE} increases, $R_{CE} \rightarrow 0$.

  In the digital domain, a transistor is typically used to switch on/off devices that require more current than a logic gate is capable of supplying, such as LEDs, lamps and relays.

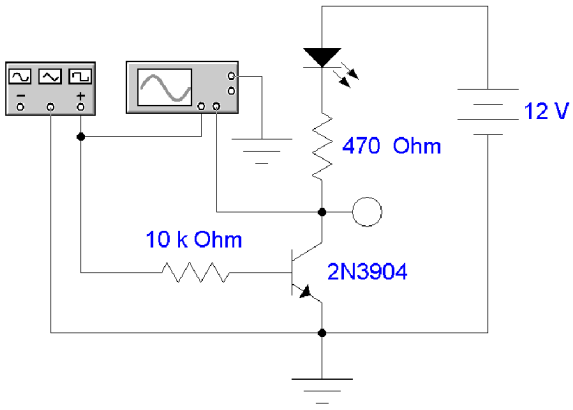


Figure 1.2: Transistor as a switch

You can incorporate these components in the circuit as shown. Use a scope (from the Instruments menu) to monitor the base and collector voltages. Drive it with a 10 V peak-to-peak square wave centered at zero volts with a frequency of less than 1 Hz. The round indicator probe turns on (solid color) at 2.5V. The LED is on when the arrows are filled in.

(!) Start the simulation and observe the behaviour of the circuit. Note when the LED is on and when it is off (circle below) in terms of the voltage at the base, V_b, and at the collector, V_c, of the transistor and the corresponding voltages across the components:

V_b	V_c	LED	probe
low		on / off	on / off
high		on / off	on / off

? Explain the operation of the circuit for the two input states. What is the purpose of the 470 Ω and 10 KΩ resistors? What logic operation does the transistor perform? What is the maximum current that could flow through the LED?

1.5 Digital indicators and controls

! Clear the screen of all components, or simply open a new circuit page. Find and drag down into your circuit page the word generator (from the Instruments submenu) and several one-bit logic probes (from the Indicators submenu). You can enter sequences of 4-digit hexadecimal numbers in the data window or load a stored pattern. The sequencer can be single-stepped or made to cycle through this list at a certain predetermined pace. The sequencer can also be put into an infinite loop, repeating the steps you programmed and returning back to the beginning.

For each number, the corresponding pattern of high/low voltage levels is produced on its 16 outputs. Edit the pattern list to contain a few lines with different bit patterns; try to include `FF`, `00`, and other interesting bit patterns in your sequence. Step through the sequence and observe how the pattern of indicators connected to the sequencer output lines follows.

! Logic probes are ideal sensors of the logical state of any wire in the circuit; a more realistic way is to use light-emitting diodes (LEDs). Drag several LEDs or an eight-LED bar graph indicator into your circuit, and connect the anodes of each LED diode to the outputs of the sequencer. Connect the cathode ends to ground through 220-Ω resistors. These serve as current-limiting resistors, without them the LEDs would "burn out" almost instantly. Enter a two-line sequence `AA, 55` which will create an illusion of lights running up your LED bar graph.

Modify your sequence to create a single running light: `00000001, 00000010, 00000100` *etc.*

! Replace the LED bar graph with a 7-segment LED display. Assemble the circuit shown.

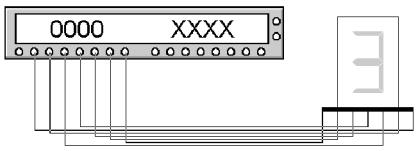


Figure 1.3: Controlling a 7-segment LED

The 7-segment LED display has seven inputs that each control the lighting up of a particular segment of the display. Thus for each pattern of bits produced by the pattern generator, there appears a pattern of lit LED segments. The labeling convention and pin assignment for each segment can be found in the help menu for the device.

! Map each segment to an output of the word generator, then generate a table of bit patterns that correspond to the 7-segment representations of the decimal digits 0-9. Program the list of patterns into the generator's table to produce a sequence of seven-segment digits from 0 to 9. Set the pattern generator into a loop mode over the list you entered, and verify that the display appears to "count" from 0 to 9.

! For an extra challenge, expand the list to 16 entries, one for each of the hexadecimal digits.

1.6 Bit combinations control a 7-segment LED display

The previous section demonstrated a "brute-force" approach; usually the job of controlling the individual segments of a 7-segment display is given to an integrated circuit called BCD-to-7-segment encoder.

(!) From the DEC menu in Flip/Flop bin, place a "Generic BCD-to-seven-segment decoder" into your circuit, between the sequencer and the 7-segment LED display. The conventional order is that on the BCD side the bit A is the least significant bit, and that the seven-segment display inputs are A-to-G left-to-right. Make sure that the LT (lamp test), BI (blanking input) and RBI (ripple blanking input) are tied high, *i.e.* attached to a $V_{cc} \equiv 5$V. Modify the sequencer program to count up from 0 to 9, and then back down to 0, a total of 19 steps.

Lab Report

Submit a lab report consisting of the work undertaken during this lab. waveforms observed, formula derivations, a description of the theoretical behaviour of the circuit and comparison with your actual observations, and answers to the pertinent questions. The presentation of your results should be organized and complete, your diagrams titled and referenced, so that someone who is not familiar with the experiments would have no difficulty understanding what was done.

At the end of the lab report, include a brief Conclusions section that summarizes and compares the results from the simulated and hands-on portions of the lab and a discussion of any problems encountered and insights gained.

Note: The lab report for this experiment is due before the beginning of the next scheduled lab session, one week later. Late labs receive a zero grade. To satisfy the lab component for the course, all lab reports must receive at least a passing grade. Be sure to have read and are familiar with the preface section of this lab manual.

Note: You need to prepare for the next experiment by designing several logic operations using NAND gates. Be sure to have these circuits ready to use.

NAME (PRINT) | STUDENT ID | LAB DATE | GRADE

Experiment 2

Logic Gates

The principles of digital logic govern the operation of all modern computers. The objective of this experiment is to become familiar with basic logic gates and their appropriate logic truth tables. These logic gates are then used to construct various simple logic circuits such as an adder, a latch, and a binary counter.

2.1 Circuit assembly techniques

You will be using a breadboard to assemble and test your hands-on circuits. The breadboard provides a convenient and organized way of implementing circuits, making quick component changes and providing trouble-free circuit connections. On the board you have access to five pairs of binding posts BP1-BP5, red and black. Also, there are five BNC coaxial connectors. A cable supplies the board with $\pm 15, +5$ and 0 Volts DC. The black binding posts and the outer ring of the BNC connectors have a common connection to ground, or 0V. These signals are all available to the protoboard, the matrix that you will build your circuits on, as shown in Figure 2.1.

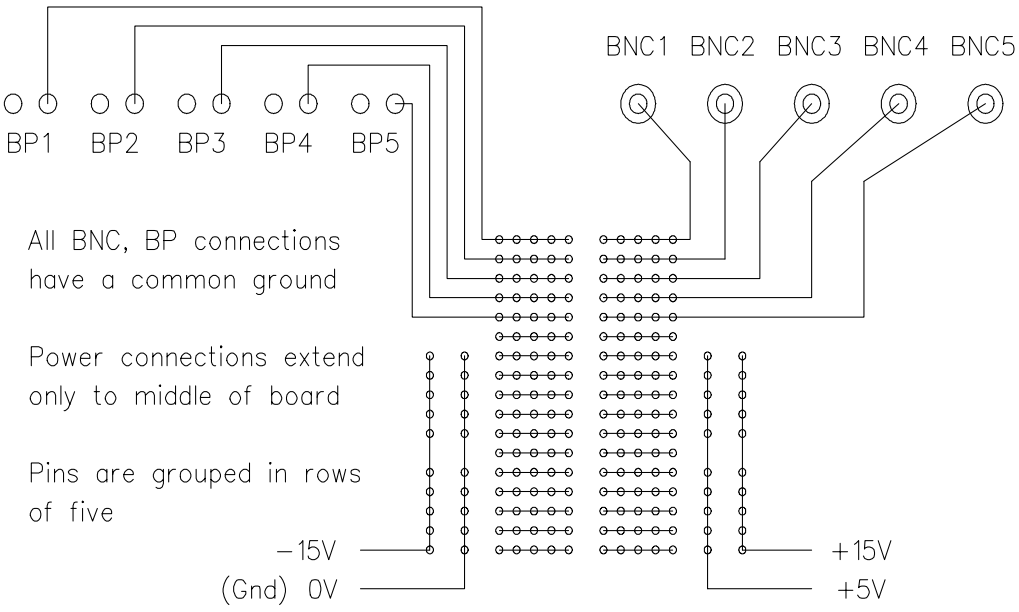


Figure 2.1: Electronics lab breadboard connection matrix

To make the experience of assembling a working circuit more enjoyable:

1. Be sure that the power is OFF;
2. verify by direct measurement the values of all the circuit components;
3. minimize the use of jumper wires by connecting components directly to one another;
4. verify that the jumper wires used are not broken (test for 0 Ω resistance);
5. assemble the circuit in a systematic and organized fashion;
6. check off each component as you add it to your circuit;
7. verify that your assembled circuit connections correspond to those of the schematic diagram.

Having taken the previous steps, turn ON the power and test the circuit for proper operation.

If the circuit does not behave as expected, you will need to do some troubleshooting. Use the schematic diagram as a guide to determine the voltage levels that should be present at various points of the circuit, then use a voltmeter to measure these nodes.

Develop a systematic approach to assembly and verification of the circuit that you are building. As the circuits get more complicated, you will find it advantageous to construct the circuit in stages, verifying the proper operation of the circuit after each progressive step.

2.2 Combinations of NAND gates implement other operations

Additional components required

- two 7400 IC chips and one 7473 IC chip
- four LEDs and five 470 Ω current-limiting resistors

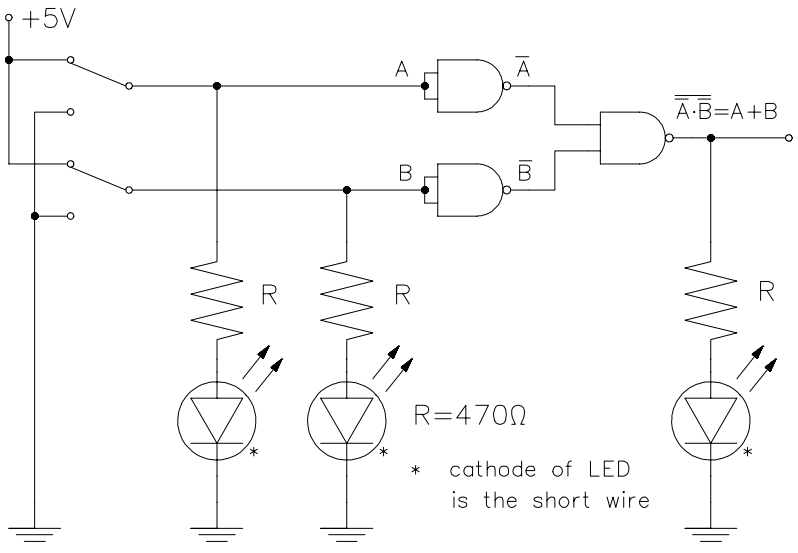


Figure 2.2: An OR gate made from NAND gates

NAND gates can be combined to form other logic gates. You must prepare for this lab by designing and drawing the schematic diagrams of the following circuits, made up entirely of NAND gates:

- an AND gate
- an exclusive OR (XOR) gate
- a half-adder
- a gated latch

In the example shown in Figure 2.2 three NAND gates form an OR gate. The LEDs on this diagram monitor the state of the inputs and the output. Each one is connected in series with a current-limiting resistor R.

Most integrated circuit (IC) chips contain a number of logic gates. In this experiment, the 7400 chip used contains four separate NAND gates. Make sure you start by making the power connections (ground and +5 V).

For each of the above circuits (an AND gate, an XOR gate, a half adder, a gated latch):

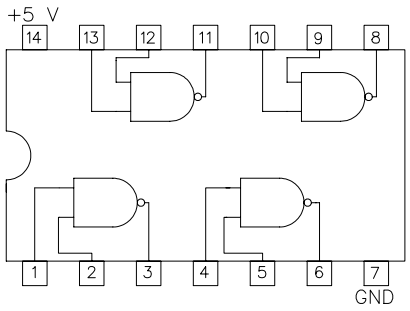


Figure 2.3: Pinout of a 7400

(!) Draw the circuit or create it using EWB. Label your gate connections with the corresponding chip gate pin numbers, as shown in Figure 2.4. This will ease the process of circuit assembly and troubleshooting.

(!) Assemble the circuit. Begin by installing the 7400 chip. Connect the power pins. Connect the gates and switches with jumper wires. Add the LEDs. Verify the correct orientation of each LED by connecting one end of the current-limiting resistor to the LED anode and the other end to +5 V. If the LED glows, it is inserted correctly. Re-connect the resistors as shown.

(!) Verify the circuit operation by testing all input combinations. Record the results as a truth table. For reference, data sheets that describe the logic states and expected operation of all these chips are available on the Web. You can verify your results using Electronics Workbench.

(!) Include in your report the Boolean explanation as done in the OR example, Figure. 2.2.

2.3 A binary counter

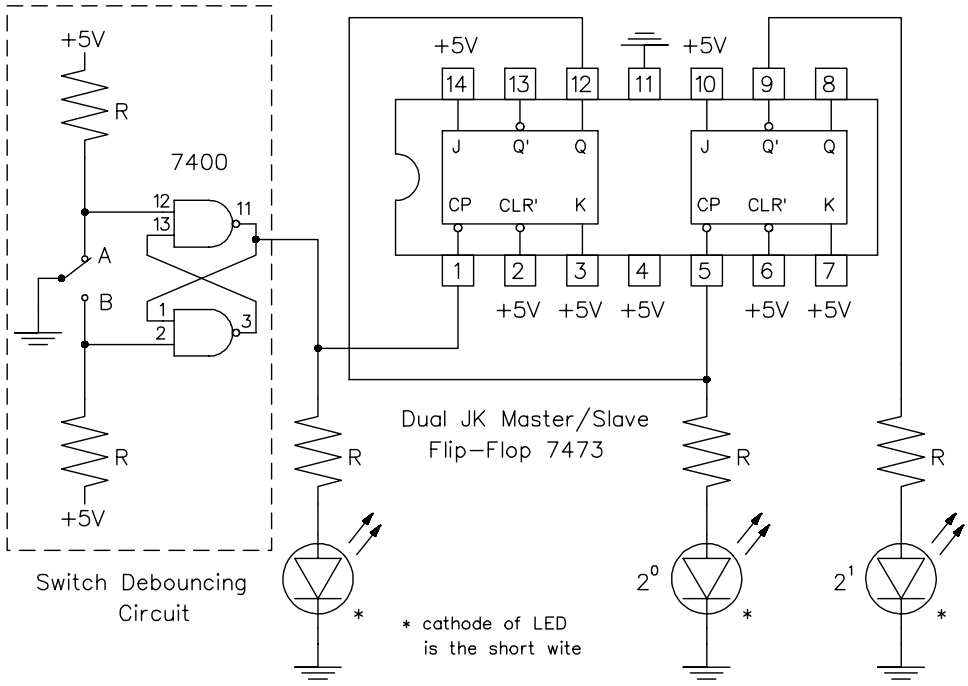


Figure 2.4: A binary counter

The binary counter, shown in Figure 2.4, is made using a 7473 dual JK flip-flop chip. This chip has several modes of operation, depending on the the signal levels at the J and K inputs. With

J=K=+5 V, the flip-flop performs a toggling function, so that the Q output changes logic state whenever a high to low transition occurs at the CP (clock pulse) input. Thus the Q output changes state at one half the rate of CP.

The switch, resistors and NAND gates make up a pulse generator that drives the counter input. When an electromechanical switch makes or breaks a contact, a series of pulses lasting some 1-5 ms are generated. This noise is known as *contact bounce* and is very undesirable in fast digital circuits. Since the gate delay and hence response time of flip-flops is around 10 ns, many pulses would be counted every time a switch is toggled, resulting in an erratic count. To *debounce* the switch, two NAND gates connected as an RS latch are used.

? Apply the above information along with the truth table for the RS latch to describe how the circuit debouncing is preformed. Again, the Web or your textbook may be a useful resource.

! Assemble the switch, resistors, gates and LED for the debounce circuit, following the assembly procedures previously outlined. Document how the output of the debounce circuit behaves as the switch is toggled. Is this the behaviour of an RS latch?

! Add the counter circuit components. Graph the input pulses at CP from the debounce circuit and the values of 2^0 and 2^1 below one another and using the same arbitrary time scale. Switch movement $B \longrightarrow A \longrightarrow B$ represents a single input pulse.

Advise the demonstrator when the counter is working and demonstrate its operation.

! Remove the debounce circuit from the CP input of the counter. Reconnect the switch as follows: A to +5V, B to 0V, and the common point to CP. Toggle the switch. Graph the output states of the counter as you did before. Describe your results.

! Remove the switch from the CP input. Connect CP to one of the BNC1 contacts on the breadboard. Connect a cable from the BNC1 connector to the SYNC output of the signal generator. This output is a 0-5V square wave that can be used to clock digital circuits. Set the function generator frequency to 1Hz and describe the circuit behaviour.

! Slowly increase the frequency and note when each of the three LEDs changes from a noticeable on/off flickering to a steady glow. Estimate from these results, the response time of your eyes.

Lab Report

Submit a lab report consisting of the work undertaken during this lab. Start the report with an overall statement of purpose of the experiments. Then for *each* exercise include a sketch or printout of the circuit and graphs of the waveforms observed, formula derivations, a description of the theoretical behaviour of the circuit and comparison with your actual observations, and answers to the pertinent questions. The presentation of your results should be organized and complete, your diagrams titled and referenced, so that someone who is not familiar with the experiments would have no difficulty understanding what was done.

At the end of the lab report, include a brief Conclusions section that summarizes and compares the results from the simulated and hands-on portions of the lab and a discussion of any problems encountered and insights gained.

NAME (PRINT) | STUDENT ID | LAB DATE | GRADE

Experiment 3

Combinatorial and sequential logic

In combinatorial digital logic circuits the state of the outputs is determined by the state of the inputs, and a Boolean logic equation or a truth table can describe the function of the circuit in a very compact way. In the sequential logic, the state of the inputs at some earlier time may also matter, and a full desription of the function of the circuit involves a timing diagram, with logic levels, transitions between levels, and their time relationship presented together. Many circuits combine both elements.

All digital circuits require some time for an output to respond to a change of state at the input. This is known as the propagation delay of the circuit. This delay needs to be taken into account in order to to properly analyze circuit behaviour and timing.

3.1 Divide-by-two flip-flop

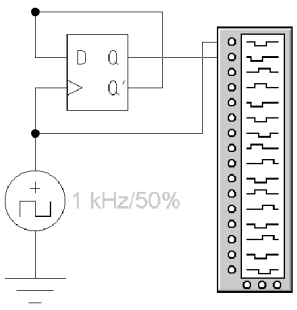


Figure 3.1: A :2 flip-flop

A D flip-flop propagates the state of its input D to the outputs Q and $\bar{Q}$ at the rising edge of the clock pulse, CP. The output is held after CP goes low. In the "feedback" mode, where $\bar{Q}$ is connected back to D as shown, it will take two clock cycles to return to the initial state. Thus a single D flip-flop performs a simple divide-by-two.

(!) Assemble the circuit shown. Use the logic analyzer to monitor the timing of the divide-by-two operation. The logic analyzer samples the logic states of the input lines at a rate determined by an internal or external clock. Check by double-clicking on the analyzer icon that this internal clock is selected and set to a frequency of 5-10 times that of the input signals, otherwise you may not see any meaningful analyzer output.

(!) Add the extra connection between the $\bar{Q}$ and the logic analyzer to monitor the second output as well. Make a record of the analyzer timing diagram.

(!) Increase CP and the analyzer clock to observe and determine the *propagation delay* from CP to the Q outputs. Record the analyzer output.

[?] Consider the relationship between the various signals in the two timimg diagrams. Why does the timing in the two diagrams seem to differ? Based on your observations, write down the

truth table for a D flip-flop. Compare with that of a real D flip-flop device, such as a 4013, by reviewing the device data sheet.

3.2 A binary counter with D flip-flops

Several D flip-flops connected in a chain perform as a binary ripple counter, each stage undergoing a change of state half as frequently as the previous one. In this circuit, slow the clock down to 1 Hz or so, and use just a set of LED indicators to read off the binary count.

(!) Assemble the circuit as shown and verify its operation. You may want to assemble one binary digit first, then select several circuit elements, and Copy and Paste several times to expand to more digits. Label the most- and the least-significant bits as MSB and LSB, respectively.

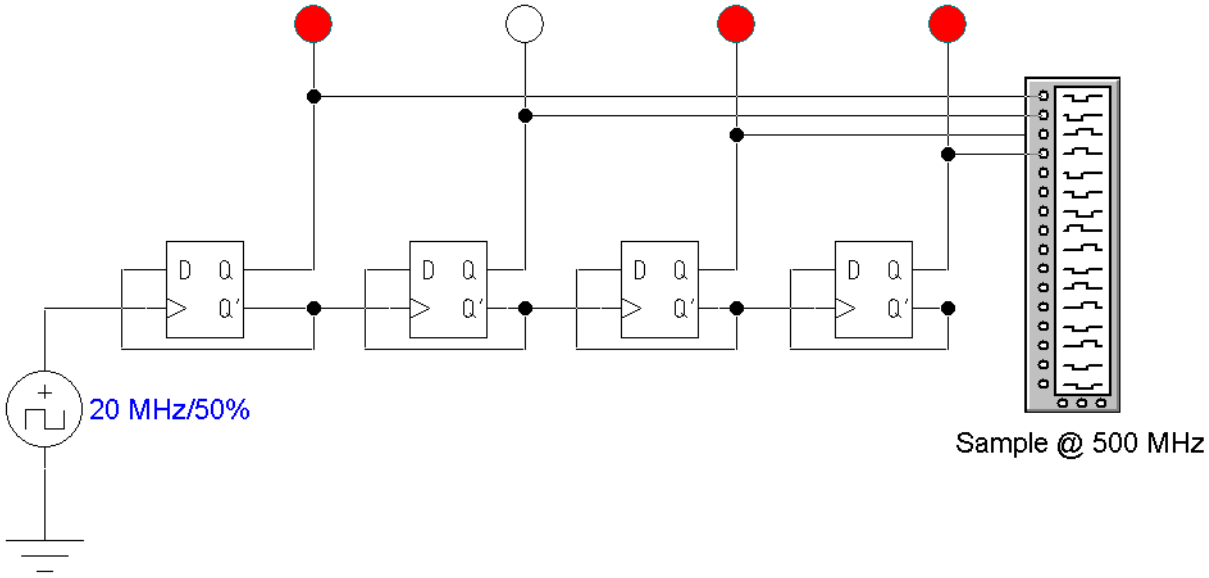


Figure 3.2: A binary counter

[?] The logic analyzer allows you to monitor the output waveforms and thus to verify the operation of the counter. Taken together, the four Q output lines represent a 4-bit binary value; how does this value change with every transition of the clock? How does the value at the Q' outputs change? Explain.

[?] Determine the propagation delay for each of the flip/flops and then the total propagation delay of the entire ripple counter of Fig.3.2. What is the maximum clock rate that should be applied to this circuit? How do the outputs behave as the circuit is *overclocked* beyond this maximum frequency? Explain this behaviour in terms of propagation delays.

[?] Predict the propagation delay of a 10-bit ripple counter. iHow does the propagation delay affect the size of a ripple counter? Is this a desirable characteristic in a counting circuit?

3.3 One-of-eight decoder

A key device needed in multiplexing/demultiplexing applications is a binary to one-of-eight decoder, similar to the one shown in Figure 3.3.

(!) Assemble the decoder as shown, drive it with the binary counter from the previous section, and verify that only one of the lights at a time, in sequence, are turned on by the binary counter.

[?] Use the analyzer to record the complete timing diagram for the 3-bit binary to one-of-eight decoder, including the clock pulse waveform.

[?] Given that all digital components in EWB exhibit by default a 10 ns propagation delay, what is the propagation delay from the clock to each of the decoder outputs? Verify this by observing the analyzer output. How could you decrease the propagation delay of the counter/decoder circuit in Figure 3.3?

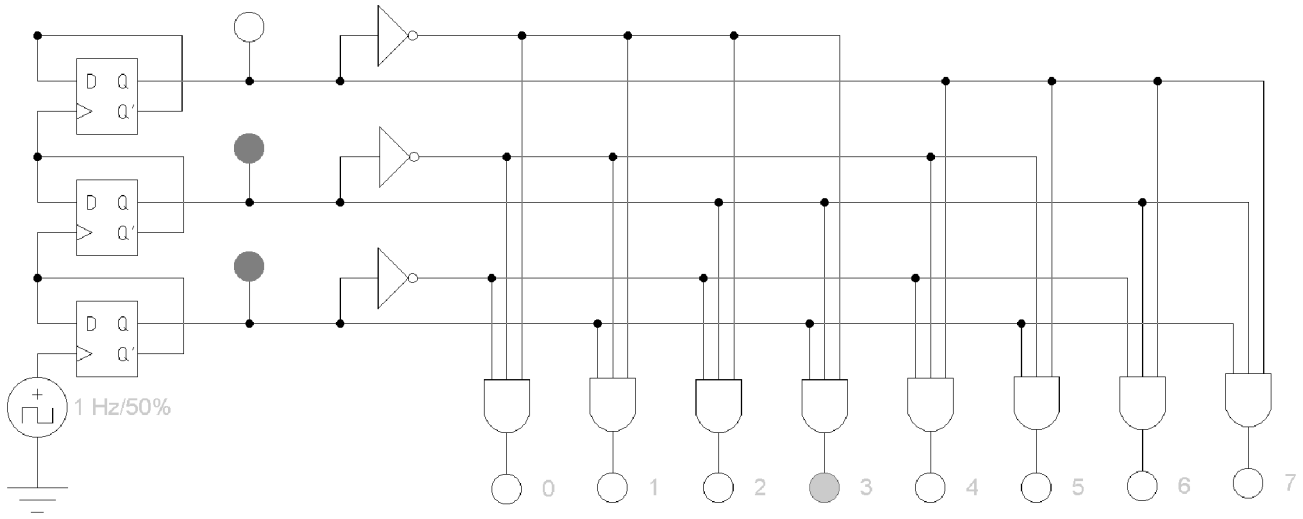


Figure 3.3: Binary to one-of-eight decoder

3.4 A design challenge

The challenge is to design an extension of the decoder circuit that lights a green indicator if and only if a special sequence (a password) of two 3-bit numbers is presented at the input, and a red indicator for any other combination of two input numbers.

(!) Sketch a flowchart of the logical steps that are required to solve the problem. From this, develop a timing diagram of the different signals that your circuit needs to generate.

(!) Replace the binary counter with the pattern generator. Verify that you know the pattern of 3-bit inputs that turns on any desired light. Note that the *Data Ready* output of the pattern generator makes a low-to-high transition when the the next data word is latched at the outputs.

? Monitor the circuit with the logic analyzer. How does the timing of your circuit compare with that of your timing diagram?

? How would you expand your circuit to decode a series of three 3-bit numbers?

NAME (PRINT)	STUDENT ID	LAB DATE	GRADE

Experiment 4

Oscillators and clock circuits

One does not always have the convenience of a frequency generator. From simple oscillator circuits that provide buzzing tones, to security system's delay circuits, to the crystal-based clocks that provide the heartbeat of every modern computer, a variety of methods exists to generate oscillating signals. In this lab we shall explore some of them.

4.1 555 timer IC as a square wave generator

Additional components required

- one 555 timer IC chip
- miscellaneous capacitors and resistors

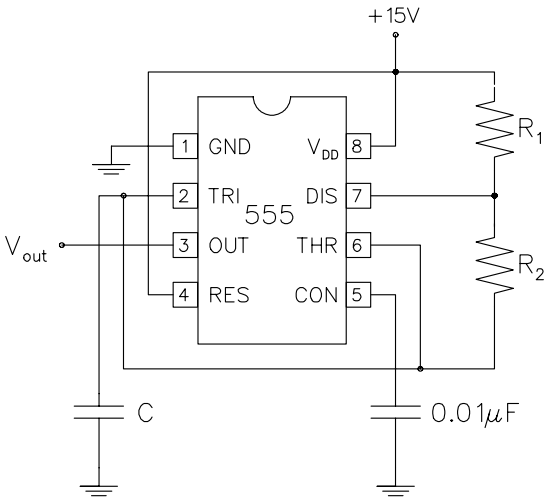


Figure 4.1: 555 Oscillator

The data sheet for this part describes the various modes of operation, connection diagrams and resulting waveforms. This timer has a monostable, or one-shot, mode where a high to low transition on the trigger input initiates a single pulse of a specific duration at the output.

The circuit can also be configured as a self-triggering (astable) pulse generator (Figure 4.1). You should pay particular attention to the charging and discharging equations for the astable configuration of the 555 timer, as they determine the duty cycle of the output pulse. The duty cycle of a digital (on/off) waveform refers to the proportion of the time that the signal is on relative to the overall period of the signal.

(!) Construct the 555 oscillator circuit shown in Figure 4.1. Refer to the internal schematic diagram shown in Figure 4.2 to understand its operation.

(!) Choose R_1, R_2, and C for an output of 1 kHz with as close to 50% duty cycle as you can. Verify that the circuit works and note the values you have chosen. Use the digital oscilloscope to monitor the output signal as well as the voltage at the capacitor and record these waveforms.

- [?] What is the range in the duty cycle of the 555 output? Consider the changes/additions that you might make to the circuit to produce output pulses of a relatively small duration, for example, a 10% duty cycle.

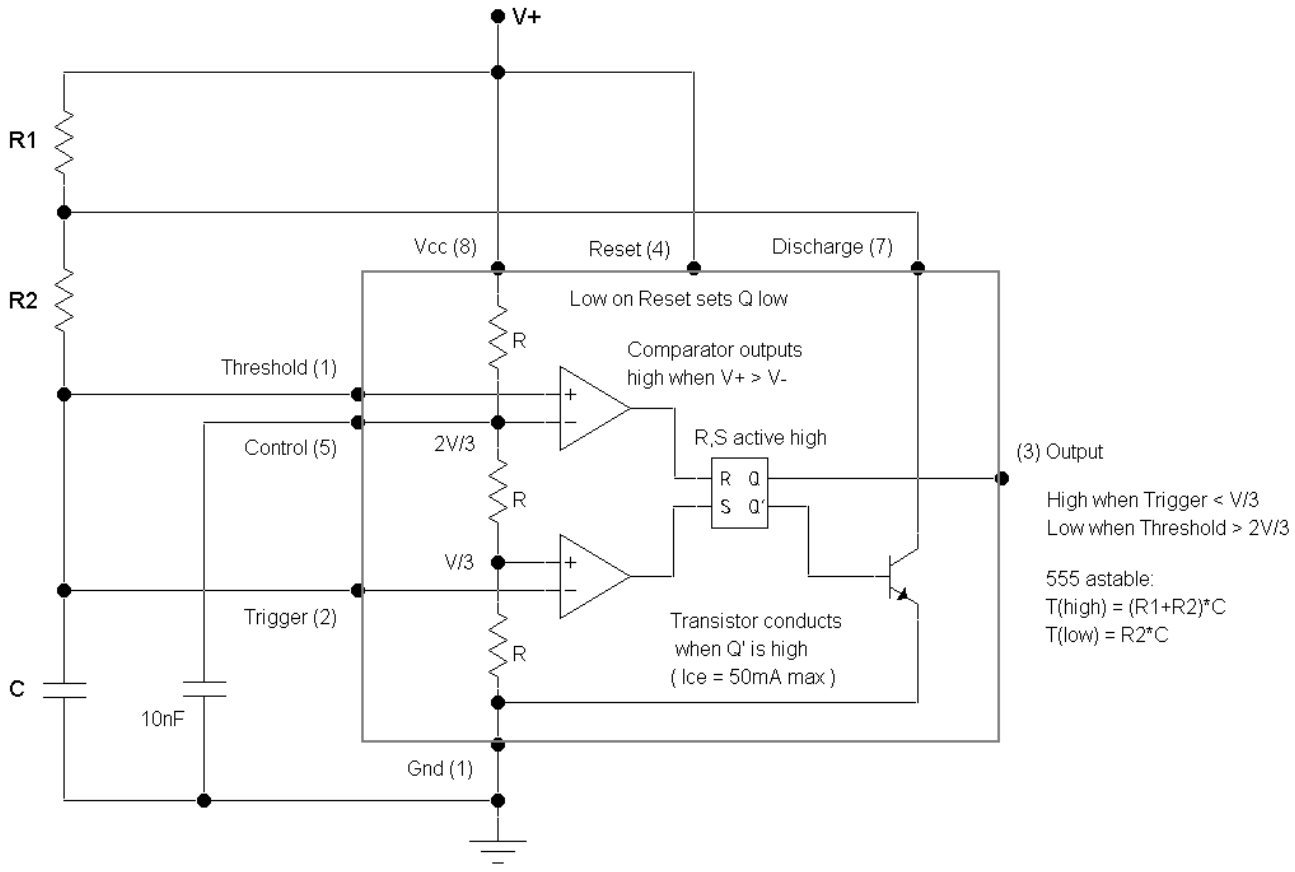


Figure 4.2: 555 bistable oscillator showing internal components

(!) Explore what happens if you disconnect the control voltage (CV) input pin 5 and instead connect it to a wiper of a 10-kΩ potentiometer that is connected between +15V and ground. This circuit is called a voltage-controlled oscillator.

[?] Comment on the variation in duty cycle as you vary the control voltage. How does the off time vary with changes in CV? Explain your observation using the fact that the oscillation is based on the exponential discharging of a capacitor.

Review: operational amplifiers

The two triangle shaped devices internal to the 555 chip are operational amplifiers. The output of an Op-Amp is given by $V_o = A * (V_+ - V_-)$ where V_+ and V_- are the voltages present at the two inputs and A is the open-loop voltage gain of the Op-Amp. This gain is typically very large, i.e. $A \approx 10^5$, so that a tiny difference in the input voltages will cause V_o to swing between the limits set by the power supply voltages of th Op-Amp.

In the 555 timer chip, the Op-Amps are used in the open-loop configuration (no feedback) as voltage comparators. The output of a comparator is high when $V_+ > V_-$ and low when $V_+ < V_-$. When $V_+ \approx V_-$, circuit noise will cause the output to randomly swing between low and high states.

4.2 Crystal oscillator and ripple counter

Additional components required

- one 4.096-MHz crystal
- one 4060 ripple counter IC chip
- 1-kΩ and 1-MΩ resistors
- 10-pf and 39-pF capacitors

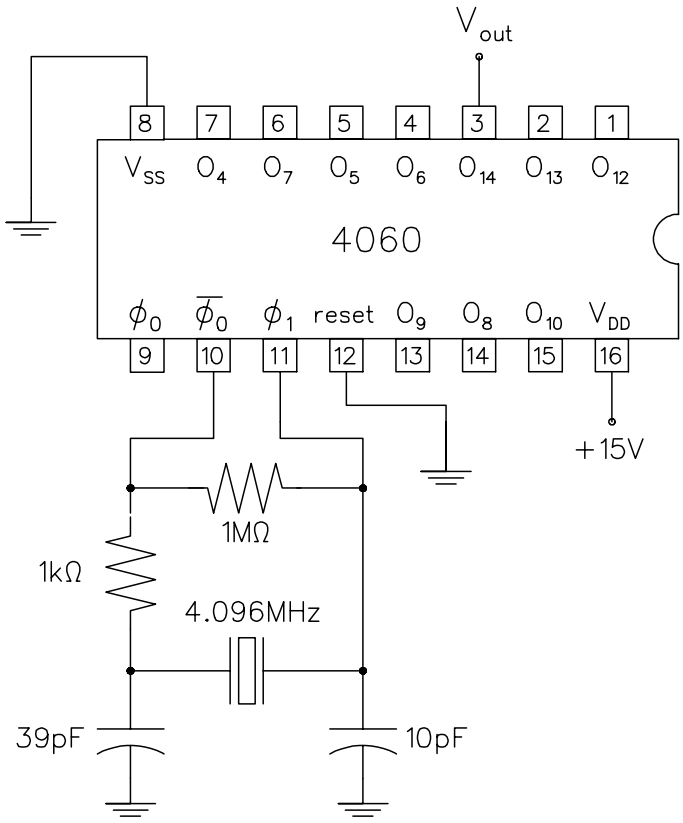


Figure 4.3: Crystal-based digital oscillator

Crystal oscillators are available in a variety of frequencies, which are determined by manufacturing them to exacting size and shape specifications. Often one encounters unusual values, such as the 4.096-MHz crystal provided for this lab. The reason for choosing such a strange value is to be able to generate oscillations of much lower frequencies by using multistage divide-by-two circuits. It just so happens that $4096 = 2^{12}$, and all we need is to build 12 divide-by-two counters in a row to get to 1 kHz, for example.

A convenient device is a 14-stage ripple carry binary counter, 4060. Depending on which output one chooses, one can obtain from a divide-by-2^9 to a divide-by-2^{14}. Thus with a single IC we can generate frequencies as low as 250 Hz, a convenient frequency if one wishes to build a small timer circuit with a time resolution of about ±0.002s.

Note how the suggested component values are selected to ensure a reliable 50%-duty-cycle oscillation, by keeping the crystal at the midpoint of the voltage divider formed by the 1 kΩ resistor and the 39 pF capacitor, *i.e.* how for $\omega = 2\pi \times 4.096\,\text{MHz}$

$$|Z_{R=1\,k\Omega}| \approx |Z_{C=39\,pF}|\,.$$

Also note that since the oscillator components are not directly connected to the chip power supply, the oscillator frequency is not affected by variations in the power supply voltage.

(!) Download a copy of the 4060 data sheet and review the internal schematic. Implement the circuit shown in Fig. 4.3. Check other outputs as well. Which output stage provides the 1-kHz signal?

(!) Determine with the aid of a two-channel oscilloscope the propagation delay t_{pd} for one of the flip/flop stages of this ripple counter. Measure also the delay of several (N) stages. Does this delay represent an integer multiple N of the single stage delay t_{pd}? Should it? How does your result for t_{pd} compare with that stated in the data sheet? Explain any noted discrepancy.

(!) Use HP function generator to provide a 250-Hz trigger signal to the oscilloscope and monitor the output of your crystal oscillator. HP signal generators contain high-precision internal frequency standards with very low phase drift. Carefully adjust the function generator's frequency until your oscillator's signal appears completely stable, then wait a few minutes and observe if the frequency has drifted. See if cooling your circuit with a gentle air flow causes changes.

[?] Why would it be a bad idea to use a 555 circuit as the frequency source for your timer circuit?

NAME (PRINT)	STUDENT ID	LAB DATE	GRADE

Experiment 5

Building a counter

This experiment explicitly involves several stages of a single project, from a conceptual design, to computer-based simulation, to physical implementation and debugging. The project: a decimal counter. Note that some of the necessary data is not provided on the diagrams of this experiment; it is essential that you consult the appropriate datasheets directly.

5.1 Building a device, stage by stage

Counting physical events is a common requirement that illustrates well many of the essential elements of a successful real-life application of electronic circuits. It involves:

the physical stage: the closing of a pair of mechanical contacts, a change in the illumunation of a photosensor, a voltage drop that activates a solid-state relay or a transistor. Here, the issues of imperfections, of efficient signal transduction (from the physical into the electronic realm), and of signal conditioning and noise resilience play an important role;

the signal processing stage, where the now electronic (logical) signal is processed, tallied up, analyzed or compared with a desired reference condition;

the display and feedback stage, where the results are either reported to the user, recorded for later analysis, or used to provide signals to actuators and control devices of a physical system. At this stage the feedback loop back to the physical realm is closed.

Pressing a push-button switch, counting these contact closures, and displaying the resulting tally on a seven-segment display are the three stages of this lab experiment, but this general three-stage structure is shared by virtually all electronic devices involved in the interactions with the physical world.

? Sketch a block diagram, representing the way an electronic system would gather the information about the physical world, process it electronically, and implement the resulting control decision. Briefly describe an example, different from this experiment, that you diagram may represent.

? For your example experiment, indicate the approximate bandwidth requirements, or the amount of information per unit time, that you expect to be communicated between the various blocks on your diagram. Where do you expect the bottlenecks to occur?

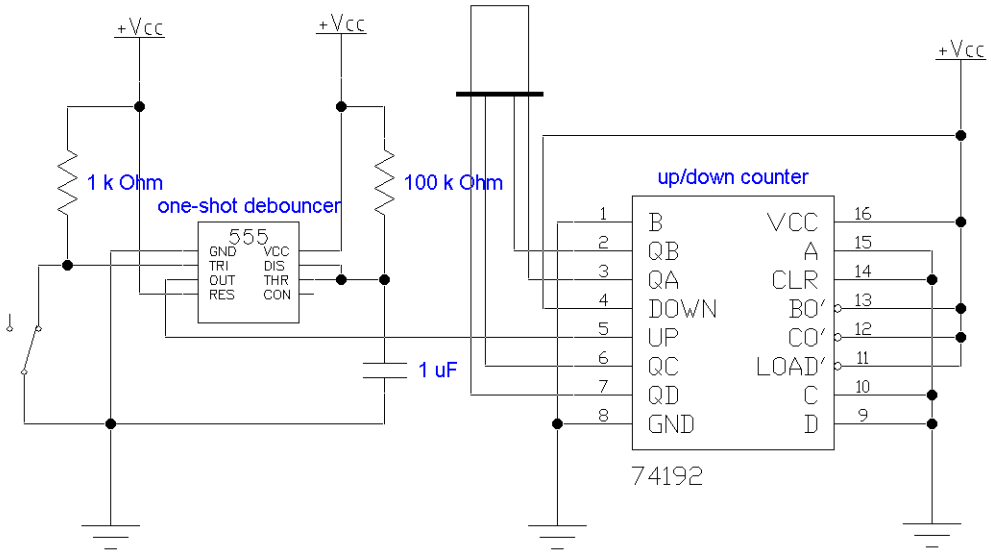


Figure 5.1: EWB circuit diagram

5.2 A virtual prototype

A 74192 IC is a convenient decimal counter, capable of reset to zero, preloading with a specified value, and of counting both up and down, as well as generating carry and borrow signals that allow one to construct multidigit counters. We will explore only a very limited subset of its capabilities.

(!) Use *Electronics Workbench* to simulate the circuit shown in Fig. 5.1. Initially, connect a 74192 and a four-input BCD-decoded seven-segment display, and connect a switch that will alternately toggle between a V_{cc} and the ground directly to the "Up" pin on the IC. A virtual switch toggles cleanly and instantly and will require no de-bouncing.

(!) Verify the operation of the counter. You may want to add additional elements such as logic probes on the outputs of the 74192, to make this task easier. See if using the switch to toggle the "Down" input of the 74192, while holding its "Up" input high produces the expected down-counting.

[?] Download from the web, or look up in a data book available in the lab, the datasheet of a 74192. How is the counting by 10 realized inside the IC out of four flip-flops connected in series? What is this technique called?

(!) Now complete the full circuit of Fig. 5.1 by connecting the 555-based debouncer. Choose external *RC* values so that the width of a pulse generated by the monostable is longer than about 50ms, a typical settling time of a real micromechanical switch. Use the virtual oscilloscope to verify that your calculations are correct and the pulse width is as expected.

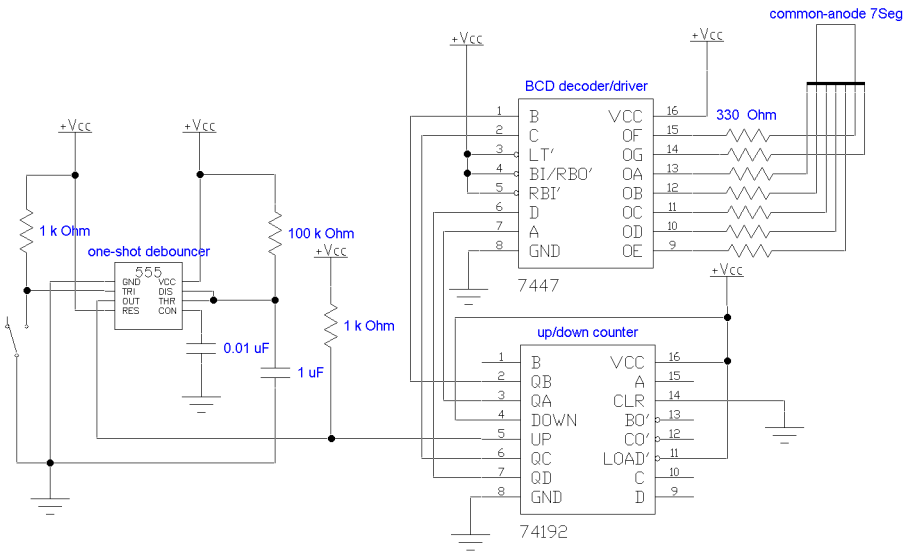


Figure 5.2: A real circuit involves additional details

5.3 Circuit realization

Additional components required

- one 555 timer IC
- one 74192 decimal counter IC
- one 7447 BCD-to-seven-segment decoder/driver
- one seven-segment LED, common anode
- miscellaneous capacitors and resistors, momentary switches

You are now ready to implement the real circuit. In addition to the de-bouncing of the real mechanical switches, the magic "BCD-encoded seven-segment display" has to be made out of separate components: a 7447 encoder/driver IC, current-limiting resistors, and a common-anode seven-segment LED display, as illustrated in Fig. 5.2.[1]

(!) Start building the counter circuit, starting with the final stage, the LED display. Make sure you have selected a common-anode LED display, and determine its pinout. The common-anode requirement is essential, as the 7447 decoder uses negative logic: it sets those segment lines it wants to activate into the low state. Do not forget to connect the common anode pin to the V_{cc} (not shown on the diagram).

[?] The suggested resistor values may not be optimal. Consult the 7447 datasheet, and that of a typical LED display device, to determine the optimal current that would make the LED display as bright as possible yet will not overload the 7447 driver, either by exceeding the current-carrying capabilities of a single driver line, or by exceeding the total power dissipation limits for the worst-case scenario of all seven segments being lit at once (the digit "8").

(!) Add the 74192 counter. Use a TTL square wave from the function generator set to a frequency of 0.5–5 Hz as the counting input.

[1]Note that our version of *Electronics Workbench* does not have a common-anode seven-segment display, only a common-cathode one, so if you were to simulate the circuit in Fig. 5.2, the display would show patterns complementary to those you would expect for the decimal digits. In the lab, the real common-anode displays *are* available, so the circuit should work as expected.

[?] Which is the active edge of the counting input? Sketch a timing diagram for a sequence of several cycles of the counter, starting with the state of the outputs that represents the decimal "9", and including all relevant input and output lines.

(!) Complete the circuit by adding a push-button switch, debounced with the help of a 555 monostable. Verify a reliable counter operation. Choose the RC values so that the fastest sequence of pushes of the switch that you can produce does not miss any and does not have any extra ones.

[?] Vary the RC until the time constant is too short to provide effective debouncing and you just start seeing multiple counts for a single pressing of the push button. What is the minimum pulse width that would provide reliable debouncing?

(!) Time permitting, explore other modes of operation of the 74192. A second switch, debounced with another 555-based one-shot, can be connected to the "Down" input, providing two-button up/down counter. A combination of four bits could be preloaded into the counter by pulsing on a "Load" input (active low). This is a decimal counter, so normally it shows only values from 0 to 9; what would happen if you pre-loaded a binary-coded four-bit value that is greater than 9 ?

NAME (PRINT)	STUDENT ID	LAB DATE	GRADE

Experiment 6

Four-bit multipliers

In this lab, we implement in hardware some algorithms that multiply two 4-bit numbers to yield an 8-bit result. The operating speed and hardware complexity of these circuits is explored.

In digital electronics there are usually several ways to express a design in hardware. Generally, the various approaches to solving the problem involve a trade-off between the speed of operation (clock cycles or gate delays) and the hardware complexity (gate count) of the resulting circuit. To implement an elementary mathematical operation, namely the multiplication of two 4-bit binary numbers, several possibilities exist: a brute-force repeat addition with counting, a shift-and-add scheme similar to the decimal multiplication scheme we learned in the elementary school, *etc.* Some algorithms naturally lend themselves to a sequential implementation, using counters to sequence, or *loop*, the hardware a specific number of times, and an array of flip/flops known as an *accumulator*, to store intermediate results. Some other algorithms are naturally parallel, and their implementations can be in the form of a purely combinatorial circuit. Hybrid versions also exist.

? What is the range of binary values that multiplying two 4-bit numbers can yield? How many bits may be required to store/display the result of such an operation?

6.1 A summing multiplier

The most basic multiplication algorithm simply involves repeated addition of a multiplicand, with the number of times the addition is performed specified by a multiplier, *i.e.* $3 * 5 = 5 + 5 + 5$. A possible implementation of this algorithm in a 4-bit adder circuit is shown in Fig.6.1.

For proper operation, any sequential circuit requires:

- initialization to some known state; and
- a clock to sequence the logic.

In the circuit of Fig.6.1, a low on the reset line clears the accumulator so that the initial sum is zero. The rising edge of the clock latches the data into the accumulator, and an add/store operation is performed each clock cycle. The clock signal can originate from a switch that single-steps the circuit, or automatically from a square-wave oscillator.

To perform the addition of each bit, a full adder is used. The carry output of each adder ripples to the carry input of the adder representing the next-most-significant bit. The output of the adder is latched by a D-type flip/flop with clear. The word to be added (multiplicand) provides one input to the adder; the other input consists of the data currently stored in the accumulator.

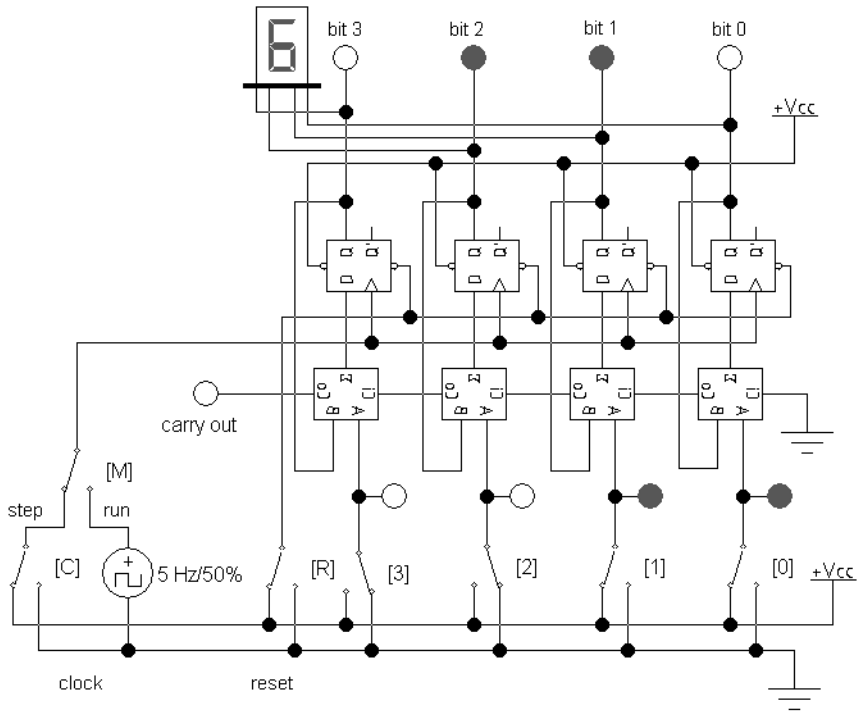


Figure 6.1: A 4-bit adder with accumulator

(!) Simulate this circuit. Try some input word values and step the circuit repeatedly by hand. What happens when the 4-bit capacity of the accumulator is exceeded?

[?] Given that the logic components used have a 10-ns propagation delay, what is the total propagation delay for the circuit? Which signal path determines the operating speed of this circuit? You might want to use the logic analyzer to monitor the timing relationship of the various signals.

(!) Increase the frequency of the clock to and beyond the operating frequency of the circuit so that a *signal race* condition occurs. What happens? Does the noted frequency agree with your estimate from the propagation delay?

The circuit will continue to add the multiplicand to itself as long as the clock continues to oscillate. In this 'manual' mode, you are responsible for counting how many clock cycles to provide to implement a particular multiplication operation. In addition, the limited number of bits quickly produces an overflow condition. We need to address both of these limitations.

(!) Extend the word size of the accumulator to double precision (8 bits). You can do this quickly by copying and pasting parts of the current circuit and making the proper connections. Verify that you can now serially multiply two arbitrary 4-bit numbers by setting the multiplicand via the four switches and clocking the circuit a number of times given by the multiplier.

Replace the manual clocking operation with an automatic one, using D-type latches connected as a down counter, as shown in Fig.6.2. Note the additions that have been made to the 8-bit adder circuit. The D-latches have active low preset and clear inputs. Switches set the 4-bit multiplier value which is loaded into the counter on reset. The rising edges of the clock pulses decrement the counter to zero and the circuit is then locked so that no further changes take place until a reset pulse restarts the circuit and another multiplication is performed.

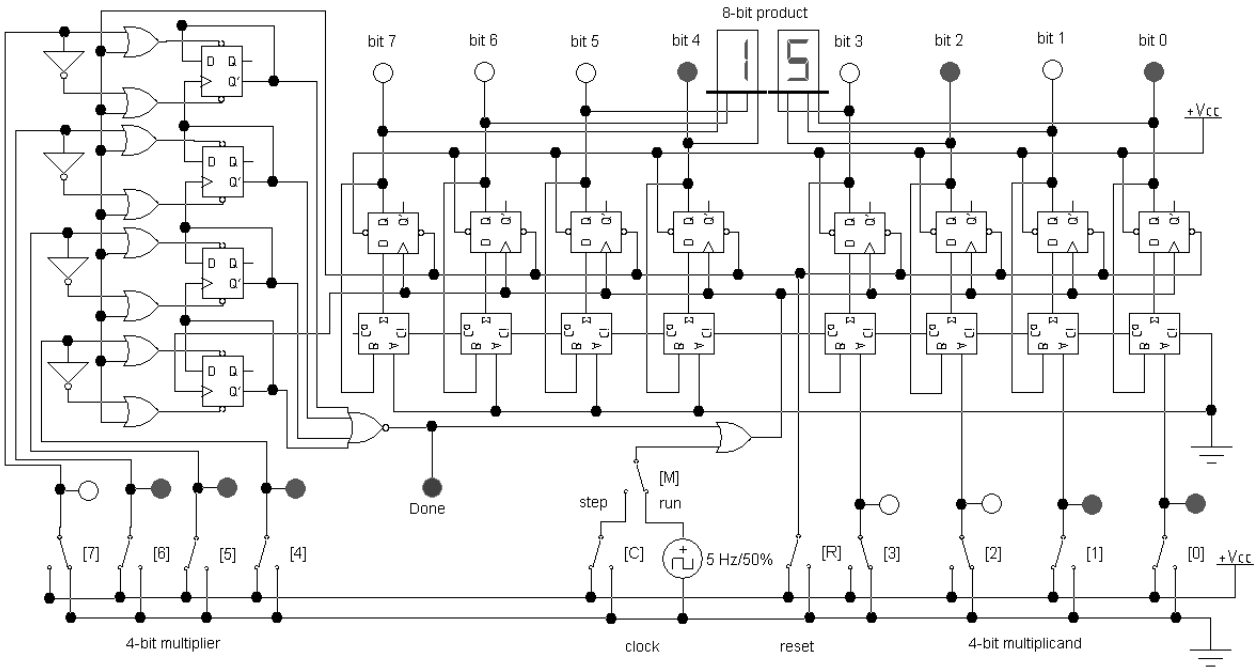


Figure 6.2: A 4x4 bit summing multiplier

(!) Modify your 8-bit adder circuit to match Fig.6.2 and verify its operation.

(!) Describe using truth tables and timing diagrams the operation of the logic used to load the counter with an initial value, and the logic used to lock the circuit so that a valid result can be read.

(!) *Optional:* Simplify this serial summing circuit. Hint: consider how the upper bits (bits 4–7) of the multiplicand enter the 8-bit adder circuit, as well as what happens to the overflow pattern of bits 0–3.

6.2 A shift/add multiplier

With some minor changes to our 8-bit adder circuit, a more efficient multiplication circuit can be realized. Binary multiplication can be performed in a manner analogous to the well-known procedure of decimal multiplication by hand, in which a series of shifts and additions — one for every digit of the multiplier — yield the product. In binary multiplication, this procedure is simplified by the fact that each binary digit has only two possible values, 0 and 1. To perform binary multiplication:

- initialize to zero both the accumulator and the loop counter;
- shift the contents of the accumulator, this is equivalent to a multiplication (or a division) by 2;
- for each bit of the multiplier in sequence: if the bit is a one, add the multiplicand to the accumulator; otherwise skip this step, *i.e.* add zero;
- repeat the above steps until the loop counter has counted the number of bits in the word.

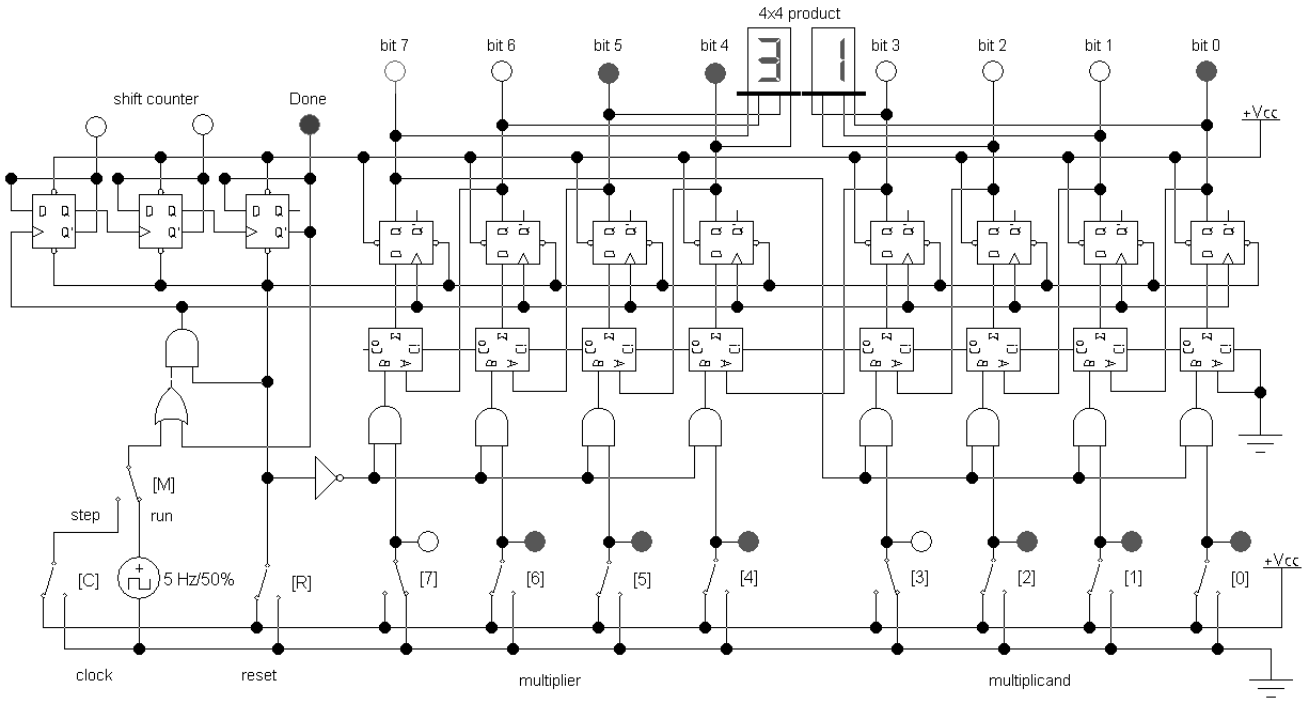


Figure 6.3: A 4x4 bit shift/add multiplier

The product is stored in the accumulator.

Fig. 6.3 shows an implementation of a 4x4 multiplier that uses a shift/add algorithm. The circuit is quite complex and we need to spend some time understanding its operation.

Let us begin by considering the operation of the circuit on reset. With a logic zero on the reset line, the accumulator and shift counter latches are cleared and the clock circuit is disabled. One input to the AND gates for bits 4–7 is set high so that the multiplier word can pass through the AND gates to the B inputs of the adders for bits 4–7, where it is added to the current value in the accumulator bits 4–7, which are zeros initially. Since bit 7 of the accumulator is zero, the AND gates for bits 0–3 are forced into a low state, setting the outputs of adder bits 0–3 to zero.

When the reset line makes a low-to-high transition, the multiplier word is latched into bits 4–7 of the accumulator, while bits 0–3 are loaded with zeros. AND gate outputs for bits 4–7 are set to zeros and the clock circuit is enabled.

Note that each bit in the accumulator is wired to the next-most-significant bit of the full adder array. Hence a left shift operation is being performed every clock cycle. The multiplicand is always added to the accumulator as a four-bit value, that is, the four most significant bits added are zero. This is exploited to reduce the total gate count: the four bits of the multiplier value can be initially loaded into the upper four bits of the accumulator without affecting the result as these bits will have shifted out of the accumulator.

On every low-to-high transition of the clock, the multiplicand (or zero, if the appropriate multiplier bit is zero) is added to the accumulator value and stored in the accumulator latches. The deciding condition is the state of bit 7 of the accumulator, which of course is reporting each of the bits of the multiplier as it gets shifted out of the upper four bits of the accumulator — which had been pre-loaded there by the reset. Note that the order in which the multiplier bits are processed is MSB-to-LSB, as the accumulator contents are shifted *left*. At the same time, the shift counter is incremented. In this case, use of an up counter is preferable since no preset logic is required for

it. A down counter would have to be loaded with the count value and use some logic to detect a zero output. When the shift counter reaches four, it sets the OR gate high, disabling further counts until a reset re-initializes the circuit.

(!) Simulate and verify the proper operation of the shift/add multiplier circuit. Consider some improvements to the circuit. For example, you may want to terminate the multiplication prematurely when one or both of the data words are zero.

[?] Since the multiplier value is set by switches, it is the accumulator value that is being left-shifted. Choose an arbitrary pair of 4-bit values for the multiplicand and the multiplier, and step through an entire multiplication cycle, recording the state of the accumulator after each step. Identify what happens to the multiplier bits, initially at bits 4–7, through the cycle.

[?] An alternate strategy would select the multiplier bit to test (is it =1 or =0) using a multiplexer driven by a down counter at its inputs. Sketch the relevant part of the circuit based on this idea. Compare this strategy to the one used in the circuit of Fig. 6.3.

[?] Compare the summing and the shift/add circuit implementations. How does the number of cycles required vary for the two circuits?

[?] Itemize the changes that you would need to make to the two circuits in order to perform an 8x8 multiplication resulting in a 16-bit product. How does the timing and gate count of the circuits vary with the size of the data word?

6.3 An array multiplier

Consider Figure 6.4, a design for a 4-bit array multiplier. Here, the four cycles of shift/add operations have been piggybacked into several stages that perform the series of operations all at once, combinatorially rather than sequentially.

(!) Simulate and verify the proper operation of the array multiplier circuit.

[?] Does this circuit offer a better performance than the shift/add multiplier?

6.4 Look-up table multiplier

Another possibility, especially when performing complicated arithmetic calculations such as evaluating the sine of an angle, is to use a lookup table. Here, a matrix of latches generically known as storage memory, is loaded with all the possible combinations of output values for a series of two input words. To implement a 4-bit multiplier, a memory array of 256 8-bit words is required. The multiplier sets bits 0–3 of the memory address while the multiplicand sets to bits 4–7. The resulting product on the output is simply the the data stored at the address selected by this combination of two 4-bit inputs.

[?] Evaluate the advantages and disadvantages of using this approach.

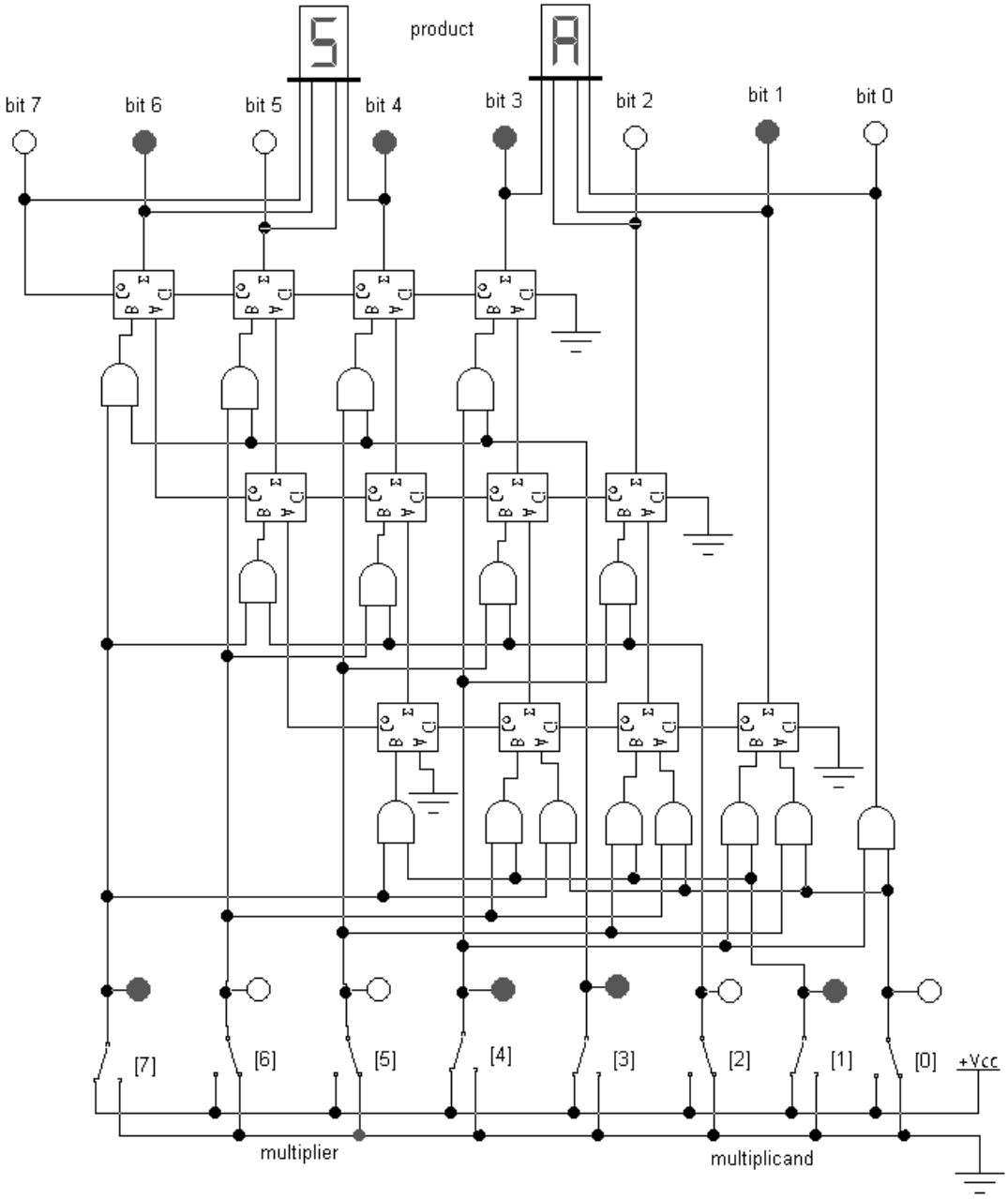


Figure 6.4: A 4x4 bit array multiplier

NAME (PRINT)	STUDENT ID	LAB DATE	GRADE

Experiment 7

PICLab project board

7.1 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 series 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.

7.2 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.

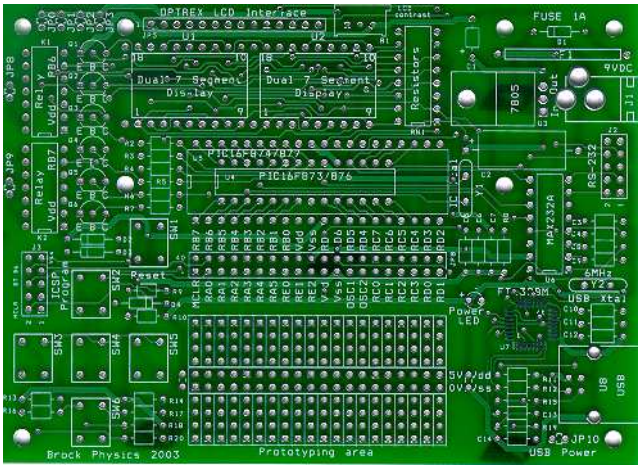


Figure 7.1: A PICLab printed circuit board, version 1.0, the component side

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 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 7.1: 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 7.1 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.

Table 7.2: 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

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. 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.

7.3 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 7.1. 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.7.2 shows you what your completed PICLab project board should look like.

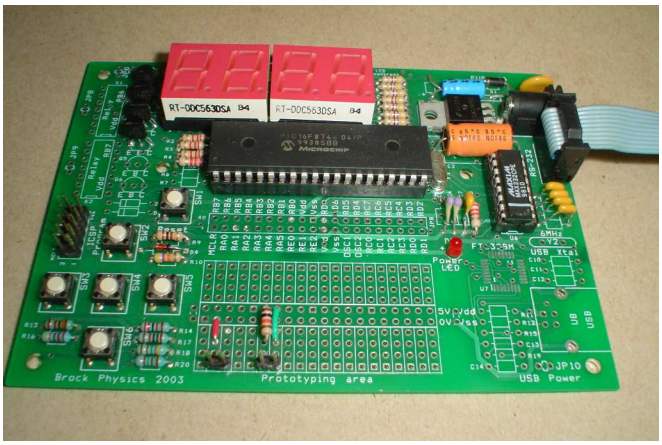


Figure 7.2: A completely assembled PICLab project board

7.4 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.
- 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.

NAME (PRINT)	STUDENT ID	LAB DATE	GRADE

Experiment 8

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.

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

Fig. 8.1 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.

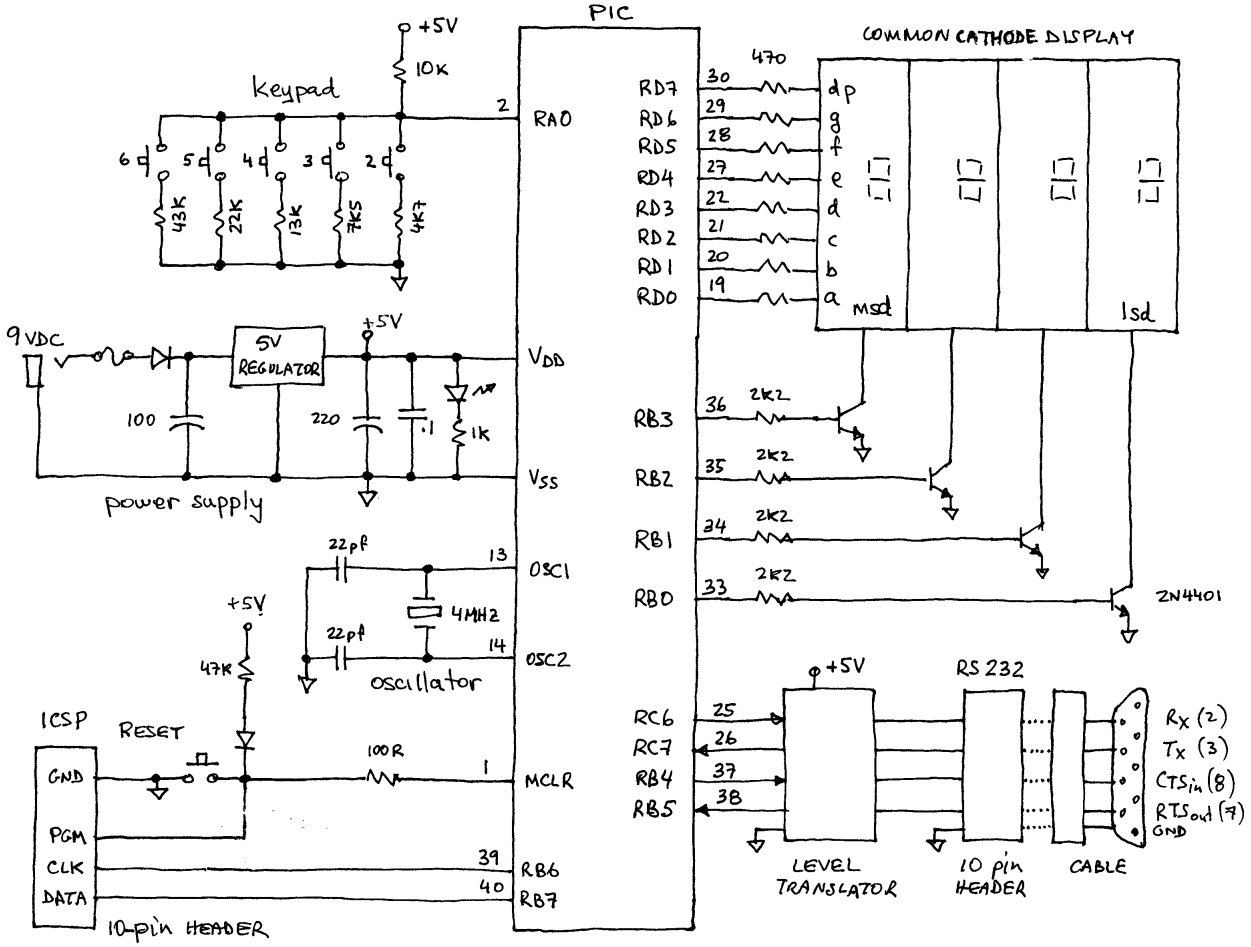


Figure 8.1: Block diagram of a PICLab board

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 &
```

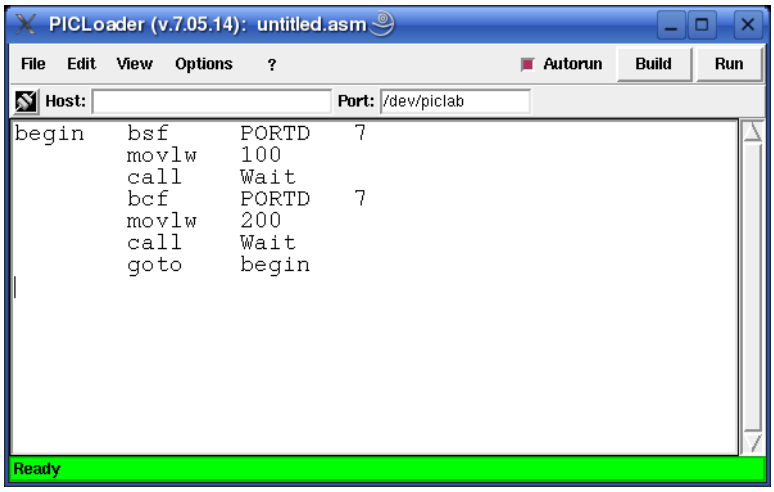


Figure 8.2: `picl` application window, connected to PICLab

Fig. 8.2 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.

8.1 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.

8.2 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. 8.1). 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.

8.3 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.

8.4 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
```

8.5 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
```

8.6 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.

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.

[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.

NAME (PRINT)	STUDENT ID	LAB DATE	GRADE

Experiment 9

PICLab data acquisition project

This PIC project is meant as a chance for you to put all your skills in digital/analog electronics, PIC programming/interfacing and practical problem solving to use and to build a useful device of your own. Your project will consist of the following steps:

1. research:
   the research cycle involves a preliminary exploration of possible project ideas where the hardware and software implementation options and perceived design difficulties are considered. Analyse the evolutionary steps involved in the design process that need to be successively fulfilled to make your project work; the choice of input sensors, the interpretation and processing of the input data, and the output of results or control signals that are required to give your design its planned functionality. Consider also the enhancements that, time permitting, could be made to your design to increase the usefulness or capabilities of your project;

2. proposal:
   the proposal consists of a written and oral presentation of the tentative PIC design project that you have selected. The proposal outlines what you are planning on doing, how you are planning to do it, and the difficulties that you expect to overcome. It should be the end result of some significant amount of time spent thinking about the practical issues involved in the implementation of the project, the analysis of several possible alternative hardware/software solutions to the problem, specifically in terms of the merits of the different approaches;

3. development:
   this involves the experimental process of attempting and achieving the milestones set out in the design proposal, and will likely involve a mixture of the assembly of mechanical and electronic components as well as software programming. A part of this process involves the resolution of problems, expected and unexpected, that arise during the development of any prototype system. A detailed, dated documentation of the development process must be preserved. This is a record of the steps undertaken to accomplish the end result, as well as the design choices that were attempted but proved to be unsuccessful;

4. presentation:
   as a final step, you will make an oral presentation consisting of a demonstration of the working project and a description of its operation. The presentation concludes with an assessment of your final result in terms of the goals set out in the project proposal. You should have available for submission a printed copy of your design records as well as a user manual that describes the device opeartion and includes hardware schematics and software code.

Below are several simple ideas, you are welcome to suggest a similar project of your own for approval by the instructor.

9.1 A PID temperature controller

One of the simple ways of controlling semi-continuously the state of a system is through the adjustment of the ratio of on/off times. For example, if the temperature of a reservoir is below the target temperature, we turn the heater on, and if it exceeds the target temperature, we turn the heater off. A temperature sensor and a TTL relay acting as the power switch for the heater under the control of a microprocessor is enough to implement a simple temperature controller. However, a truly intelligent controller would implement the so-called PID (Proportional-Integral-Derivative) control algorithm, whereby the temperature difference with the target temperature (P), the time integral of temperature (I) and the time rate of change of temperature (D) are all monitored and the output is controlled by all three, with varying-weight contributions. The reasoning is simple: a purely Proportional control always overshoots the target and, depending on the thermal inertia of the heater, may end up always oscillating about the target temperature. Including the other contributions (I,D) can produce an approach to the target temperature that is free of overshooting, and maintain a stable target temperature. A further improvement is to automatically adjust the relative weights of the P,I, and D contributions, to maintain this stability even if the thermal inertial properties are changing, for example as the level of the fluid in the reservoir is changing.

9.2 An ultrasonic pinger

First developed for distance measurement in autofocus cameras, ultrasonic distance measurement is based on a simple time-of-flight measurement for an ultrasonic sound pulse. It can be used to measure distances with high precision ($\pm$2mm is typical), using a standard ultrasonic transceiver (both a transmitter of the pulse, and a receiver of the echo signal from a distant object) and a timer-counter that measures the time between the sending of the pulse and the receiving of the echo. With proper calibration (the speed of sound in the air varies with temperature and humidity), a transceiver interfaced to a microcontroller is an excellent distance measuring device. It can be used in a Mechanics lab, or for non-contact level measurements in a well, or as a parking guide device in tight spaces.

9.3 A chaotic dripping tap

The incessant drip-dripping sounds of a leaky tap drive us insane late at night, but a precise measurement reveals that the time interval between drips is almost never perfectly periodic. In fact, drop formation is influenced by a number of factors, including the size of the previous drop. As a result, the dripping tap is an example of a chaotic system, and a variety of dynamic measurements can be performed, demonstrating period-doubling, attractors, and other features typical of the dynamics of chaos. An infrared LED/detector pair interfaced to a microcontroller is sufficient to perform time-between-drips measurements that can be logged and analyzed.

9.4 A universal digital knob

There is a distinct advantage to being able to "turn a knob" to control something; there is a tactile feedback that tends to feel natural to the operator, and reduces the error rate. However, analog knobs, based on variable resistors or capacitors have limited range, and are mechanically unreliable; they are also quite expensive. On the other hand, an electronically-controlled counter that uses

up/down buttons is awkward to manipulate and provides no tactile feedback. One interesting compromise can be realized if a digital rotary encoder can be used to control a digital register value. In general, these devices generate a series of pulse that can be used to increase/decrease a counter value; in the best designs, some accounting of the speed of the turning is made, so that slowly turning the shaft produces individual pulses, while turning it more rapidly also increases the "step size", generating more pulses per unit rotation, a form of adaptive "ballistic" control. One of the most inexpensive ways to implement such a universal digital knob is by interfacing a stepping motor from an old floppy drive to a microcontroller.

9.5 An LCD bar-graph

A companion project to the previous one is a way to display the value of a register as a simulated bargraph, using the special block graphics characters available on an LCD display interfaced to a microcontroller. Some button action (up/down scrolling through a list of registers, or simply one-of-five single-button selection of registers) selects a particular variable/register; the "digital knob" changes its value, and the LCD displays the name/address of the variable and its value, with a simulated LCD bargraph providing a visual representation of the value. As a stand-alone project, an LCD bargraph could be used as an add-on feature to a straightforward digital voltmeter.

9.6 A joystick-controlled servomotor

Fly-by-wire controls physically separate the controller (a joystick) and the devices that are performing the action (servomotors). Instead of mechanical or hydraulic linkages, an electronic connection is made. In an analog joystick, the position controls two variable resistors (x– and y–directions) and in a digital one, it effectively presses one of four switches, one for each of x, y, $-x$, and $-y$ (or eight, if each direction has two positions, indicating half and full strength). This information is processed by a microcontroller and converted into an electronic signal that is then sent to a remote microcontroller that in turn uses it to control the state of an action device. Along the way, the command signal may be manipulated in some way, for example to ensure that an unstable or a dangerous configuration is not accidentally requested by a user, or that all transitions from one state to another are optimally smooth and do not exceed a certain rate of change (in an aircraft this might result in a dangerous-to-pilot g-force). A single microcontroller with a joystick at its input and a servomotor at its output is enough to create a simple fly-by-wire controller that demonstrates most of the essential features of such a system.

9.7 Decoding an infrared remote control

Infrared pulses can be used to carry signals wirelessly across short distances in line-of-sight configurations. These are used in a variety of devices, from TV remote controls, to shared office printers, to toys such as the Furby, capable of some form of communication with its own kind. To build or debug an IR-enabled communications device it is useful to have a decoder of what these devices are sending through the IR, into a sequence of readable ASCII codes. A more ambitious project would involve two-directional communications, both reading what the other IR transmitter is sending, and sending one's own signals out. This is easily implemented using a microcontroller; this is what is at the heart of all "universal programmable" remote controls that are widely available. With

a microcontroller-based decoder/encoder attached to a serial port, you can make your computer communicate with all such devices, and to replace all of your remote controls with a software one.

9.8 A PIC-based mouse controller

A typical electromechanical pointing device (a mouse) has a large rubberized ball rolling on the work surface and in turn rotating two orthogonal slotted wheels, one for the x- and one for the y-motion. Each slotted wheel interrupts two infrared beams, producing a pair of logical pulses, 90° out-of-phase. Each pair of pulses corresponds to a step of displacement, and their relative phase determines the $\pm$ direction of motion. Typically, these TTL signals are converted by a dedicated mouse controller IC into a stream of codes sent through a serial connection to the computer, reporting the displacement of the mouse. The same task could be performed in software running on a microcontroller. In this way, non-standard display modes (for example, direction and velocity display) could be programmed. If a sufficiently fine resolution can be obtained, the two-wheel interface could be adapted to display in real time the action of a chaotic two-pendulum magnetic toy, or to the monitoring/control of a model Foucault pendulum. For the latter, useful pointers are:

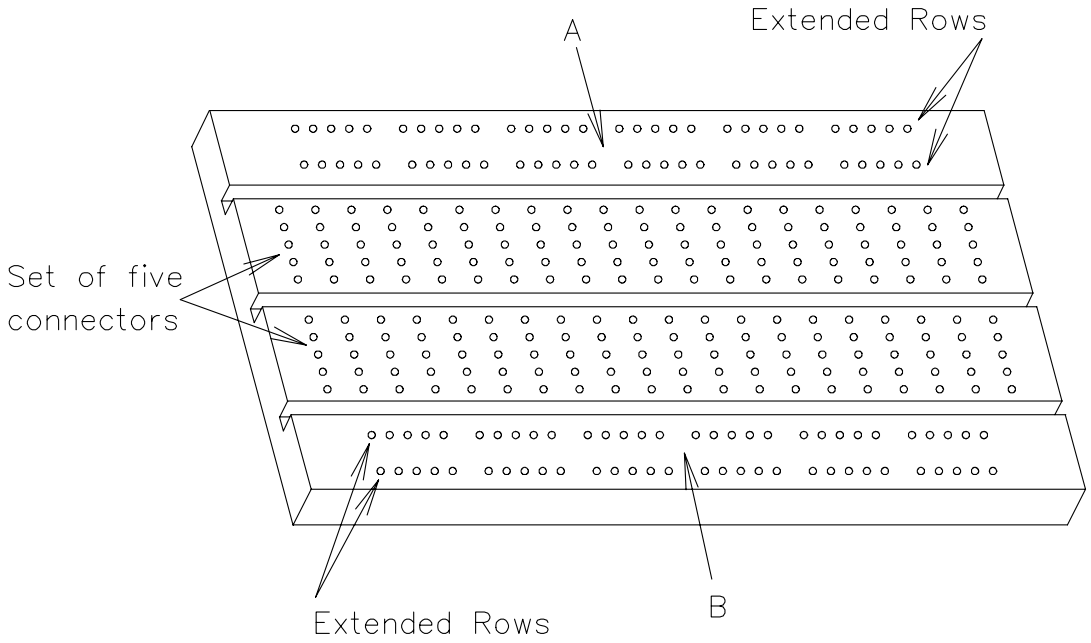
- Richard Crane's article in the References section on the PHYS 2P32 website
- `http://en.wikipedia.org/wiki/Foucault_pendulum`
- `http://www.iop.org/EJ/article/0031-9120/19/6/412/pev19i6p294.pdf`
- `http://www.sas.org/E-Bulletin/2002-04-26/handsOnPhys/body.html`

NAME (PRINT) | STUDENT ID | LAB DATE | GRADE

Appendix A

Breadboards

Breadboards permit quick solderless connections between the components of an electronic circuit. As indicated in the diagram, the holes of the breadboard are split into parallel sets of five (5). Within each hole is a metal clip to hold a wire and the clips in each set are connected together.



Two wires can be connected electrically by placing their ends into two holes belonging to the same set of 5 holes. The connector sets in the outside rows have been joined together to form four (4) extended rows. (Some breadboards do not have connections at points A and B resulting in eight "half" rows.) These outside rows are often used to supply power to the board. After an external power supply has been connected to one of these rows, power can be withdrawn to supply electronic circuits at any location along the board.

Each of the breadboards you will use is assembled on a plug-in unit that fits into one of the connectors on the common backplane. On this backplane you have access to five pairs of banana jacks (red and black), and five coaxial BNC connectors. All black banana jacks and the outside contacts on the BNC connectors are grounded and are thus electrically equivalent. The red banana plugs and the center conductors of the BNC connectors are all routed to each of the plug-in breadboard modules. In addition, ±15V, +5V, and 0V DC power connections are also available on each breadboard. Fig. A.1 shows the location of the common connections on the breadboards.

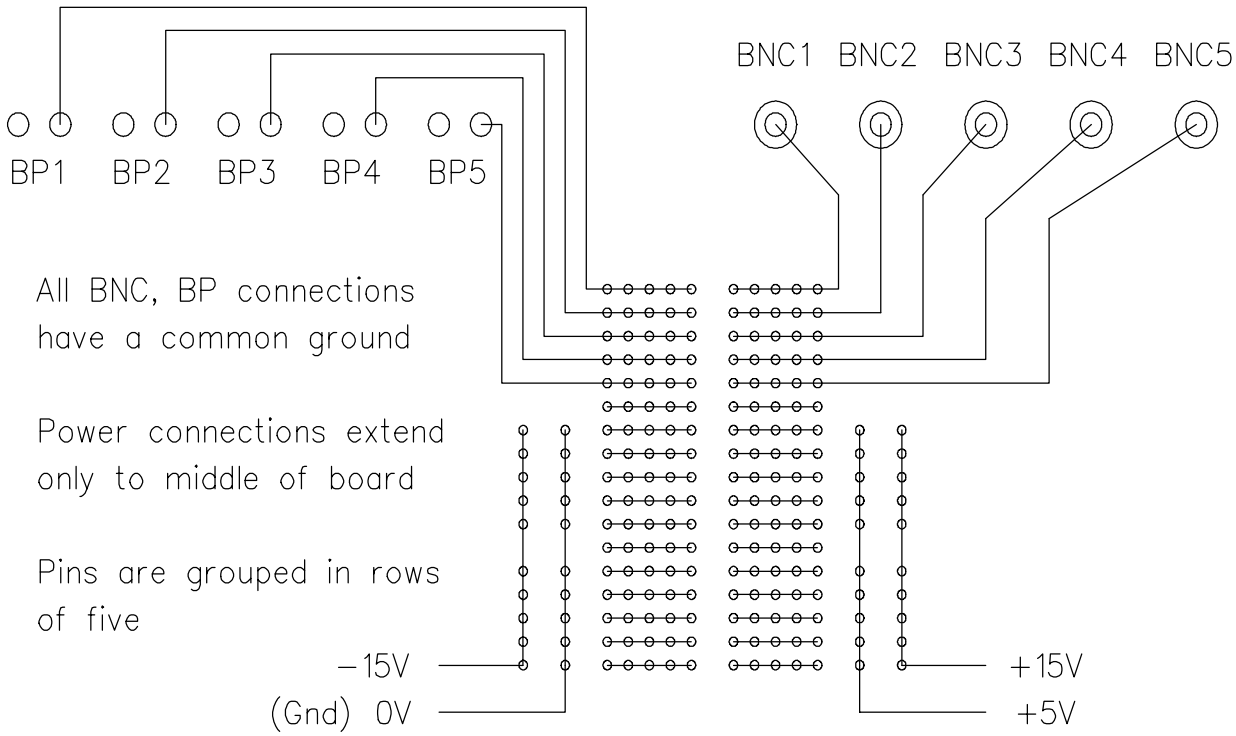


Figure A.1: Layout of common connections on the breadboards

The banana plugs and the BNC connectors are used to connect the components on the breadboard to the external devices such as meters, scopes, and function generators. You should not connect wires directly between the breadboard and an external device; it is unsafe. The proper breadboarding technique is illustrated in Fig. A.2.

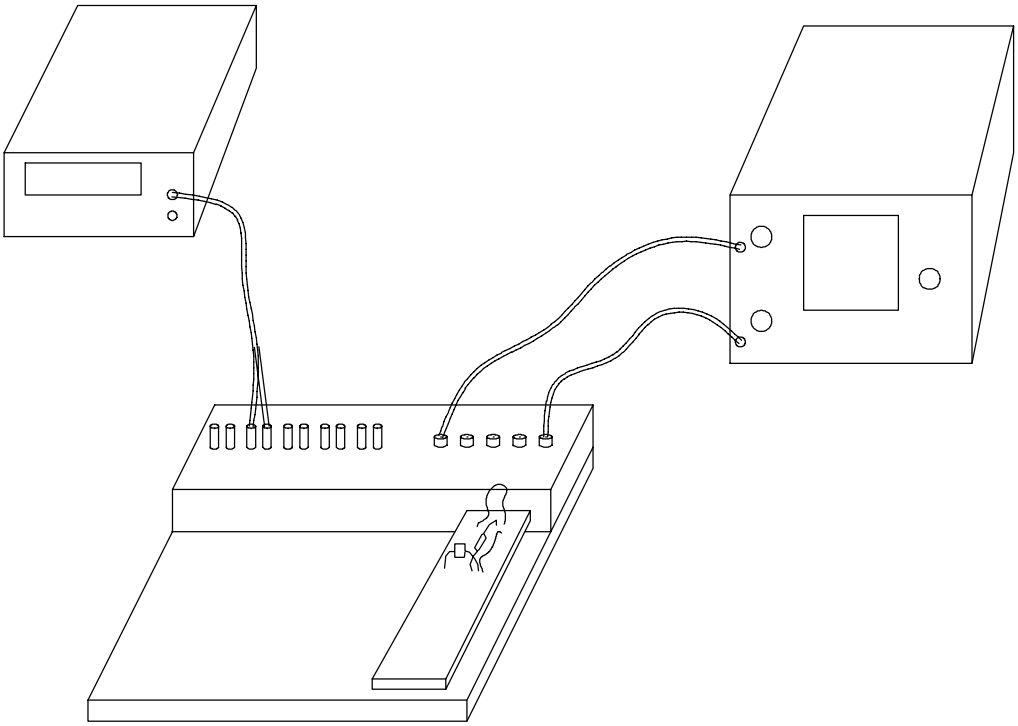
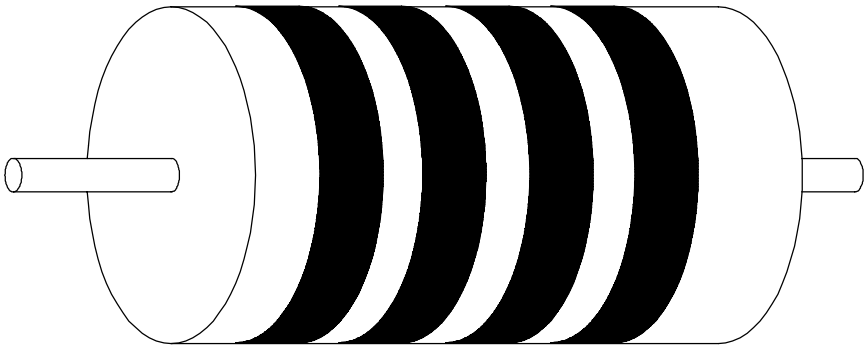


Figure A.2: The proper breadboarding technique

NAME (PRINT) | STUDENT ID | LAB DATE | GRADE

Appendix B

Resistor Colour Code



Colour	First Band	Second Band	Third 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}
Silver	-	-	10^{-2}

	Silver is	$\pm 10\%$ tolerance
Fourth Band:	Gold is	$\pm 5\%$ tolerance
	No band is	$\pm 20\%$ tolerance

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

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

NAME (PRINT)	STUDENT ID	LAB DATE	GRADE

Appendix C

Plotting with `physica`

An integral part of every lab is an analysis of the results, and it is best done with the help of a scientific visualization/plotting/fitting computer program. There is a large number of such programs for different computing platforms. If you are comfortable using one such package already, you may use the software you already know. However, bear in mind that:

- the software must be able to perform multi-parameter non-linear fits, and a proper statistical evaluation of convergence (*e.g.* χ^2);
- you must bring your own laptop computer to the lab;
- the instructor may not be able to help, not being familiar with the quirks of your software.

What is made available to you in the lab is a powerful scientific plotting and fitting package called `physica`, written at the TRIUMF accelerator in Vancouver, BC. This is the recommended software for use in the analysis of experimental data and in the preparation of lab reports, theses, and scientific articles.

The main `physica` "engine" is an "old-fashioned" piece of software in the sense that it has a command language and requires typing of commands at the prompt, and not clicking a mouse and using visual widgets. On the other hand, it is easy to learn, its numerical engine is an extremely powerful one, and a macro language allows you to automate many tasks using only a text editor. In order to harness the full power of `physica` you may need to spend some time learning its command language.

In addition, *Physica Online* is a web-based interface into `physica` which may prove adequate for most tasks. It is fairly self-explanatory and can be invoked by pointing a web browser to

`www.physics.brocku.ca/physica/`

For more advanced tasks, the web-based *Physica Online* provides the "expert mode" which does allow access to full capabilities of `physica`.

A stand-alone version of *Physica Online* is also installed on all the Linux machines in the Physics cluster under the name of `PhysicaLab`. It is a local Tcl/Tk script that acts as an interface into the same "engine" that drives the web-based *Physica Online*.

Recently, a new, graphical user interface version of `physica` called *eXtrema* has become available from `exsitewebware.com/extrema/`, in both Linux and Windows versions. You may want to download and install it on your home computer.

On-line `physica` tutorial

A quick way to get into `physica` is through the on-line tutorial created here at Brock.

- log on to a Linux workstation;
- open a web browser and an 80x24 shell window (a terminal icon) side-by side; point the web browser to

  `www.physics.brocku.ca/doc/physica/`

  and type `physica` in your shell window; a separate graphical output window will open up, and the shell window will display the `PHYSICA:` prompt; proceed at your own pace.
- You will likely want to use your favourite text editor to create small macro command files. You may want to arrange all windows side-by-side for convenience. Remember to **not** resize the graphics window of `physica` with a mouse (use a `resize` command at the `PHYSICA:` prompt).