

A Quick Guide to Microcontrollers

The following activities are designed to provide the user with useful concepts in electronics and microcontroller programming. I have chosen to base these exercises on the PIC16F876, my favorite low-priced microcontroller (MCU) and the CCS C compiler, not necessarily the best compiler, but probably the cheapest. An introductory text of electronics is a necessary companion to this guide, such as the McGraw-Hill [Benchtop Electronics Handbook](#) or [Practical Electronics for Inventors](#). Use these books to fill in the gaps or to learn more about a particular subject. A PIC prototyping board for the PIC16F876 is needed- they can be purchased at <http://www.melabs.com>. A digital multimeter is very useful and can be purchased cheaply at www.mpja.com. A basic text of the C programming language, a CCS C compiler <http://www.ccsinfo.com/ccscorder.html>, and a Microchip PIC Start programmer (or WARP 13 programmer) are also needed <http://www.microchip.com>. Users committed to the exercises presented here and willing to spend many hours hovering over solder fumes or in front of glaring CRTs may find themselves soon programming microcontrollers and even enjoying it.

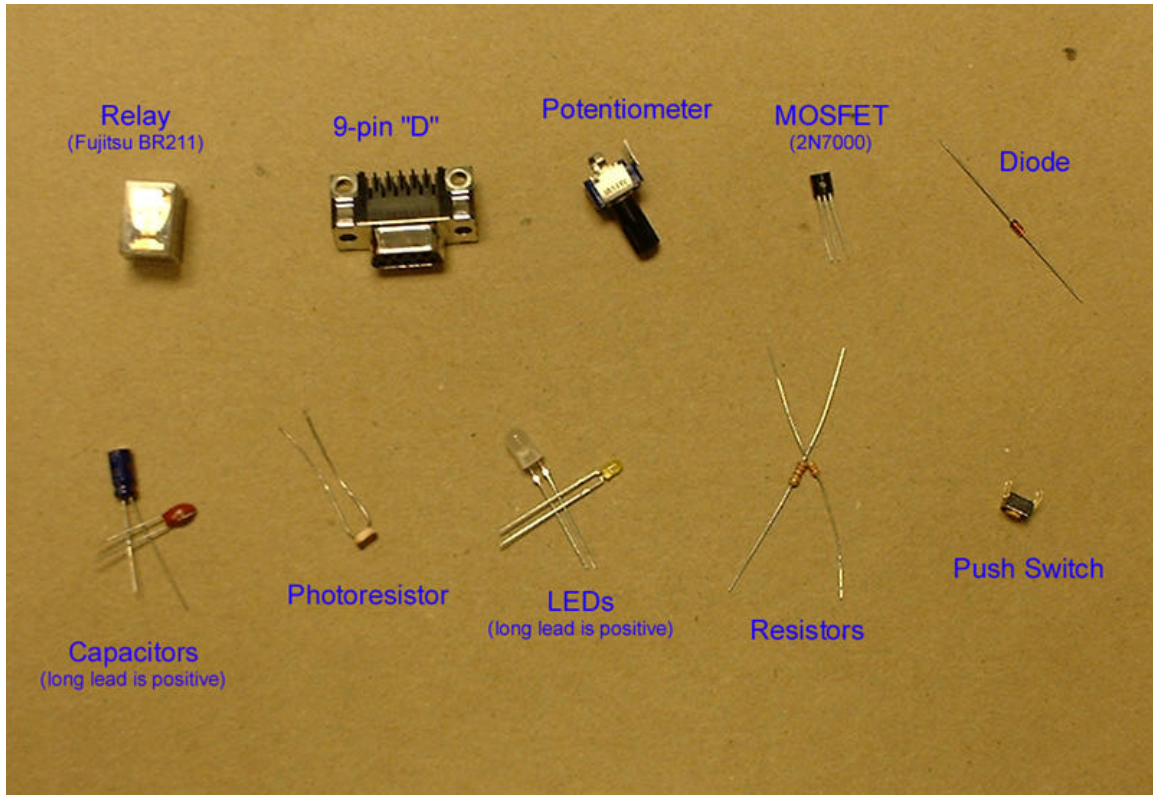
Modern microcontrollers are quite capable and flexible devices. The MicroChip PIC16F876 has 22 I/O pins, two PWM modules, three timers, a RS-232 port, and five analog-to-digital converters. The following exercises will discuss each of these capabilities and guide the user through their use. The goal is to provide enough experience with the basic concepts in microcontrollers and electronics¹ to let the user begin playing with ideas and to initiate them into the enormously helpful community of artists and engineers using these materials in their work. This is a heavy document that glances off many complex topics. Do not expect to understand all of it, but do not underestimate the time and commitment that these topics deserve. Most of all, build something fun with these exercises. It makes it all worthwhile.

Depending on your experience level, you may want to read up on the basic concepts in electricity and electronics (try to understand the concepts of voltage, current, resistance, circuits, capacitors, and the relationships between them). The activities that follow assume little previous knowledge, but they also assume that you have some outside resources and motivation to seek out additional ones as needed.

If you choose to design your own board, a schematic for the simplest circuit needed to run the 876 is included in the back.

¹ One important skill to master is the ability to read electronics specification sheets. One must wade through endless technical nonsense when all one really wants is a simple explanation. Included with this manual are spec. sheets for almost every component used. For some circuits in this manual, you might even have to read them.

Additional parts needed for the exercises:

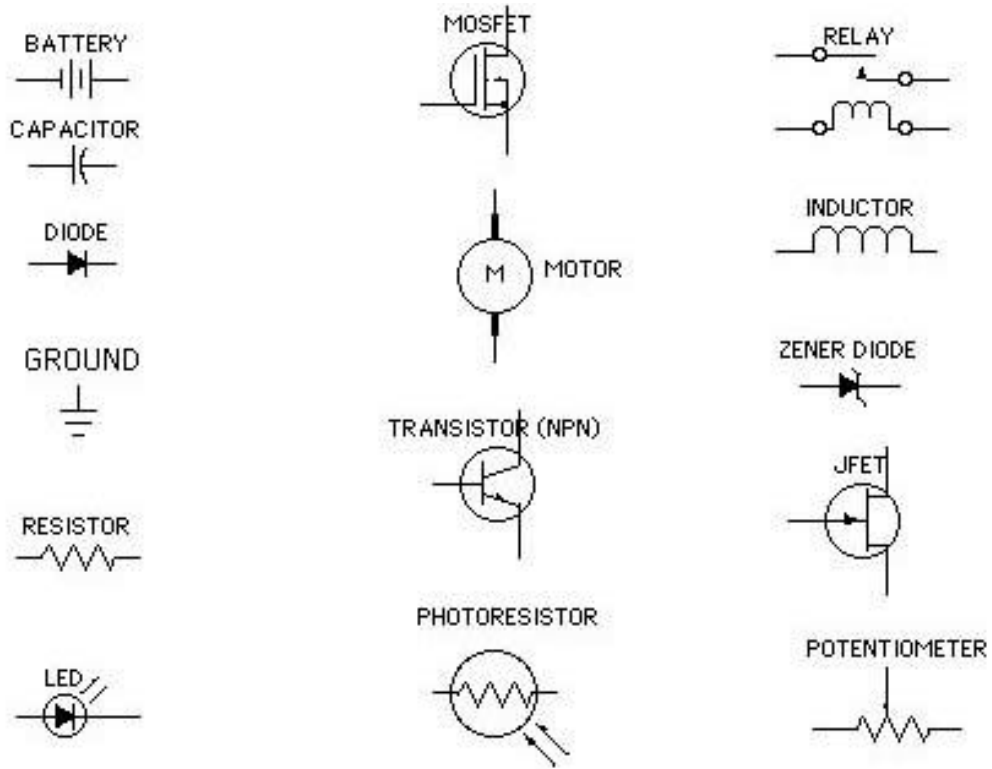


Digikey Part Numbers: (may be slightly different than those pictured above)

Z830-ND – Relay
P4E3103-ND – Potentiometer
2N7000FS-ND - MOSFET
N4001MSCT - Diode
A2100 – 9 Pin D
401-1182-ND - Switch
P392-ND – bi-color LED

Get the rest at Radio Shack. While Radio Shack is a substandard electronics store and my snobbery *almost* prevents me from shopping there, it is the only store close.

Common symbols used in electronic schematics:



Installing MPLAB and the CCS compiler and Compiling Activites1-6

1. After installing MPLAB on your computer and unzipping the CCS files into their own folder on your hard drive (put the unzipped folder on c:/, so that the path is simply c:/picc), open MPLAB. Choose Project/Install Language Tool. Now, choose CCS and the C-Compiler and then choose the executable PICC\CCSC.EXE (this is in the folder you unzipped, wherever you put it) You also want it Windowed and not Command Line.

2. Now we must tell the computer where to find the header files (the header files are the files specific to the type of PIC we are using that come included with the compiler). To do this (in Windows 2000), we click on the Start Menu and choose Settings/Control Panel. In Control Panel, choose System/Advanced/Environment Variables. Edit the System Variables section labeled Path and add the line c:/picc/EXAMPLES (yes, case matters). Do not erase or change any of the other variables. Now, restart your computer.

3. After copying the source code (available at www.junkfunnel.com/resources/source.zip) to you hard drive, select Project, New. Now find the source code folder, and create a new project with the name led.pjt in that folder (the project name should be the same as the source file, in this case 'led' (the source file is 'led.c'). Hit OK. The Edit Project window should pop up. The Target File name should be led.hex. You can leave the other boxes within the Project heading blank. If the Development Mode box does not read Editor Only16F876, click on the Change button. Click the check box labeled None (Editor Only), and find the PIC16F876 in the Processor field. Hit OK. Now, change the Language Tool Suite to read CCS. A box will pop up to warn you that you will lose your 'target command line options'. That's quite fine, so hit OK. Now, highlight the temp [.hex] file in the Project File box. The Node Properties button should now be available. Click on this. Click on the box labeled PCM. It should now be checked. Hit OK. Now, one last step. Click Add Node. Find the file led.c and add it. Hit OK, then back in the Edit Project dialog box, hit OK.

4. Now, we need to enable the PICSTART programming board. Click on the Menu item PICSTART Plus. With the programming board plugged into your serial port, click on Enable PICSTART PLUS.

5. To compile the file led.c, we choose Project/Build All. The program has compiled successfully if the resulting window reads 'Build Completed Successfully'. If there are errors in our code, the window will contain the line number of our error (or perhaps the line before it) and some cryptic error message. The reference for the CCS compiler will explain these error messages a bit better, but not great. Also, since the compilers only good clue to what is wrong with your program is the line number, we must use a text editor that includes the line numbers. I use the EditPlus text editor, available as shareware. Note: After we change anything to the source file, led.c, we must save it and compile it again.

6. After placing the PIC to program in the programmer (depressed dot on the PIC should be at the top of the programmer), we hit Program. The yellow light on the programmer will light up, and a dialog box should read 'success' after the programming is finished. You are ready to rock and roll. Place the PIC back in your protoboard and power it up.

Good luck!

Activity One: LED array!

The PIC16F876 has 22 *pins* available for use as an input or an output (I/O), divided among three *ports*. A port is a group of pins, intended to give the user control over an entire collection of pins at a time. These ports are names A, B, C. For instance, pin 5 of the PIC16F876 is named RA4, meaning pin 4 of the A port, while pin 12 is named RC1, meaning pin 1 of the C port. Don't think too hard about this, it's simply an imposed order. Before a pin can be used, we must tell the microcontroller whether it is an input or an output. We will start by making the pin RA2 an output with the function *bit_clear* \$A2². The name *bit_clear* has origins that we will discuss later, but now is not the time. For now, understand that *bit_clear* using PORT_X_DDR makes a pin an output (capable of producing 0 Volts or 5 Volts) and *bit_set* using PORT_X_DDR makes a pin an input (capable of reading 0 Volts or 5 Volts). Once the pin is set as an output, we can set its value low (0 volts) or high (5 Volts). This is also accomplished with the function *bit_set* but with PORT_X, which makes the pin high, and *bit_clear* with PORT_X, which makes the pin low. These functions look like:

```
bit_clear(PORT_A_DDR, 2); //make pin RA2 an output

bit_set(PORT_A, 2);      //turn pin RA2 high
delay_ms(500);          //wait 500 milliseconds
bit_clear(PORT_A, 2);   //turn pin low
delay_ms(500);
```

Let's add a LED to make this more visually exciting. We know that pin A2 will be set at 5 Volts. Most LED's will draw too much *current* at this voltage, so we will add a resistor to our circuit that will act as a *current limiter*. This will bring us to the first equation of the activity, the famous Ohm's Law:

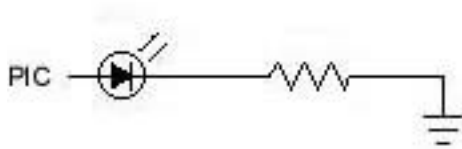
Voltage = Current * Resistance (V = I * R) (Units: V = Volts, I = Amps, R = Ohms)

We know that pin A2 will be at 5 Volts. This forms the left hand side of the equation. LED's are rated by the amount of current they can draw so that they don't burn out. The included LED's are rated at 20 mA. This is the current term. Then, we can assume that the LED itself has negligible resistance and solve for R.

$$5 \text{ V} = 20 \text{ mA} * R$$

$$R = 250 \text{ Ohms}$$

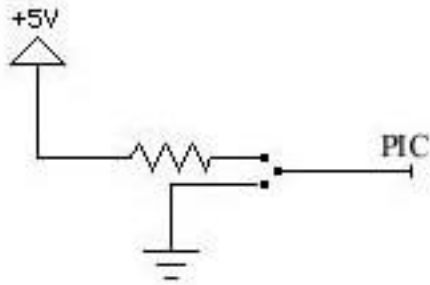
The schematic of the circuit you will build looks like the following:



Now, to turn this schematic into a circuit: On the prototyping area (known as a breadboard), use the small segments of wire included in your kit to connect pin A2 on the PIC to the long lead (positive side, called the *anode*) on the LED (for more information on using a breadboard and creating circuits from schematics, please see the appendix on this topic). Take the resistor (in this case, 250 Ohms) and connect one end to the short side of the LED (negative side, called the *cathode*) and the other end to the ground (Ground can be found on the bottom left corner of the board. One header is 5V, the other ground. If you look closely, they are labeled as such). After building this circuit, compile the *led.c* file and program your PIC. Stick the little guy in your protoboard, power up, and let him blink mindlessly.

Next, we will add a slide switch that will signal the microcontroller to turn on this LED. To do this, we will make pin A3 an input to check the state of our switch. We also will choose an input of 0V to be off and an input of 5V to be on, but this could be flipped. Once configured, the status of an input pin is checked using the *pin_test* function. This function returns a 1 at 5V and a 0 at 0V. This is the convention used in most digital systems (1 = high voltage level, 0 = low voltage level). Before wiring the below circuit, use the multimeter³ to probe the pins of the switch. To do this, use the resistance measuring function of the multimeter (called an *ohmmeter*, because it measures in ohms) to check which pins are connected with the switch in a given position, and which ones are connected when the switch is in the other position. Drawing a diagram of the switch might help. The pin that gets connected in either case will go to the PIC (probably the center pin). One of the remaining pins will go through a resistor to 5V, the other to ground.

³ See appendix for more details of using a multimeter



Compile led.c and try it out!

As a challenge, figure out how to safely use the push button to turn on the LED. Hints: There should be a resistor between 5V and any given path to ground or the PIC. Probe the switch...unlike the slide switch, you only have two pins which get connected on push, and nothing which is connected otherwise. This circuit is included in the appendix .

Now you are own your own. Go crazy- you have 20 pins on 3 ports to use any way you wish. Warning: Four pins on the PIC16F876 are special: **1.** Pin A4 doesn't have the ability to go high by itself. We must 'tie it high', as they say. Here is how it works: We place a large resistor (~10 kiloOhm) between the pin and 5V. This will allow the pin to go into its high state. However, when the pin is set low at 0V, the resistor is large enough that not much current flows. In fact:

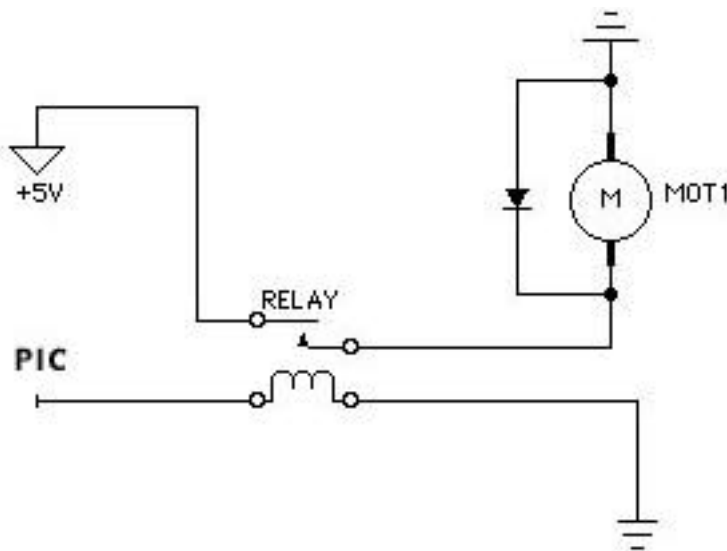
$$5V = I * 10 \text{ kOhm}$$

$$I = .0005 \text{ amps (very small!)}$$

Activity Two: Relays!

Try the below circuit to turn on and off a motor with a *relay*. A relay is a switch turned on and off by a voltage applied to the *coil* of the relay. Basically, current flowing through the coil produces a magnetic field that closes two pieces of metal together, thus completing a separate circuit. This separate circuit contains the load to be switched, in our case a motor, and a power supply. The battery in the diagram can be any direct current source, the motor can be any load that falls within the specifications of the relay used. Remember, a PIC pin can only source 20mA, so choose you relay to fit within that specification. Also, make sure that the motor and power supply are compatible. The diagram below may be a bit confusing. Lets divide it into two parts: The first is the coil, the little loop-dee-loop thing. Find this part on the relay spec. sheet. Wire one side to the PIC (pin RA2 again), and one to ground. When you compile and run the program relay.c below, you should here the 'click' sound of the switch flipping over (the blinkers of your car are controlled by relays, hence the familiar click-clack sound).

Now, lets look at the other part of the schematic, which is the switch. Your relay has on pin that is the *lead*, one pin that is *normally open*, and may have one pin that is *normally closed*. Without current through the coil, the *lead* is connected to the *normally closed*. With current flowing through the coil, the *lead* is connected to the *normally open*. With schematic in-hand and multimeter fired up, verify this. Then wire up the rest of the circuit as shown below. The pins shown in the diagram are the lead (with the little arrow) and the normally open.



The diode that bridges the motor is to protect the components from harmful voltage that builds up when the motor switches off. This is because motors have a high *inductance*.

Relays are just a switch, making them useless if you want to control the speed of a motor or dim a light. However, they are very useful at switching AC loads or very large DC loads. We will control the speed of a motor in activity 4.


```
delay_ms(500);
```

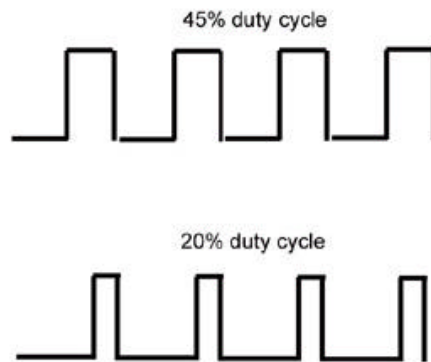
Hook up the bi-color LED, through a resistor between pins A0 and A1. (That is, run a wire from A0 to the bi-color LED, then a resistor from the other pin on the LED to A1.) Then compile and run the tris.c program.

This concept takes a while to fully wrap your head around. It is a trick, mostly. It takes advantage of a way of representing numbers and uses that representation for a completely different reason, like setting up pins. There is no reason that tricks like that should make sense- you just accept them at first and then appreciate them later.

Activity Four: Motors and MOSFETS!

In this activity, we will use one of two available Pulse Width Modulation (PWM) modules to power and control a DC motor. Later, we will also use this module to produce a continuously variable DC analog signal (from 0-5V).

Pulse Width Modulation refers to a continuous series of square pulses produced by a pin on the PIC. The width of the pulses is variable, as illustrated below. The *duty cycle* of a PWM signal is the amount of time the pin is held high in each cycle. For instance, a duty cycle of 50% means the pin is low for the same amount of time as it is high, while a 10% duty cycle means that the pin is high for 10% of the time. A PWM signal also has a frequency, which is the number of high-low cycles per second. This frequency is not important in our examples. A PWM signal fed into a motor effectively turns the motor on and off. However, this is happening very fast when compared to how long the spindle of the motor takes to slow down. In the end, the motor ‘averages’ the highs and lows of the PWM signal, and its speed is proportional.



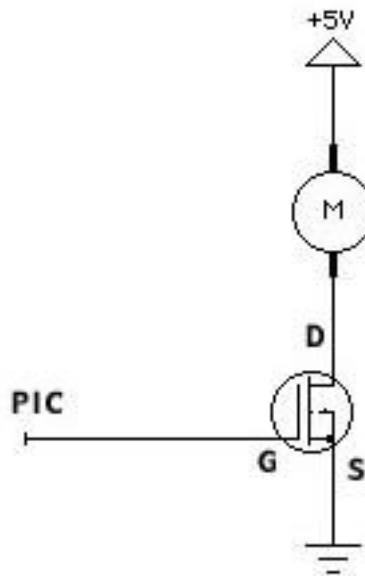
DC motors of reasonable size (big enough to do something useful with) are power-hungry devices. They draw anywhere from 100 mA to 10 A or more. A pin on our PIC can only *source* 20 mA. While the relays used in Exercise 1.5 can turn a motor of this type on, the mechanical switching action of the relay is not fast enough to keep up with the high-speed PWM signal. For this exercise, we will use a high-current MOSFET (metal oxide field effect transistor, not that it matters) that acts just like a relay, but can switch faster since it does not use moving parts to switch. The amount of current that can run through a MOSFET depends on the MOSFET, but we will use .4 A devices. Note that the pins of the MOSFET are labeled as D,G,S (in the spec. sheet). These stand for Drain, Gate, and Source, respectively. The Drain is the *drain of electrons*, or where the electrons flow to, which is the positive voltage supply. The source is the source of electrons, which is the ground. This is confusing! Current flows from positive voltage to ground, right? Yes, but only because current was defined wrong by Benjamin Franklin way back when. Current does flow from higher voltage to lower voltage (by definition), but what is actually flowing are electrons (from lower voltage (or ground) to higher voltage). To avoid confusion, always think of current flowing from high to low, except when reading spec. sheets and seeing the terms *drain* (or *V_{dd}*, which is the drain voltage)

and *source* (V_{ss} , source voltage). Don't blame Ben. He knew something was flowing and guessed wrong. He did bring us bifocals and chimneys, after all.

The PWM module on the PIC16F876 relies on an internal timer, specifically Timer2. The first step in using the PWM module is setting up this timer. We use to the function *setup_timer2* with a 1,2 or 3 passed as an argument. Passing a 1 will setup the timer near its fastest rate and 3 slowest, with 2 somewhere in between. Verify this with an oscilloscope once the PWM module is running. The next step is turning the PWM module on. The PIC16F876 had two modules, PWM1 on RC2 (pin 13) and PWM 2 on RC1 (pin 12). The functions needed are:

```
setup_timer_2(T2_DIV_BY_1,255,1);    //setup timer2
setup_ccp1(CCP_PWM);    //setup the pwm module
```

Then, we set the duty cycle of the module. The function is *set_pwm1_duty* or *set_pwm2_duty* followed by an argument 0-255. One would imagine that the duty cycle should be 0-100. I would think that too. CCS doesn't. Use 0-255. The circuit is sketched below:



Compile and run *pwm.c*.

After successfully driving the motor, remove it and the MOSFET. Place a 100 μF capacitor (mind the polarity) in its place, with the positive side of the cap on the PWM line and the negative side to ground. Write a program to sweep the duty cycle from 0 to 255 repeatedly. While running this program, measure the voltage level of the positive

side of the capacitor⁵ with a voltmeter or oscilloscope. Analog out! Neat trick, although it turns out it isn't necessarily useful in most circuit designs. Some interesting things *might* be possible, as there are voltage-controlled amplifiers and filters used in musical synthesizers that might be interfaced to a PIC using this technique. One more PWM trick: Replace the capacitor with an LED (wire a resistor and LED from the PWM pin to ground). Change your program to ramp from 0% duty to 30% and back down again (as shown below). You should see a nice 'heartbeat' from the LED.

```
for ( i=0;i<30; i++) //this is a common control stucture in C
{
    set_pwm1_duty(i); //set the duty cycle to the variable 'i'
    delay_ms(50);
}

for ( i=0;i<30; i++)
{
    set_pwm1_duty(30 - i);
    delay_ms(50);
}
```

⁵ The value of the capacitor here should be played with. Too small of a capacitor and you will still be able to see individual spikes of the PWM signal with an oscilloscope. Too big, and it might take a mighty long time to discharge itself. You can also place a resistor in series from the PIC to the capacitor and see what effect it has. Watch the difference on an oscilloscope.

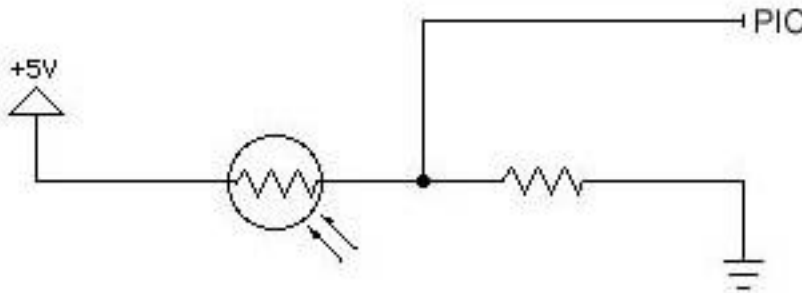
Activity Five: Analog to Digital (A/D) Conversion:

Digital systems have many advantages and have led to many great advances. However, the sensors used to sense or detect an aspect of the physical world measure an analog signal, and we are forced to convert an analog signal to a digital one for it to be useful. Fortunately, this is largely done for us in the PIC. It contains 5 pins (A0, A1, A2, A3, A5) that can be configured to perform direct AD conversion on a signal between 0V and 5V. The functions we will use to setup the ADC are:

```
setup_adc_ports(ALL_ANALOG);           //setup the port to be a convertor
setup_adc(ADC_CLOCK_INTERNAL);        //another needed call
set_adc_channel(0);                   //setup pin A0 as the adc channel
```

The function *set_adc_channel* gets passed a number (0,1,2,3,5) for the pin of Port A to perform the conversion on. The function *read_ad* returns a 10-bit number (0-1024) proportionally to the signal, with 0 representing 0V and 1024 representing 5V.

Many sensors use environmental factors to change the sensor's resistance, such as thermistors (temperature effects resistance), photoresistors (light effects resistance), or force-sensitive resistors (FSRs). To create an analog voltage signal from these devices, we create a voltage divider with the sensor and an ordinary resistor, as shown below.



Lets analyze this circuit. The total resistance from 5V to Ground is the resistance of the sensor (R_s) plus the resistance of the resistor (R)⁶, in our case we will use a 30-50 k resistor. The current flowing through both of these resistors is $I = 5V / (R_s + R)$. Now, since current cannot flow into the PIC (it has very high resistance, or *impedance*), the

⁶ How do you pick the value of this resistor? You should match this resistor to the change of resistance of your sensor in the conditions you are most interested in detecting. Just be certain that the total resistance at any time is bigger than a kiloOhm or more so that the current flow is small.

current flowing through the sensor is equal to the current flowing through the resistor (where else would it go?). We use this fact to see that:

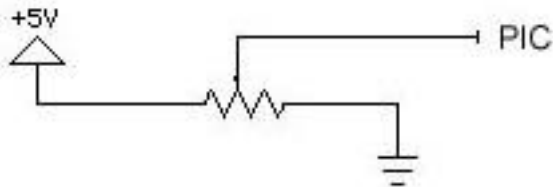
$$(5V - V_{PIC}) = I * R_z$$

Solving this for V_{PIC} gives us $V_{PIC} = 5V - 5V * R_s / (R_s + R)$

Note that the voltage at the PIC (V_{PIC}) is proportional to the resistance of the photoresistor (R_s). Just what we wanted! We have our analog voltage signal for the PIC to measure! Setup this circuit, running the variable analog voltage (labeled PIC above) to pin RA0 of the PIC. Wire up a LED like you did in the PWM exercise and run `adc.c`!

More:

There are devices known as variable resistors that have a knob or slider that is used to vary the resistance of the device. Many of these devices are set up in a voltage divider configuration for easy AD conversion. These devices are known as a potentiometer. We use one below with the same AD functions.



The center pin of a potentiometer is usually the *wiper*, or the point of variable resistance. This pin will be connected to the PIC, and the outer two pins to 5V and ground.

Exercise Six: Serial I/O

(Note: this exercise requires the installation of the Proce55ing program from <http://www.proce55ing.net/>)

Serial communication is a method of communicating between electronic devices. It refers to the passing of information bit-by-bit in time, in contrast to parallel communication, in which many bits are passed simultaneously. Many different protocols exist for this serial exchange. One very useful and ubiquitous protocol is known as RS-232 and it is used, amongst other things, for communication between computers and their peripherals (although this use is being rapidly replaced by USB protocol). The nice feature of RS-232 is that most of the work is done for us, either in the PIC itself or in the C compiler. We will be using RC4 and RC5 to send and receive.⁷



(Backside of the female D connector)

Setup the photoresistor circuit shown in Exercise Five. Connect pin two of the 9-pin D connector to RC4, pin three to RC5, and pin five to ground. Compile and run `serial.c`.

Once this is running, run `Sketchbook/Standard/SimpleSerialDemo.pde` within the Proce55ing environment and connect the serial cable to the computer. Proce55ing is an environment and libraries made specifically for graphics programming in Java. Once you have opened `SimpleSerialDemo`, click on the right arrow button to start. A small square should appear and change from black to white as the light level on the photoresistor changes. Serial data from the PIC might also be used for controlling graphics in Java, video in Director, or sound in Max. Much information regarding this can be found on the glorious, glorious Internet.

We can also receive serial on the PIC from the computer. Check into the functions `getc` and `gets` in the compiler reference manual.

⁷ The PIC 876 has a 'hardware' serial port built in, called a USART (universal synchronous/asynchronous receiver/transmitter built in). However, this feature was meant to work with a *level converter*, which converts the 0-5V signal to -12 to 12V (most computers these days don't need these levels...0-5 works fine). These level converters also invert the signal, which means that the signal coming from the hardware USART is inverted to what the computer wants to see...thus we are forced to not use the hardware USART, unless we want to wire up a level converter. However, this only means slightly larger and slower code on the PIC, and nothing more.

Appendix:

Binary and Hexadecimal numbers:

We use the decimal representation of numbers in ordinary math. However, computers operate with a binary representation. At some point, people realized that a hexadecimal representation was a good compromise between the two, allowing humans to better visualize numbers as the computer might see them, but not make them stare at ones and zeros either. We will briefly cover all three and how to convert between them.

In the decimal system, we count from 0 to 9, then add a place holder in the next space over to represent 10. We then continue counting from 0 to 9 again. In binary, we count from 0 to 1, placing a 1 in the next space over when we hit one in the first. In hexadecimal, we count from 0 to 15 before marking the next space. However, the number 15 takes two spaces! To solve this, we start counting in with letters after nine..ie, count from zero to F, going from 0 to 9 to A, then B, and so on.

Decimal: 0,1,2,3,4,5,6,7,8,9

Binary: 0,1

Hexadecimal: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Binary is denoted by a lower-case b, hexadecimal by a '\$' or '0x' prefix or a 'h' postfix.

To convert, it's best to use calculator! In time, you will be able to convert back and forth in your head. For now, try out Marin Steen's Hex Calculator at <http://www.martin-steen.de/hexcalc.html> or use a scientific calculator.

Pinout for 9-pin D connector

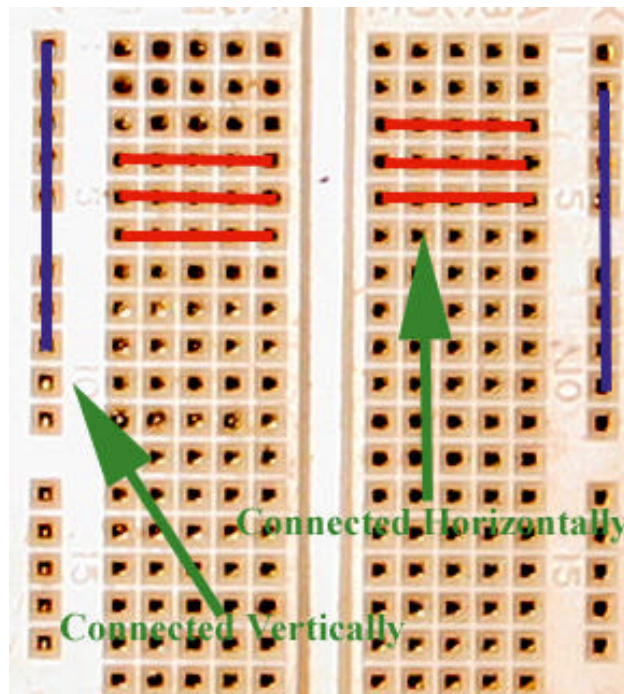
Pin No.	Function	Pin No.	Function
1	DCD (Data Carrier Detect)	6	DSR (Data Set Ready)
2	RX (Receive Data)	7	RTS (Request To Send)
3	TX (Transmit Data)	8	CTS (Clear To Send)
4	DTR (Data Terminal Ready)	9	RI (Ring Indicator)
5	GND (Signal Ground)		

Multimeters

Multimeters are many measuring instruments packaged in one device. These instruments are usually an ohmmeter (measures resistance), a voltmeter (measures voltage), an ammeter (measures current), and sometimes a faradimeter (measures capacitance). Also, some have a continuity tester, which beeps when a connection exists between the leads. This is a handy feature. The leads of the multimeter may or may not have different plugs for the different functions. This is important to remember. Also, the ammeter will certainly have a maximum input current it can accept. Do not ignore this. Also, there might be a switch for AC/DC coupling. Use DC for all purposes in this manual. For probing a switch, use the continuity tester (if possible) or ohmmeter on lowest setting. For measuring voltage, connect the black lead to ground and the red lead to the voltage being measured. For measuring amperage, connect the device in series with the circuit, so that current is flowing in the red lead and out the black. For measuring capacitors, make sure that you discharge them by touching their leads together before inserting them into the faradimeter.

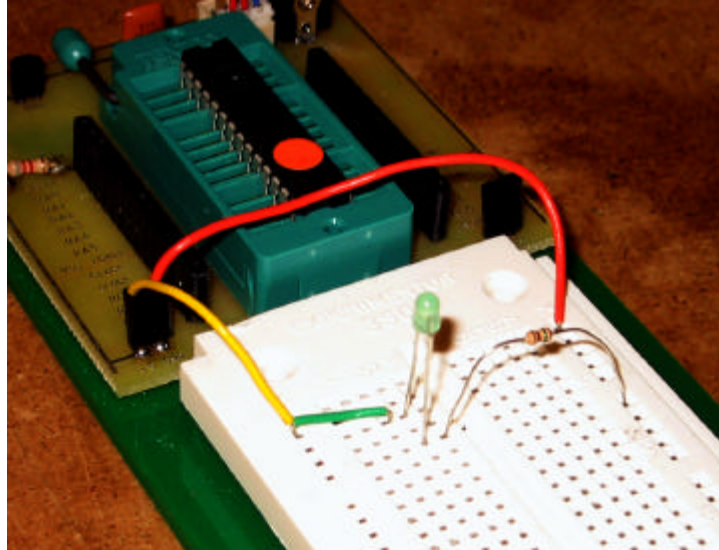
Using Breadboards and Building Circuits

A breadboard (or solderless breadboard) is a device that allows for rapid creation of electronic circuits. Notice that there are two different types of hole patterns. The holes in the center area of the board are electrically connected horizontally. The holes that run down the edges of the board are connected vertically, as shown in the picture below.

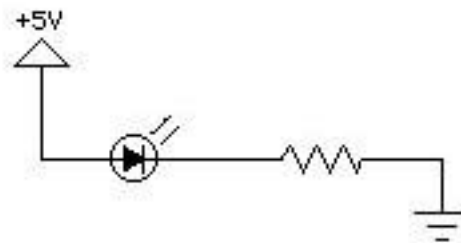


The outer edge holes (sometimes called rails) are usually used for 5V and ground (0V). By plugging in a wire to the 5V header and plugging in the other end to the left hand rail,

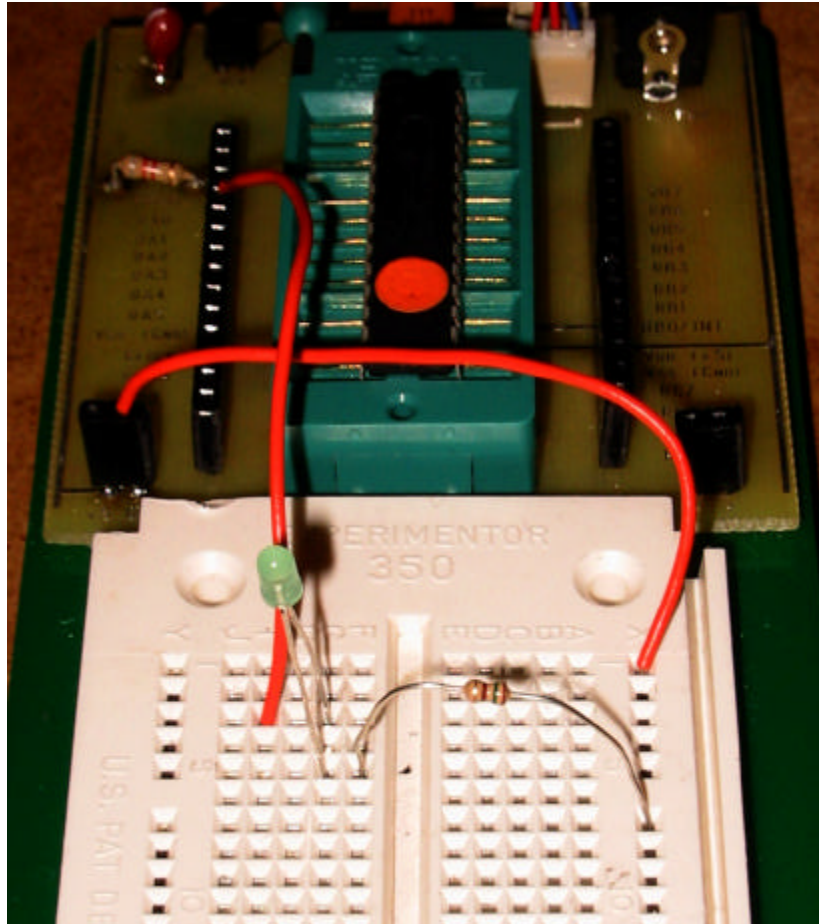
we can now connect 5V into any circuit by utilizing the fact that all of its holes are connected vertically. Do a similar thing for ground. Now, plug a wire into the left hand rail (5V), and insert the other end into an arbitrary hole in the center area. Since these holes are connected horizontally, we will now plug the long end of an LED (the anode, or positive voltage side) into any hole in that row. This lead will now be at 5V. Insert the short end of the LED into another arbitrary hole (anyone except one in the same horizontal row as the first!) Now plug one end of a 100 Ohm-1kOhm resistor into a hole in the same horizontal row as the short end of the LED. The other end of this resistor should now be plugged into the ground rail. Your circuit should look like this:



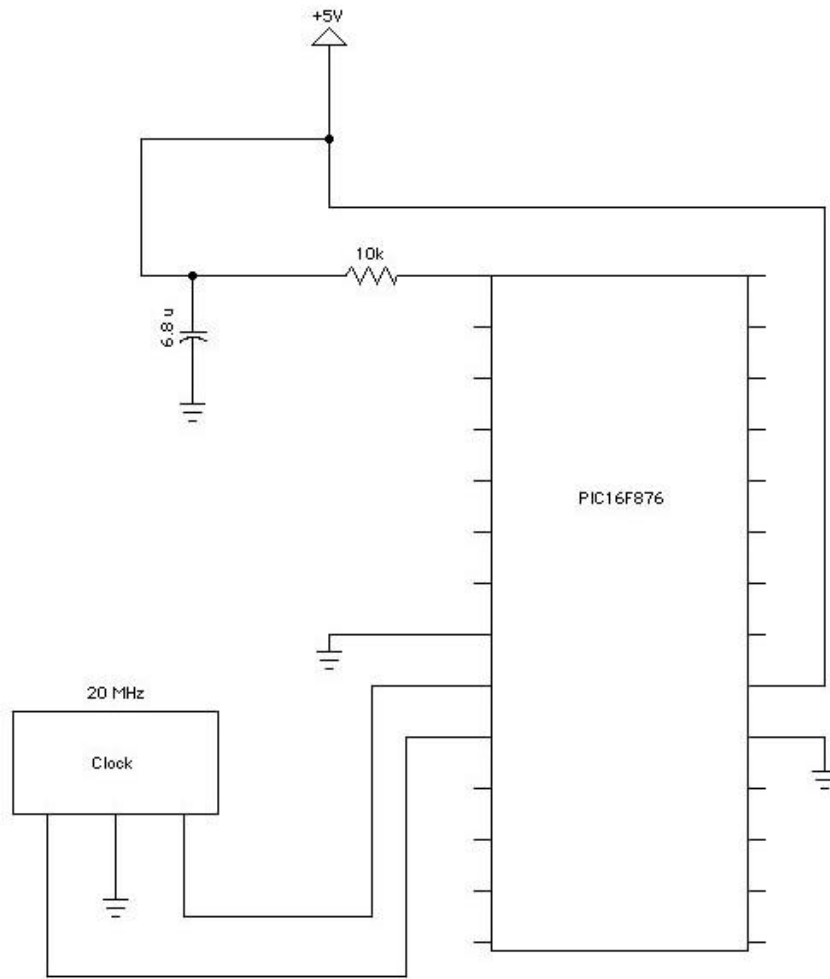
The schematic of this circuit looks like:



The schematic pictured in Activity One looks like this on the protoboard:



Simplest Circuit Needed to Run the PIC16F876



Button Schematic

