PIC 16F877A Tutorials for Pitt Robotics Club

Lukas Hoffmann, 2010

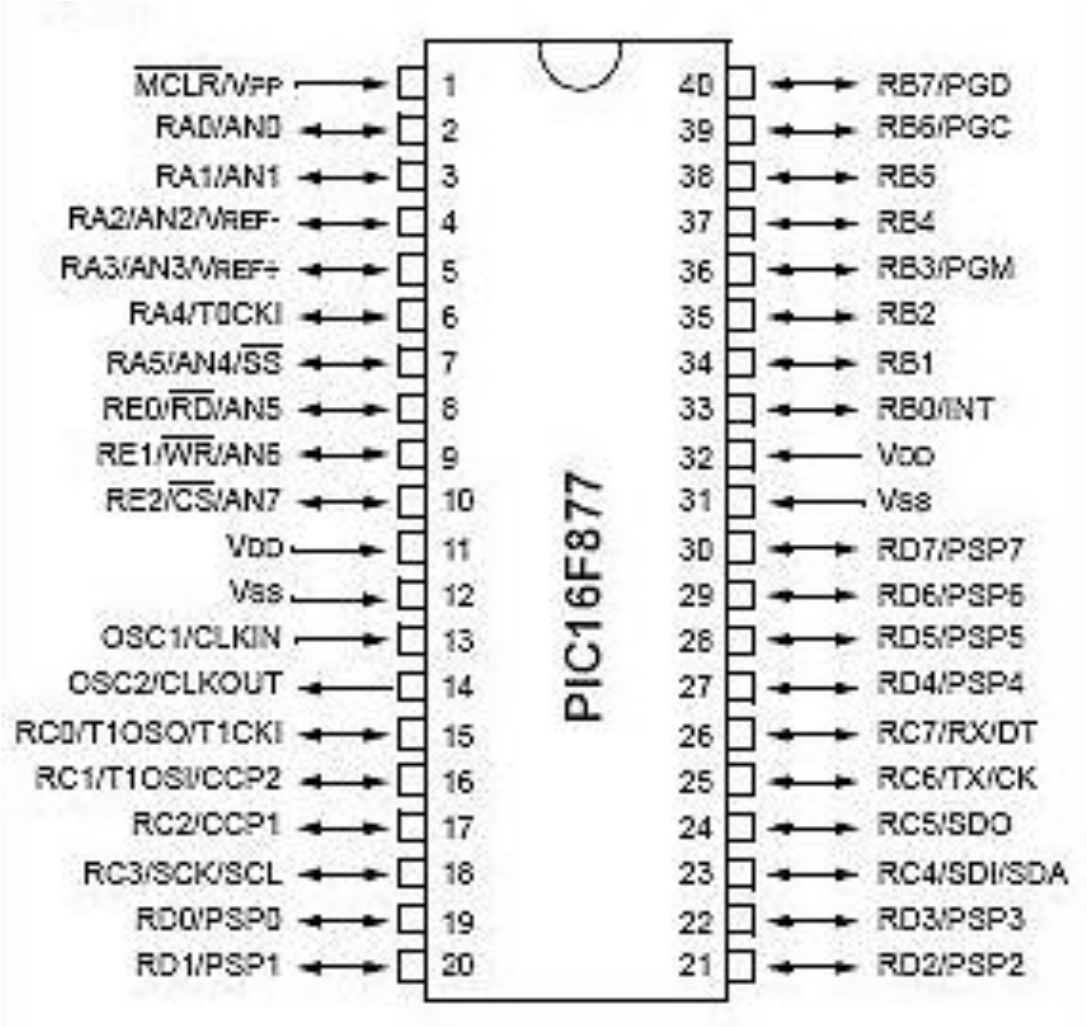
These tutorials demonstrate how to code simple input & output for your robot using the popular PIC 16F877A microcontroller. They help you get your sensors & motors working. The rest of the code is up to you!

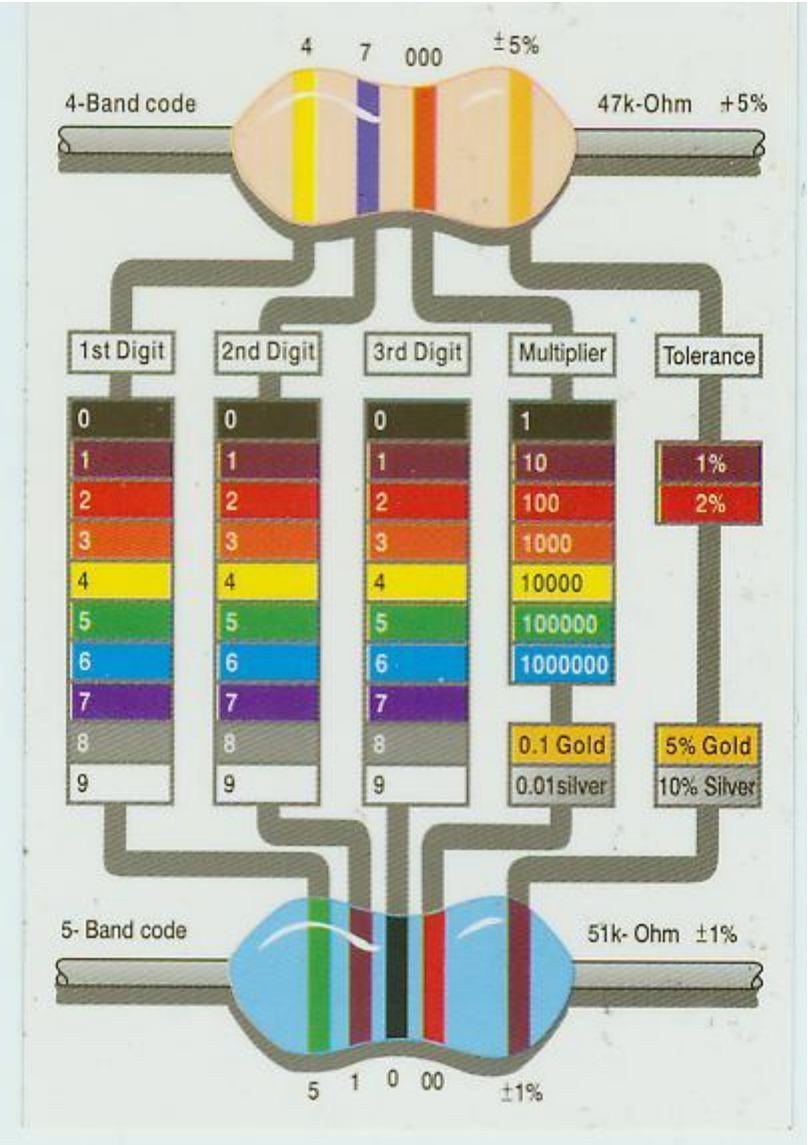
Questions? Visit our website: http://www.pitt.edu/~sorc/robotics/
Email: roboticsclubatpittsburgh@gmail.com

The tutorials assume basic knowledge of programming and electronics. If you are new to programming, you should learn the basics of C or C++. You should also know rudimentary electronics, and how to construct simple circuits.

I got most of this information from the PIC MCU Compiler reference manual & the help of other club members. You are welcome to edit or add new tutorials. Please email the club if you make another tutorial.

Introduction to the PIC16F877A



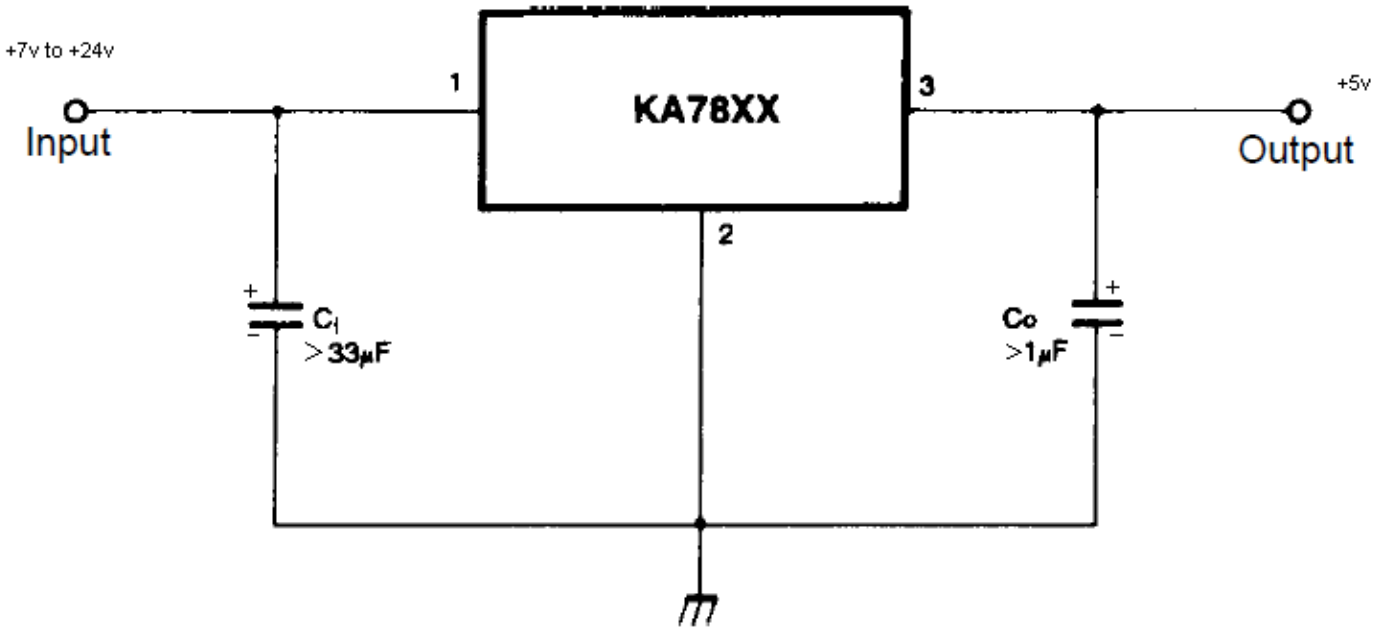


How to hook up the PIC so it will run?

Materials

PIC16F877A
Breadboard
20 Megahertz crystal oscillator
2 7-20pf capacitor
Black & red wires
>7.2 volt battery

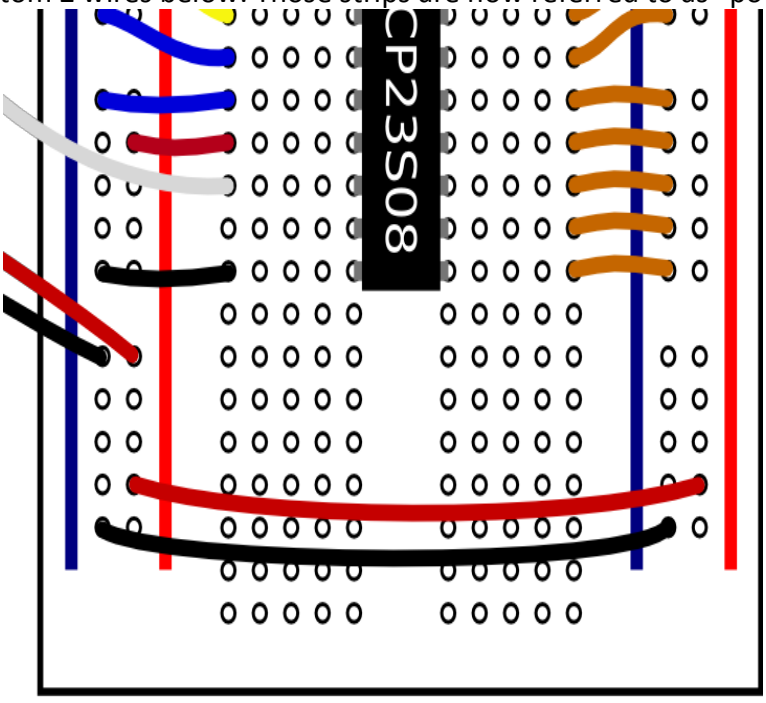
- For use with a higher voltage supply, you will need to use a voltage regulator to convert the higher voltage to 5v. Typically a 7805 is used. Look up the datasheet by searching Google “7805 voltage regulator”.



- If you don't know how these materials work or what they look like, look it up online.

Circuit

1. Insert PIC on breadboard. Refer to pin diagram above for pin numbers. To prevent confusion, orient the PIC so the dimple on top faces the same way as the pin diagram.
2. Connect breadboard side strips - power to power (red strips), ground to ground (black strips). Just like the bottom 2 wires below. Those strips are now referred to as "power" & "ground".



3. Power to Pin 1 (Vpp) – red wire (connected via 47k Resistor in between)
4. Power to Pin 11 (Vdd) – red wire
5. Ground to Pin 12 (Vss) – black wire

6. Power to Pin 32 (Vdd) – red wire
7. Ground to Pin 31 (Vss) – black wire
8. Crystal Oscillator to Pins 13 & 14 (OSC1, OSC2), the polarity doesn't matter. (polarity = which of the 2 wires goes in which hole).
9. Capacitor into power & ground – make sure the polarity is correct according to the datasheet for the capacitor. If not it could damage the circuit.
10. Ground to battery ground (-) – black wire.
11. Power (red) to battery power (+) – red wire. Or, solder that wire to a switch & run another wire from the switch to battery power, so you can switch the power on & off.

If using voltage regulator:

raw battery power wire goes into voltage regulator input
breadboard ground strip goes to battery ground
voltage regulator output wire (+5 volts) goes to breadboard power strip.
voltage regulator ground wire goes to breadboard ground strip

What do the pins mean?

The pins RB0-RB7, RC0-RC7, and RD0-RD7 are digital I/O pins. The pins CCP1 and CCP2, which share locations with RC1 and RC2, can be used for a PWM signal (see DC Motor tutorial). The pins AN0-AN7 are for analog I/O (see Photoresistor tutorial). TX and RX are for debugging I/O (see Output Messages to Computer tutorial). The remaining pins deal with power/ground, the clock signal, and programmer I/O.

A PIC is made of several "ports." Each port is designated with a letter, RB0-RB7 are a port. RC0-RC7 and RD0-RD7 are a port as well. RA0-RA5 and RE0-RE2 are also ports, but with fewer pins. Some of these pins have special purposes, but most can be used as basic input/output pins.

For example, you can set pin RB0 to be either an input pin, or an output pin. As an input pin, the digital voltage on the pin can be read in. For example, if RB0 is connected to ground (0v), then you would read a digital 0. If RB0 was connected to power (5v), then you would read a digital 1.
On the other hand, if you wanted to set RBO as an output pin, you could choose to make RB0 either be 5v, or 0v. This can be used, for example, to turn off or on a LED, or to turn off or on a motor.

How to program the PIC?
I omit step by step details of how to program the PIC because software always changes.

As of spring 2010, the club uses software from CCS (http://www.ccsinfo.com/). We use the PCW Compiler to write & compile programs and CCS Load and the ICD-U40 programmer to write programs to the PIC. Always load a .hex file onto the PIC – the compiled machine language program. The hex file should be located in the same folder as your project file.

You need to connect the ICD's Ethernet cable pins to the appropriate pins on the PIC in order to load a program. You also need to have the PIC running (i.e. under +5V power) when programming.
See "How do I connect the CCS ICD to my own hardware?" in the CCS FAQ,
http://www.ccsinfo.com/faq.php?page=icd_connection.

Don Crouse, the 2010 president, made custom programming boards to do that. They might still be in the lab. You can also cut up the wires on an Ethernet cable & connect them to the appropriate pins on the breadboard. If you don't want to make your own device, you can buy what you need to program the PIC.

Another available programmer the club most likely had is the PicKit 2. To program with a PicKit, connect the pin marked by the arrow on the PicKit device to the mclr pin. Pin 2 (next to the arrow) is +5, Pin 3 is Gnd, Pin 4 is B7, Pin 5 is B6, and Pin 6 is the optional debug pin PGM (not necessary to connect). Also, connect the PIC to power & ground (the Vdd/Vss pins). To program a PIC, connect the PIC IC to the PicKit, and a usb mini cable from the PicKit to the computer. To use the PicKit, run the PicKit V2 program on the desktop of the PC. The first necessary step, if the PicKit is configured to program a different pic than your own, is to select the family of the PIC. For the 16f877A, you must select "base device" in the device options. In order to program a hex file, it must be loaded / imported first. Do this by going to file and importing your program's hex file. After the file is loaded, click on the "write hex" button. This should write your hex file and say it programmed successfully. The program will automatically start running even with the PicKit connected unless the /MCLR checkbox is selected.

Note: The PicKit has the capability to supply power to the pic via the USB bus. It will automatically detect if the device is powered when trying to program. If you want to run your PIC just from the USB, you can select the "Vdd On" checkbox. This can supply up to 1A of current at 5V. Make sure the voltage number next to the checkbox is set to 5 – unless your processor does not support 5V. The PIC used in this tutorial is a 5v device.

Miscellaneous Advice

You must have the PIC wired up & be able to compile & load programs to do the other tutorials.

<u>To run the other tutorials, follow these steps:</u>

1. Connect circuit as described.
2. Compile the example code with whatever compiler you are using.
3. Load the program onto the PIC.
4. Turn on the power to run the program

If you don't know how to code something, look at old PIC code on the robotics lab's computers. I have found that we need to solve similar programming problems year after year, especially when we are talking about sensor input & motor output. Chances are good that somebody already wrote it. The PIC MCU C compiler manual is a good reference too.

I recommend that you try to be neat when attaching wires & other components. Cut your wires so that they lie flat on the breadboard, not sticking up. Make sure the bare ends are long enough to make a solid connection with the breadboard – you don't want a loose wire! Be orderly when you run wires to the various sensors & motors, too. You will have a confusing spaghetti forest when you add sensors, motors, switches, and lights.

Another good idea is to use wire colors consistently. For example, every time you have a wire going to ground, use the same color, usually black. That way, you can understand what it does with a glance. This is important when you have 50+ wires on the board. I use red for power (the + on the battery & breadboard), black for ground (the -), blue for inputs, and green for outputs.

Always double check your connections before flicking the switch. If just one wire is in the wrong position, it could disable your circuit or burn something out.

You can build your circuit with test LEDs, to make sure it is working when you turn it on. I have a power light directly connected to the battery to so I know when the circuit is on. I also have an LED controlled by the PIC. In my programs, I make the PIC turn the LED on when it starts running. That way I know that the PIC is working correctly.

The Anode of the LED goes to the +5v rail out of the regulator. The cathode of the LED goes to a current limiting resistor – any value between 220 and 1k is fine. The other end of the resistor goes to Ground. This LED should turn on at full brightness when the circuit is turned on. If the LED does not turn on, you may have the LED in backwards. If you know the LED is in correctly, and it does not turn on or turns on dimly when the power is turned on, TURN OFF YOUR CIRCUIT, and check your wiring. This most likely means there is a short somewhere with your power rails.

If you have been running the robot for a while, check the power battery voltage. If the voltage gets too low, the robot will behave strangely due to PIC doing a "brownout".

LED Blinker

Materials

Circuit from "Introduction to the PIC16F877A"
100 Ohm resistor
LED

Circuit

1. Pin RB7 to resistor
2. Resistor to LED
3. LED to ground

Code

```
//all these # below set up the PIC
#include <16F877A.h>
#device adc=8
#FUSES NOWDT     //No Watch Dog Timer
#FUSES HS        //Highspeed Osc > 4mhz
#FUSES PUT       //Power Up Timer
#FUSES NOPROTECT  //Code not protected from reading
#FUSES NODEBUG    //No Debug mode for ICD
#FUSES NOBROWNOUT //No brownout reset
#FUSES NOLVP      //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD      //No EE protection
#use delay(clock=20000000)                                    // Sets crystal oscillator at 20 megahertz
#use rs232(baud=9600, xmit=PIN_C6, invert) //Sets up serial port output pin & baud rate

//main program starts here
void main() {
  //Infinite program loop starts.  LED blinks forever.
  while(true){
    output_high(PIN_B7);   //send a "1" to pin RB7, making RB7 "High" at 5v
                                                    //this will turn on the LED hooked to RB7
    delay_ms(500);                  //wait half a second, delays for 500ms
    output_low(PIN_B7);    //send a "0" to pin RB7, making RB7 "Low" at 0v
    delay_ms(500);                  //wait half a second, delays for 500ms
  }
}
```

Notes

You can easily add more LEDs and make them flash in different patterns.
For more readable code, use

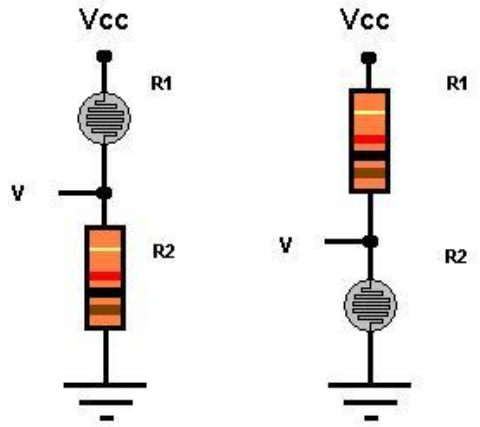
```
#define RED_LED   PIN_B7
#define GREEN_LED PIN_B6
…
…
output_high(RED_LED);
output_high(GREEN_LED);
```

Photoresistor Input

Materials

Circuit from “Introduction to the PIC16F877A”
Photoresistor
1K ohm resistor – or whatever is appropriate for your photoresistor
Wire

Circuit – the diagram on the right



1. Breadboard power (Vcc) to 1K-ohm resistor.
2. 1K-ohm resistor to photoresistor power.
3. Photoresistor ground to breadboard ground.
4. Wire from between resistor & photoresistor to PIC pin AN0.

If R1 is the photoresistor, the voltage will increase with increasing light intensity. If R2 is the photoresistor, the voltage will decrease with increasing light intensity.

Code for single photoresistor

```
//all these # below set up the PIC
#include <16F877A.h>
#device adc=8
#FUSES NOWDT     //No Watch Dog Timer
#FUSES HS       //Highspeed Osc > 4mhz
#FUSES PUT      //Power Up Timer
#FUSES NOPROTECT  //Code not protected from reading
#FUSES NODEBUG    //No Debug mode for ICD
#FUSES NOBROWNOUT //No brownout reset
#FUSES NOLVP     //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD     //No EE protection
#use delay(clock=20000000) //crystal oscillator at 20000000 hertz
#use rs232(baud=9600, xmit=PIN_C6, invert)  //serial port output pin & baud rate

//run photoresistor signal wire to pin AN0
//connect LED/resistor to pin RB7

void main(){
  int16 photo=0; //16 bit integer, safer than using int because
            //int is only 8 bit which might lead to overflow problems for add, multiply

  setup_adc(ADC_CLOCK_INTERNAL); //configure analog to digiral converter
  setup_adc_ports(ALL_ANALOG);   //set pins AN0-AN7 to analog (can read values from 0-255 instead of just 0,1)

  while(true){ //loop forever
    set_adc_channel(0);//set the pic to read from AN0
    delay_us(20);//delay 20 microseconds to allow PIC to switch to analog channel 0
    photo=read_adc(); //read input from pin AN0: 0<=photo<=255

     //turn on LED when input > 127, else turn off LED
     //Put finger over photoresistor & take it off to see LED turn on/off
     //127 may not be the actual value that separates light from dark, so try different values
```

```
    if(photo > 127){
      output_high(PIN_B7);
    }
    else{
      output_low(PIN_B7);
    }
  }
}
```

Code for multiple photoresistors

```
//all these # below set up the PIC
#include <16F877A.h>
#device adc=8
#FUSES NOWDT     //No Watch Dog Timer
#FUSES HS        //Highspeed Osc > 4mhz
#FUSES PUT       //Power Up Timer
#FUSES NOPROTECT  //Code not protected from reading
#FUSES NODEBUG    //No Debug mode for ICD
#FUSES NOBROWNOUT //No brownout reset
#FUSES NOLVP      //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD      //No EE protection
#use delay(clock=20000000) //crystal oscillator at 20000000 hertz
#use rs232(baud=9600, xmit=PIN_C6, invert)  //serial port output pin & baud rate

//read input from 3 photoresistors
//run photoresistor signal wires to pin AN0, AN1, AN2

void main(){
  int16 photo0=0; //16 bit integer, safer than using int
            //int is only 8 bit which might lead to overflow problems for add, multiply
  int16 photo1=0;
  int16 photo2=0;

  setup_adc(ADC_CLOCK_INTERNAL); //configure analog to digiral converter
  setup_adc_ports(ALL_ANALOG);   //set pins AN0-AN7 to analog (can read values from 0-255 instead of just 0,1)

  while(true){ //loop forever
    set_adc_channel(0);//set the pic to read from AN0
    delay_us(20);//delay 20 microseconds to allow PIC to switch to analog channel 0
    photo0=read_adc(); //read input from pin AN0: 0<=photo<=255

    set_adc_channel(1);//set the pic to read from AN1
    delay_us(20);
    photo1=read_adc();

    set_adc_channel(2);  //set the pic to read from AN2
    delay_us(20);
    photo2 = read_adc();

    //You could add 3 LEDs and turn them on if photo0/1/2 > 127
    //just as with code for single photoresistor
  }
}
```

Analog Input (Photoresistor, Sonar sensor, IR sensor, …)

Materials & Circuit

Depends on the device. For example, a sonar sensor might have 3 wires: power, ground & signal. You would connect the signal wire to one of pins AN0-AN7. See the device's datasheet for help.

Code

```
//the rest of the code is identical to code for photoresistor input, except here we call it 'sonar'
int16 sonar = 0;
set_adc_channel(1);//set the pic to read from AN1
delay_us(20);
sonar=read_adc();
```

Notes

You can use the photoresistor tutorial code to read any sort of analog input device, as long as the device is designed so you can run a signal wire to the input pin(s).
Remember, only the AN0-AN7 pins allow analog input, and you have to call setup_adc functions & set_adc_channel as in photoresistor tutorial.
Roughly, PIC will read 0 volts as 0, and 5 volts as 255.

Digital Input (Switches)

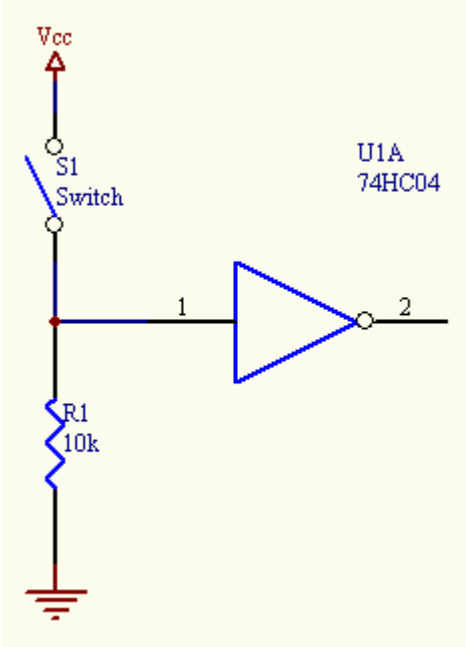
Materials

Circuit from "Introduction to the PIC16F877A"
LED
Switch
1K ohm resistor
10 or 47K ohm resistor
Wire

Circuit

Pin RB7 to 1Kohm resistor. Resistor to LED. LED to ground.
Pin RD1 to 10/47Kohm resistor. Resistor to ground.
Pin RD1 to switch. Switch to power.

This circuit has a "pull-down resistor". When the switch is closed, the PIC reads 5 volts and when it is open it reads 0 volts. If we had no pull down resistor then the pin would be in a "floating" state when the switch was open, meaning that the voltage can fluctuate. We need to connect the pin to ground so the PIC always reads 0 when the switch is open. If there was a wire connected to ground instead of a resistor, it would create a short that burns up the circuit.



Code

```
#include <16F877A.h>
#device adc=8
#FUSES NOWDT     //No Watch Dog Timer
#FUSES HS        //Highspeed Osc > 4mhz
#FUSES PUT       //Power Up Timer
#FUSES NOPROTECT  //Code not protected from reading
#FUSES NODEBUG    //No Debug mode for ICD
#FUSES NOBROWNOUT //No brownout reset
```

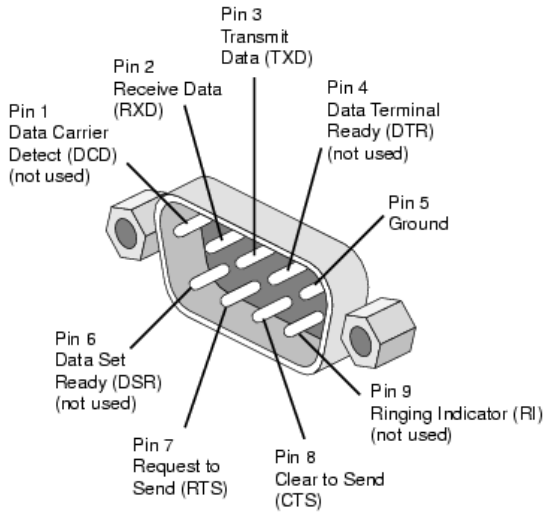
```
#FUSES NOLVP      //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD      //No EE protection
#use delay(clock=20000000)          // Sets crystal oscillator at 20 megahertz
#use rs232(baud=9600, xmit=PIN_C6, invert) // serial port output & baud rate

//close switch to see LED turn on
//open switch to see LED turn off

//if the pin is low (0 volts) x = 0, or FALSE
//if the pin is high (5 volts) x = 1, or TRUE

void main() {
  int x = 0;

  while(true){
    x = input(PIN_D1);
    if(x==1){
      output_high(PIN_B7);
    }
    else{
      output_low(PIN_B7);
    }
  }
}
```

Output messages to computer screen

I have used 2 methods.

1. PICkit 2 Development Programmer/Debugger. See their instructions in the program or website.

2. Serial port. I used HyperTerminal to read text from the serial port & display on screen, but other programs can do that too. Make sure the baud rate is 9600 bits / sec, just like in the PIC code: `#use rs232(baud=9600, xmit=PIN_C6, invert)`



Serial Port Steps (if using HyperTerminal)

1. Wire serial port pin 2 to pin TX (same as RC6) on the PIC.
2. Wire serial port pin 5 to ground.
3. Connect serial port to computer. Use a serial-USB adapter if computer has no serial port.
4. Open HyperTerminal.
5. Enter any name, select the icon that says “MCI”
6. Connect using COM5/COM13/etc. If there are multiple COMs: pull out the USB, reopen HyperTerminal, & see which COM has vanished. That’s the one you want.
7. Set bits per second = 9600.
8. HyperTerminal is set up. It will keep reading input until you close the program.

9. Load code onto PIC
10. Hit switch to turn on PIC

Code

```
#include <16F877A.h>
#device adc=8
#FUSES NOWDT     //No Watch Dog Timer
#FUSES HS        //Highspeed Osc > 4mhz
#FUSES PUT       //Power Up Timer
#FUSES NOPROTECT  //Code not protected from reading
#FUSES NODEBUG    //No Debug mode for ICD
#FUSES NOBROWNOUT //No brownout reset
#FUSES NOLVP      //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD      //No EE protection
#use delay(clock=20000000)          // Sets crystal oscillator at 20 megahertz
#use rs232(baud=9600, xmit=PIN_C6, invert) //Sets up serial port output pin & baud rate

void main(){
  int x = 0;
  while(true){
    x = x + 1;

    //This is an ordinary C language printf statement that will display on the screen of your PC.
    //But, you need to open a special program to read serial port input, like HyperTerminal.
    //Make sure the baud rate of the program matches this code's baud rate (9600 bits / second)

    printf("hello, x=%d\r\n",x); //send this text to serial port

    delay_ms(100); //wait 100 milliseconds
  }
}
```

Servo Motor Output

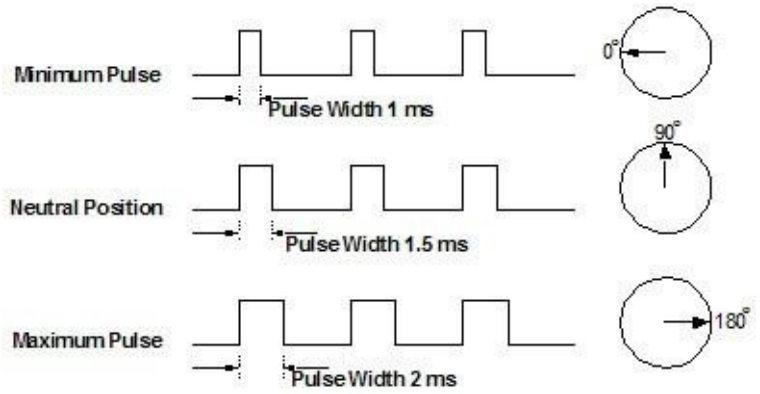
Materials

Circuit from "Introduction to the PIC16F877A"
Servo Motor (this tutorial tested on Parallax Standard Servo)
Wire

Circuit

See servo motor datasheet. Most likely it will be

1. Servo power wire to power
2. Servo ground wire to ground
3. Servo signal wire to pin RD2. (could be any I/O pin on the PIC, just change the code)



Code

```
#include <16F877A.h>
```

```
#device adc=8
#FUSES NOWDT     //No Watch Dog Timer
#FUSES HS        //Highspeed Osc > 4mhz
#FUSES PUT       //Power Up Timer
#FUSES NOPROTECT  //Code not protected from reading
#FUSES NODEBUG    //No Debug mode for ICD
#FUSES NOBROWNOUT //No brownout reset
#FUSES NOLVP      //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD      //No EE protection
#use delay(clock=20000000)           // Sets crystal oscillator at 20 megahertz
#use rs232(baud=9600, xmit=PIN_C6, invert) //Sets up serial port output pin & baud rate

//pin RD2 wired to servo
//servo connected to power, ground, and the signal wire from the PIC

//this program steps the Parallax standard servo slowly from 0 to 180 degrees,
//then rushes back to 0 degrees to restart.

//for code readability, could use #define SERVO_PIN PIN_D2 --> output_high(SERVO_PIN);

void main(){
  int16 pulse_width = 1000;
  int i;

  while(true){
     //send short pulse to servo indicating which angle it should move to.
     //for example, for one type of servo, 1000us=1ms indicates 0 degrees,
     //while 2000us=2ms indicates 180 degrees.
     output_high(PIN_D2);
     delay_us(pulse_width); //sending 5 volts to servo for pulse_width microseconds
     output_low(PIN_D2);
     delay_ms(20); //wait 20 milliseconds - refresh cycle of typical servos

     pulse_width = pulse_width + 1; //each time, servo is instructed to move a little farther
     if(pulse_width == 2001){ //if servo reached angle 180, reset: it will rush back to angle 0
       pulse_width = 1000;
     }

     /*
     If want servo to go to an angle & stay there, need to send the same pulse several times. (50 is good)
     If only send 1 pulse, the motor won't get all the way there, and it will stop, waiting for
     another pulse.  Example below shows how to move servo to 90 degrees.
     for(i=1;i<=50; i++){
       output_high(PIN_D2);
       delay_us(1500); //want servo to move to 90 degrees.
       output_low(PIN_D2);
       delay_ms(20);
     }
     */
  }
}
```

Notes

Other servos may have different pulse width requirements. And, they might go only 90 degrees, or up to 360 degrees. There are also issues where servos draw too much current or waste heat, which is beyond the scope of this tutorial.

DC Motor Output

Normally, you will use an H bridge for motor control. There are many internet articles about H bridges if you want to learn how they work.

Materials

Circuit from “Introduction to the PIC16F877A”
DC motor
H bridge chip
Wire

Circuit

Depends on the type of H bridge. I have used an old SN754410 quadruple half-h driver to run 2 motors at once. I also used a custom H bridge circuit board designed by club president Don Crouse. See the data sheet.

Important: whenever you change the direction, you should set all motor signal lines to 0 (output_low). This prevents short circuits.

Code & Notes

I will not include an entire program here, because each H bridge model has different rules for motor control. The datasheet will tell you how to set the signal lines.

There are two major types of output signals on most H bridges.

The first signal turns the motor on or off. It is often called the "Enable Pin". Whenever you want to turn the motor on, respectively off, type `output_high(MOTOR_ENABLE_PIN);` and `output_low(MOTOR_ENABLE_PIN);`

The second type are the signals indicating which direction the motor should turn, whether the motor free runs, and whether the motor brakes. Send 0 or 1 to those control signals: `output_high(MOTOR_DIRECTION_PIN_1)`, for example.

A simple illustration:

Direction Pin	**Enable Pin**	**Results**
0 (output_low)	1 (output_high)	One direction
1 (output_high)	1 (output_high)	Other direction
X (doesn't matter)	0 (output_low)	Free run (slow down & stop)

Some H bridges will only let you set the motor direction. In that case, you brake by reversing the motor direction for a short time. Beware: the current draw can spike if you do this, and the PIC might black out. But sometimes sudden direction reversal is OK; you will have to see for yourself.

If you have set the signals to make the motor go left, it will keep going left until you explicitly tell it to stop (or go right).

It is a good idea to write functions like "forward", "backward", "left", "right", "stop". That way you can just call the function & not have to worry about the details.

Below are code excerpts from motor control for SN754410 H bridge, controlling 2 motors. Your motor control code will NOT be the same, because you will use a different H bridge or have different needs. But it will be similar to this.

```
#define LEFT_CONTROL_2A  PIN_C2  //2A
#define LEFT_CONTROL_1A  PIN_C1  //1A
#define LEFT_MOTOR       PIN_D1  //1,2EN
#define RIGHT_MOTOR      PIN_D2  //3,4EN
#define RIGHT_CONTROL_4A PIN_D5  //4A
#define RIGHT_CONTROL_3A PIN_D4 //3A
int going_forward;  //current motion
int going_backward;
int going_left;
int going_right;
```

```
int going_hard_left;
int going_hard_right;
…
…
//brake by reversing motor direction, then turn motors off
void halt(){
  if (going_forward)
    backward();
  else if (going_backward)
    forward();
  else if (going_left)
    right();
  else if (going_right)
    left();
  else if (going_hard_left)
    hard_right();
  else if (going_hard_right)
    hard_left();
  delay_ms(50); //reverse direction for 50 ms, enough to make robot stop
                //but not long enough to make it start moving in opposite direction
  resetMotorControl();
}

void forward(){
  resetMotorControl();
  output_high(LEFT_CONTROL_1A);
  output_high(RIGHT_CONTROL_3A);
  output_high(LEFT_MOTOR);
  output_high(RIGHT_MOTOR);
  going_forward=1;
}

void backward(){
  resetMotorControl();
  output_high(LEFT_CONTROL_2A);
  output_high(RIGHT_CONTROL_4A);
  output_high(LEFT_MOTOR);
  output_high(RIGHT_MOTOR);
  going_backward=1;
}

//run only 1 motor for gentler turn
void right(){
  resetMotorControl();
  output_high(RIGHT_CONTROL_3A);
  output_high(RIGHT_MOTOR);
  going_right=1;
}

//run only 1 motor for gentler turn
void left(){
  resetMotorControl();
  output_high(LEFT_CONTROL_1A);
  output_high(LEFT_MOTOR);
  going_left=1;
}

//run one motor forward and one backward for sharper turn
void hard_left(){
  resetMotorControl();
  output_high(LEFT_CONTROL_1A);
  output_high(RIGHT_CONTROL_4A);
  output_high(LEFT_MOTOR);
  output_high(RIGHT_MOTOR);
  going_hard_left=1;
}

//run one motor forward and one backward for sharper turn
void hard_right(){
  resetMotorControl();
  output_high(LEFT_CONTROL_2A);
  output_high(RIGHT_CONTROL_3A);
  output_high(LEFT_MOTOR);
```

```
  output_high(RIGHT_MOTOR);
  going_hard_right=1;
}

//Set all pins to H-bridge low to prevent short circuits
//Reset "current motion" flags
void resetMotorControl(){
  output_low(LEFT_MOTOR);
  output_low(RIGHT_MOTOR);
  output_low(LEFT_CONTROL_1A);
  output_low(LEFT_CONTROL_2A);
  output_low(RIGHT_CONTROL_3A);
  output_low(RIGHT_CONTROL_4A);
  going_forward=0;
  going_backward=0;
  going_left=0;
  going_right=0;
  going_hard_left = 0;
  going_hard_right = 0;
  delay_us(20);
}
```

Special Topic: PWM

Now you know how to run the motor at full speed. But a situation might arise where you want half speed, or ¾ speed, etc. In that case, you can send a PWM signal to the H bridge enable pin. There are many internet articles about PWM if you want to learn how it works.

To use PWM in motor control, replace `output_high(MOTOR_ENABLE_PIN)` with `set_pwm1_duty(x)` . Make sure that the "enable pin" wire goes into the PIC pin CCP1 or CCP2. Those are the only pins that can have PWM signal.

PWM code excerpt:

```
  setup_timer_2(T2_DIV_BY_4,255,1);
  setup_ccp1(CCP_PWM); //set pin CCP1 as a PWM pin instead of a regular I/O pin
  setup_ccp2(CCP_PWM); //set pin CCP2 as a PWM pin instead of a regular I/O pin
  while(true){
    //if setting duty cycle (set_pwm1_duty), only give values between 0-255
    //0 means the PWM signal is always at 0.  128 means it is 1 half the time & 0 the other half
    //leading to an averaged "1/2 power" signal.  255 means it is 1 all the time, motor full speed
    set_pwm1_duty(255);  //enable pin for motor 1: PIC pin CCP1
    set_pwm2_duty(255);  //enable pin for motor 2: PIC pin CCP2
    delay_ms(3000); //full speed for 3 seconds
    set_pwm1_duty(128);
    set_pwm2_duty(128);
    delay_ms(3000); //half speed for 3 seconds
    set_pwm1_duty(0);
    set_pwm2_duty(0);
    delay_ms(3000); //motors off for 3 seconds
  }

//Use PWM in your DC motor control code to change the speed that the robot goes forward/backward/left/right
//call forward(255); to go forward at full speed.  Call forward(128); to go forward at half speed.
void forward(int speed){
  resetMotorControl();
  output_high(LEFT_CONTROL_1A);
  output_high(RIGHT_CONTROL_3A);
  set_pwm1_duty(speed); //wire from PIC pin CCP1 to right motor enable pin
  set_pwm2_duty(speed); //wire from PIC pin CCP2 to left motor enable pin connected to
  going_forward=1;
}
```

Notes

If you want to use higher voltage DC motors, or run motors at high speeds, you might have to construct a special circuit to deal with the extra current draw, heat, high voltage batteries, etc. DC motors have their own set of issues to deal with, which is beyond the scope of this tutorial.