ECE 375 Linux Guide

Spencer Moss, Cody Holliday

September 2017

1 Packages to Install

Assuming you are running a Linux distribution with a package manager such as `apt`, `dnf`, `yum`, or `pacman`, you will need two basic programs in order to assemble and program your ATmega128 chip on the ECE375 TekBots board.

1. `avrdude`: AVRDUDE is the standard program available on Linux/Unix systems for flashing programs onto AVR chips. This will be used to program your ATmega128 chip with your compiled code.
2. `avra`: avra is an open-source implementation of the AVR assembler that is included with AVR Studio. This program will be used to translate your assembly source code to AVR machine code.

On Fedora Linux, one might install these packages using DNF:
`sudo dnf install avra avrdude` or `sudo yum install avra avrdude`

On Ubuntu Linux (and Debian as well):
`sudo apt install avra avrdude`

On Arch Linux:
`sudo pacman -S avra avrdude`

Other distributions/package managers may not have these immediately available for use. The source code for these programs are hosted at the following locations as of writing:
`avrdude`: http://www.nongnu.org/avrdude/ `avra`: http://www.nongnu.org/avrdude/

Other packages that may be useful include:

1. `avrp`: An alternative Flash/EEPROM programmer to AVRDUDE.
2. `avr-binutils` (on Fedora): Cross Compiling GNU binutils targeted at AVR.
3. `avrdude-doc`: Documentation for `avrdude`.

Lastly, there are ports of GCC (including GCC-C++) and GDB to work with AVR code. These packages are usually named `avr-gcc` and `avr-gdb` respectively. Unfortunately, the GCC assembler uses a different syntax/format for things such as assembler include files and other assembler directives, which will make it difficult to do the labs as the `m128def.inc` file used throughout ECE 375 uses the `avra` assembler syntax, and would need to be translated to use for turning in source code in addition to lab code.

2 Assembling and Running Programs

In order to run Lab 1 of ECE 375 as an example, one must obtain the BasicBumpBot.asm file from the lab website. Also, one must obtain the file 'm128def.inc' from somewhere (it is available in Atmel Studio); here is a link to a copy hosted on GitHub.

With these two files, the assembly source may be translated into machine code and programmed with two simple commands:

```
avra BasicBumpBot.asm
```

From the above command, a hex file should be created in the current directory that can be moved into the ATmega128's flash program memory. The following `avrdude` command should program your ATmega128 board when connected properly to the USB ASP.

```
avrdude -c usbasp -p ATmega128 -U flash:w:BasicBumpBot.hex
```

Here, the flags are as follows:
`-c` specifies the type of hardware programmer to use (USB ASP is what comes in the lab kit.)

`-p` specifies the type of chip to be programmed (ATmega128 is the AVR MCU on the board.)

`-U` specifies a type of command for avrdude to execute; here, it is writing (w) the flash memory of the ATmega128 with your program (the .hex file).

A basic Makefile for compiling and programming the board may be found here. A more advanced one is here. Do note that using these requires that the `make` package is installed. Additionally, you will need to edit the `TARGET` variable within the Makefile to assemble and flash other lab source code onto the ATmega128 board.

3 Simulation

One benefit of using the GCC tools is that there exist a number of different hardware simulators for the AVR architecture, including `simavr`, `avrsim`, and `simulavr`. `avra` takes in assembly source code and produces the Intel HEX format binary file that gets written straight to the AVR's flash memory for programming. The GCC toolchain has an intermediate step of having an .elf file (Executable and Linkable Format) that allows for better debugging/simulation support. This missing step in `avra`'s assembler process is what makes it difficult to simulate ECE 375 programs on GNU/Linux systems.

While tools such as `simulavr` and `avrsim` allow one to hook up the extremely powerful GNU Debugger to a running AVR program and have functionality that rivals the simulator built into Atmel Studio, it is reccommended to stick to `avra` for compatibility's sake with the lab requirements. As such, setting up a full simulation environment for AVR under Linux is outside the scope of this document.