USC Viterbi School of Engineering

Ming Hsieh Department of Electrical Engineering

EE 109L - Introduction to Embedded Systems

Fall 2014 Lab Assignment #0
Installation of the Arduino Tool Chain

1 Introduction

The purpose of this lab assignment is to help students install the software that will be used throughout the semester to program the Arduino microcontroller development board. Most of the following lab assignments can not be completed until the programming environment has been successfully set up on whatever computer a student plans to use.

The Arduino company that makes the microcontroller development boards we will be using also provides a software package for programming the boards known as the "Arduino IDE". However, for this class we won't be writing programs within that environment. The Arduino IDE does a number of things that makes programming the boards very simple, but our goal in EE 109 is to learn how these things are done, not just to do them.

The software package we will be using is "avr-gcc", a version of the gcc C compiler that creates code for the Atmel AVR processor used on the Arduino boards. The collection of programming tools included in avr-gcc (compiler, linker, downloader, etc.) is known as a "tool chain". The avr-gcc tool chain is command-line based and is used via a program like Terminal on a Mac or the Windows Command Prompt. If you are not comfortable using that type of programming environment, you'll have to find some other way to program the chips.

2 Installing the Development Software

2.1 Mac OS X Installation

The avr-gcc software for use on a Mac can be obtained from the instructor or downloaded from a couple of web sites. It should run on any Intel-based Mac OS X system.

The Apple Xcode developer tools are not supposed to be required in order to run the avr-gcc software, however we have only used the avr-gcc software on systems that have already had the Xcode tools installed so we can't confirm that they aren't needed. If you wish to have the Xcode tools on your system it can be downloaded for free from Apple's App Store. It may also be necessary to install the Xcode command lines tools. On Mac OS X 10.9 (Mavericks) this is done after Xcode has been installed by using the Terminal application to issue the command "`xcode-select --install`". Older versions of Xcode on previous versions of OS X may require either a separate download to install the command lines tools, or they can be installed from the Preferences panel of the Xcode program.

To install the avr-gcc software:

1. The avr-gcc software can be downloaded from "`http://www.obdev.at/products/crosspack`". Obtain the file "`CrossPack-AVR-20131216.dmg`" or a more recent one. Double click on it to open the disk image and double click on the "`CrossPack-AVR.pkg`" file and follow the instructions to install the cross compiler software.

2. Open a Terminal window and type the command "`printenv PATH`" to show the search path used for finding executable programs. It should contain the directory "`/usr/local/CrossPack-AVR/bin`".

3. Type the command "`which avrdude`" to confirm that the `avrdude` program for downloading programs to the micros has been installed.

You should now be ready to run the avr-gcc development software to program a microcontroller.

2.2 Windows Installation

The avr-gcc software for use on a Windows system can be obtained from the instructor or downloaded from a couple of web sites. It has been run on Windows 7 systems and probably runs on other versions.

To install the avr-gcc software:

1. The avr-gcc software can be downloaded from "`http://sourceforge.net/projects/winavr/`". Download the file "`WinAVR-20100110-install.exe`" or a more recent one. Double click on it to start the installation. Follow the instructions and use the default settings and locations.

2. Download the Arduino IDE software from "http://arduino.cc/en/main/software". We won't be using this to program the Arduinos, but it is handy to have installed for checking on which I/O port the Arduino is connected to. Install the package using the default options. At some point it may ask you to install the USB driver. Choose "Install". When done, you can click "Complete".

3. Open a Command Prompt window and type the command "`path`" to show the search path used for finding executable programs. It should contain the directories "`C:\WinAVR-20100110\bin`" and "`C:\WinAVR-20100110\utils`.

4. Type the command "`where avrdude`" to confirm that the `avrdude` program for downloading programs to the micros has been installed.

5. Connect one of the Arduino Unos to the computer using a USB cable. Windows will probably try to install a driver and this sometimes fails or it installs the wrong driver. Open the Device Manager, find the Arduino device and double click on it. On the Properties windows that come up, click on the Driver tab and then on the Update Driver button. On the next windows, click on the Browse button and, assuming you installed the WinAVR software in the default location, search for the directory "`C:\WinAVR-20100110\utils\libusb\bin`" and tell it to install the driver from that directory. With any luck the installation should succeed.

6. Find the port that the Arduino is connected to by running the Arduino IDE program. From the "Tools" menu, select "Serial Ports" and it should be listed there as "`COM3`" or something like that. Make a note of the port since you will need it later.

You should now be ready to run the avr-gcc development software to program a microcontroller.

2.3 Linux Installation

For Debian and Ubuntu Linux the following packages need to be installed.

```
gcc-avr
binutils-avr
gdb-avr
avr-libc
avrdude
```

The installation can be done with the command

```
sudo apt-get install gcc-avr binutils-avr gdb-avr avr-libc avrdude
```

This may also cause several other packages to be installed. Check to see if the "`make`" program is present on the system. If not also do "`sudo apt-get install make`" to install it.

3 Create a New Project

The easiest way to create a new project is to start with a couple of template files from the class web site. Create a new folder with a name for the new project. It is best to not use spaces in the folder names for the project folder nor any parent folders in which you create the project folder. Spaced names are great when navigating folders in a GUI, but not so great (i.e. very painful) when using command line interfaces.

From the class web site download the Zip file containing the the files "`lab0.c`" and "`Makefile`", unzip them and put them both in the project folder.

The C file is a very simple program to blink an LED on the Arduino board. The goal here it to just be able to program the board so this should be sufficient and the file should not have to be changed. However the file `Makefile` will probably have to be edited to make it work with your computer.

4 Editing Files

In order to edit the text files associated with your project you will need a text editing program on your system. There are a large number of these available for free and it's pretty much personal preference as to which one to use. The main requirement is that it be able to edit a file of ASCII text characters and not turn it into something like RTF (Rich Text Format) or some other word processing document type.

For Unix-type systems (Mac OS X, Linux, etc.) the most common command-line interface text editors are probably "`emacs`", "`vi`" or "`vim`" (improved `vi`). Some of these may already be installed on the system and all are available in various forms from several Internet sites. There are also numerous GUI type text editors like TextWranger and Aquamacs that will also work.

For Windows, the easiest one to find is probably "Notepad++". One requirement of editing text files on Windows is that the editor be able to save files without adding a filename extension like "`.txt`". If the editor insists on adding the extension it is usually necessary to then use a Command Prompt window to rename the file and remove the extension.

5 The Makefile

The file `Makefile` is the key to correct compiling and downloading of the Arduino programs. It contains all the information and commands needed to compile the program and produce binary data that can be downloaded into the ATmega328P processor on the Arduino Uno board. Once the proper information is in the `Makefile`, all the required program modules can be compiled and linked together by just typing the command "`make`". If a `Makefile` is copied from an existing project to a new one most of the information in it can usually be left unchanged, however a few lines at the top may have to be changed in order for the compiling, linking and downloading to work properly.

It's important to note that the name of the file must be "`Makefile`" (or "`makefile`"). On occasion your Mac or Windows machine may try to add an extension to the file and call it something like "`Makefile.txt`". This will prevent the `make` program from working. Some systems will add the extension, and then make the extension invisible so from the GUI the file looks like its name is `Makefile` but it's really not.

If the `make` command returns an error message about no Makefile found, use a Terminal or Command Prompt window to go to the project directory and the `ls` (Mac or Linux) or `dir` (Windows) command should list the files showing the full names. If necessary use the following command to fix the name.

```
rename makefile.txt makefile      <-- on Windows
mv Makefile.txt Makefile          <-- on Mac or Linux
```

Using your text editor, open the `Makefile` and look at the five lines at the top. These provide information to the compiler on what type of processor is being used. Regardless of where the `Makefile` came from, it's very important to make sure the correct information is at the top.

```
DEVICE     = atmega328p
CLOCK      = 16000000
PROGRAMMER = -c arduino -b 115200 -P <your_port_name>
OBJECTS    = lab0.o
```

```
FUSES       = -U hfuse:w:0xde:m -U lfuse:w:0xff:m -U efuse:w:0x05:m
```

The DEVICE line contains the name of the microcontroller and must be correct for the program to compile correctly. For the Arduino Uno this should be "atmega328p".

The CLOCK line tells the frequency of the clock being used in Hertz. This needs to be correct so any calls to delay functions will work and also for the communication between the your computer and the microcontroller. The Arduino Uno board has a 16MHz clock so this should say "16000000".

The PROGRAMMER line contains parameters used by the program that downloads the data to the microcontroller. The first two parameters tell the program that it's an Arduino on the other end of the cable, and what speed to download the data. However the parameter that comes after the -P is the name of the port used by your computer for the connection and this will be different depending on the type of system you are using.

For Macs: Use "/dev/tty.usbmodem*"
```
PROGRAMMER = -c arduino -b 115200 -P /dev/tty.usbmodem*
```
For Windows: Use the port name you found above using the Arduino IDE program (e.g. "COM3")
```
PROGRAMMER = -c arduino -b 115200 -P COM3
```
For Linux: Use "/dev/ttyACM0"
```
PROGRAMMER = -c arduino -b 115200 -P /dev/ttyACM0
```

The OBJECTS line lists all object modules that need to be linked together. If the program has been divided into several source files this line will contain more entries. For this lab, it should just list the one file: lab0.o.

The FUSE line contains parameters used to control various aspects of the microcontroller. The ones in the provided Makefile should be correct for the Uno board and should not have to be changed.

Make whatever changes are needed to the PROGRAMMER line to be correct for your system and save the file. As stated above make sure the editor didn't add an extension to the filename when saving it.

6 Download the Program

Once the Makefile is correct, try compiling the lab0.c program by typing the command "make". If the compile works you should see something like this on the screen with no reports of any compiling errors.

```
$ make
avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -c lab0.c -o lab0.o
avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -o main.elf lab0.o
rm -f main.hex
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
```

This says the lab0.c file was successfully compiled to create the lab0.o object file, and this was then used to create the main.elf file. Finally the main.elf was converted into main.hex which contains the binary data containing the program. This will be followed by some lines about how big the program is.

If it isn't connected already, hook one of the Arduino Unos to your system. Type the command "make flash" to download the data. The make program will run the program that downloads the data using the parameters you specified in the Makefile. If it works a bunch of stuff will fly by on the screen as it writes the data to the microcontroller and then verifies it. At the end you should see something like

```
avrdude: verifying ...
avrdude: 176 bytes of flash verified

avrdude: safemode: Fuses OK (H:00, E:00, L:00)

avrdude done.  Thank you.
```

At this point the new program will start executing in the Uno and you should see a small LED near the minus sign in the Arduino logo start to flash once per second.