Controlling an LED

Use the microcontroller to control a light emitting diode (LED).

Site: iCODE
Course: Machine Science Guides (Arduino Version)
Book: Controlling an LED
Printed by: Guest User
Date: Thursday, August 7, 2014, 02:29 PM

Contents

About the Introductory Activities

By now, you have set up the microcontroller on the breadboard and learned how to transfer code from your computer to the chip. In each of the following activities, you will add basic electronic components to the breadboard--an LED, a button switch, a speaker, a light sensor, an LCD, and a temperature sensor--and then write a few lines of code to get them working. These activities are simple, but there's a lot to learn here, so take your time and read carefully. The skills you learn in this section will be very important later on. Figure 1 shows some electronic devices that have LEDs, buttons, speakers, and sensors.

Figure 1. Electronic devices with LEDs, buttons, speakers, sensors, and LCDs.

About Inputs and Outputs

Before you get started, take a close look at the microcontroller. A photo of the chip, taken out of the breadboard, is shown in Figure 2.

Figure 2. Microcontroller.

The microcontroller may seem like a mysterious device, but actually, it interacts with other electronic components on the breadboard in only two, very simple ways:

#1 It can set the voltage on any pin HIGH (5 volts) or LOW (0 volts).
#2 It can detect whether the voltage at any pin is HIGH (5 volts) or LOW (0 volts).

It also knows a few other tricks, but that's basically all the chip does: set the voltage on certain pins, and detect the voltage on others. Coupled with the ability to store and execute code, these two simple abilities make the microcontroller a very powerful device.

Consider #1 first. If you connect an output device, such as an LED, to a particular microcontroller pin, the microcontroller can turn the device on or off by raising and lowering the voltage on that pin. Remember: electric current flows whenever there is a voltage difference between one side of a component and the other. In similar fashion, the chip can control more complex output devices, like speakers, motors, and the LCD, by raising and lowering the voltage on its pins in precisely timed patterns.

Now consider #2. The microcontroller can detect whether the voltage at any of its pins is 5 volts or 0 volts. Imagine that you connect a simple input device, such as a button switch, to a particular microcontroller pin, and set up the circuit so that the button connects that microcontroller pin to power (5 volts) when it is pressed and ground (0 volts) when it is not pressed. The microcontroller can tell whether the button is pressed or not pressed, simply by detecting whether the voltage at the pin is high or low.

That, in a nutshell, is how all modern electronic devices work. Think of all the times you have seen an LED light on your computer keyboard, read text on an LCD, pressed a cell phone key, or hit a game controller button. Inside those devices, there were computer chips, raising and lowering the voltage on some pins, and detecting high and low voltage on others.

The microcontroller's code is simply the logic that connects the inputs and the outputs.

About C Code

A C program is a series of instructions that tell the microcontroller what to do. The microcontroller executes these instructions in sequence from top to bottom. Statements can be combined into larger groups, called functions, which enable the microcontroller to perform more complex operations. In order for the compiler to understand your instructions, the program must always be "wrapped" with a few specific lines of code. Figure 3 shows a simple C program, with these important "wrapping" parts identified. You don't have to write this program yet; just look at it and try to understand its basic structure.

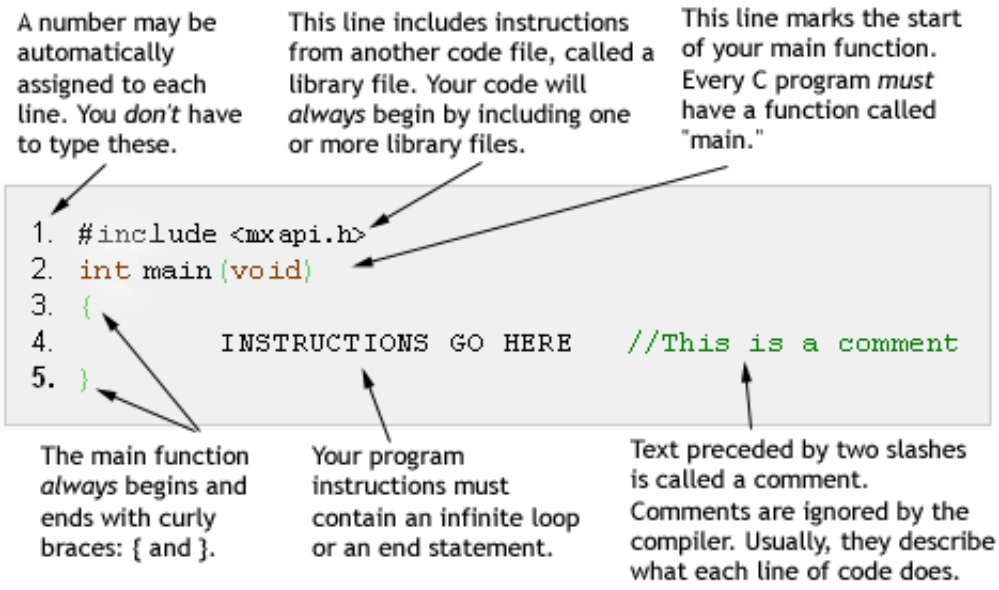

Figure 3. Anatomy of a C program.

If you don't understand every line of the program, don't worry. Just remember four things:

- Every Machine Science code file should start with **#include <mxapi.h>**
- Every C program must have a main function set up with **int main(void)**
- Every main function must be contained in curly braces: { and }
- Every main function must contain an **infinite loop**. Since microcontrollers have no operating system, no program may reach the end of the main function.

Connecting the LED to the Chip

Using the schematic in Figure 4 as a guide, insert the two pins of an LED into the breadboard. You can position the LED in any free space on the breadboard, but the two pins should not be in the same numbered row of holes. Use a flexible jump wire to connect the longer power pin to Port B0 on the microcontroller, and use an 1000-Ohm resistor to connect the shorter ground pin to ground.

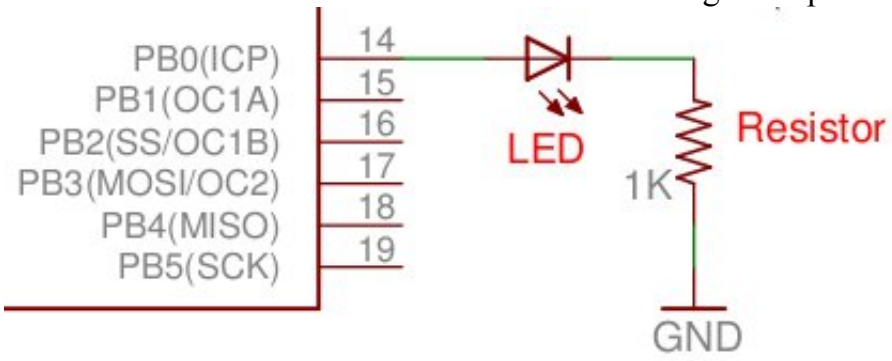

Figure 4. LED circuit.

Turning the LED On

OK. You are ready to write a simple program to turn on the LED. The main function has three statements: the first statement sets up Port B0 (the pin connected to the LED) as an output pin, meaning electricity can flow out of it; the second statement sets the value of the Port B0 pin high, meaning its voltage is 5 volts; and the third statement ends the program.

1. Open the Arduino IDE.
2. Type the following code in the window. NOTE: You don't have to type the comments, but it is good practice to do so.

```
#include <mxapi.h>

int main(void)
{
   output_pin(PORT_B0);    //Set up Port B0 as an output pin
   high_pin(PORT_B0);      //Set the Port B0 pin high
   while(1==1);            //End the program
}
```

3. Save your code file as ledon.c.
4. Compile and test your code.

Your LED should now light and stay lit. Electricity is flowing from the Port B0 pin, which has a value of 5 volts, through the LED and the resistor to ground, which has a value of 0 volts.

Turning the LED Off

Turning an LED off is as simple as turning it on. You simply set the pin it is connected to low (0 volts), instead of high (5 volts). This way, the voltage at the pin is the same as the voltage at ground, and no electricity flows through the circuit.

1. Rename your code file ledoff.c.
2. Modify your code file, as follows:

```
#include <mxapi.h>

int main(void)
{
   output_pin(PORT_B0);  //Set up Port B0 as an output pin
   low_pin(PORT_B0);     //Set the Port B0 pin low
   while(1==1);          //End the program
}
```

3. Compile and test your new code.

Turning the LED On and Off

So now you know how to turn an LED on and off. How about doing both of these things these things in one program? To do this, you will need to introduce a new instruction in your code, called a delay. A delay statement tells the microcontroller to wait for a specific period of time before executing the next instruction. Depending on what type of delay statement you use, these intervals are measured in milliseconds (1/1000ths of a second) or microseconds (1/1,000,000ths of a second), as shown below.

delay_ms statements wait in milliseconds (1ms = 0.001 second)
delay_us statements wait in microseconds (1us = 0.000001 second)

1. Rename your code file ledonoff.c.
2. Modify your code file, as follows:

```
#include <mxapi.h>

int main(void)
{
   output_pin(PORT_B0);  //Set up Port B0 as an output pin
   high_pin(PORT_B0);    //Set the Port B0 pin high
   delay_ms(500);        //Wait for 500 milliseconds
   low_pin(PORT_B0);     //Set the Port B0 pin low
   delay_ms(500);        //Wait for 500 milliseconds
   while(1==1);          //End the program
}
```

3. Compile and test your new code.

Programming Challenge

Modify your code file so that the LED lights for 1 second before turning off. Then, modify your code file so that the LED lights for 0.1 seconds before turning off.

Making the LED Flash

With just a few more lines of code, you can make the LED turn on and off in a flashing pattern. To do this, you need to introduce a new structure into your code, called a while...loop. A while...loop is a set of instructions that is repeated over and over. It's called a while... loop, because the instructions are repeated while a certain condition is true. In this case, you will use a condition that is always true, so the loop will repeat indefinitely (or at least until you switch off the power!)

So how can you specify a condition that's always true? In C programming, this is done with a very simple mathematical equation. For example, it's always true that one is equal to one. In C, this is expressed as 1==1. Note that a double equals sign is placed between the two numbers (it's 1==1, not 1=1). You will learn more about the double equals sign later; for now, just make sure you use a double equals (==) in the statement setting up your while...loop. Everywhere else, use a single equals (=).

1. Rename your code file ledflash.c.
2. Modify your code file, as follows:

```
#include <mxapi.h>

int main(void)
{
   output_pin(PORT_B0);  //Set up Port B0 as an output pin
   while(1==1)           //Set up the while loop
   {                     //Start the while loop
      high_pin(PORT_B0); //Set the Port B0 pin high
      delay_ms(500);     //Wait for 500 milliseconds
      low_pin(PORT_B0);  //Set the Port B0 pin low
      delay_ms(500);     //Wait for 500 milliseconds
   }                     //End the while loop
}
```

3. Compile and test your new code.

Programming Challenge

Change your delay statements to make the LED flash even faster. How fast can you go before your eyes can't even tell that it's flashing? Change the condition for your while...loop to something else that's always true, such as 5==5, or 9>6. Does this affect your program at all?