SE3910 Lab 1: Getting started with the development environment and the Beaglebone Black

Due: March 16, 2014

1 Introduction

This is your first lab with the Beaglebone. In this lab, you will learn a bit about how the Beaglebone works, as well as start some basic development with the system.

2 Lab Objectives

- Understand how to create an sd card image of an operating system
- Understand the concept of cross-compilation
- Understand how a makefile functions.
- Understand the operation of a cross compiler.

3 Laboratory Equipment Needed

- One Beaglebone Black embedded systems board
- One 10W 5 V power supply
- 2 ethernet networking cables

4 Introduction

In this lab, you are going to setup the Beagleboard and associated systems, as well as perform your first cross compilation exercise and run your first program on the Beagle.

4.1 Virtual Machine setup

The linux distribution you are using will execute using the VirtualBox virtual machine software. You will need to download this and install the VirtualBox software on your laptop. You will need to install the appropriate version of the software, which is 64 bit for the MSOE machines.

When you have completed installing Virtual box, you will need to download and install the linux virtual machine image from the instructor's website. The virtual machine will run xubuntu, a variant of the Ubuntu linux distribution. XUbuntu has been chosen due to its similarity with Ubuntu (which is used for much development) and its simpler windowing environment which improves the performance of the Virtual machine under windows.

Once the Virtual machine is downloaded and setup, log onto the operating system. The default user is se3910, and the default password is se3910. Install the guest additions software on the virtual machine, and create a shared folder between the windows workstation and the virtual machine. This shared folder allows you to share data between the two operating systems as well as reliably back up you work from the virtual machine. You can use any location that you would like for the share point. Useful shares might be the entire d drive of your laptop or the Box.com folder if you use Box.com to automatically back up and synchronize your data.

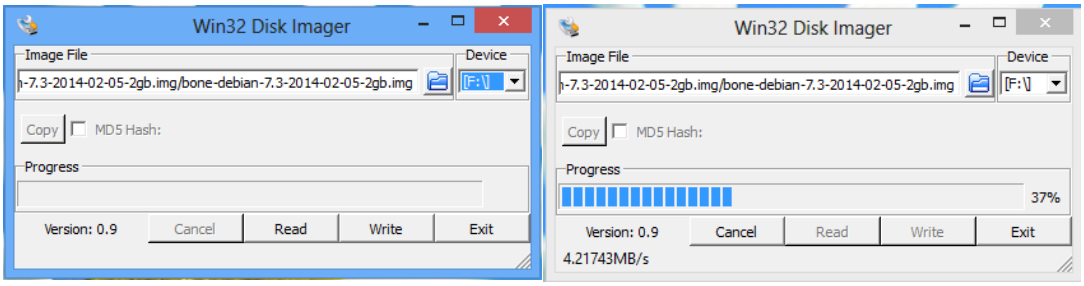
4.2 Creating your bootable flash image

After you have installed your virtual machine, you will need to make a bootable flash image for the Beaglebone Black. To do this, you will need at least one 8GB microSD card. (Note: While making the image, it might be wise to make two of them. That way, you will always have a backup image should you either loose the SD card or the SD card gets damaged during software development.)

The first step involves downloading the Win32DiskImager software. This software will allow you to burn an SD card from windows with a binary image. This is a simple program that can be executed without full installation on your machine.

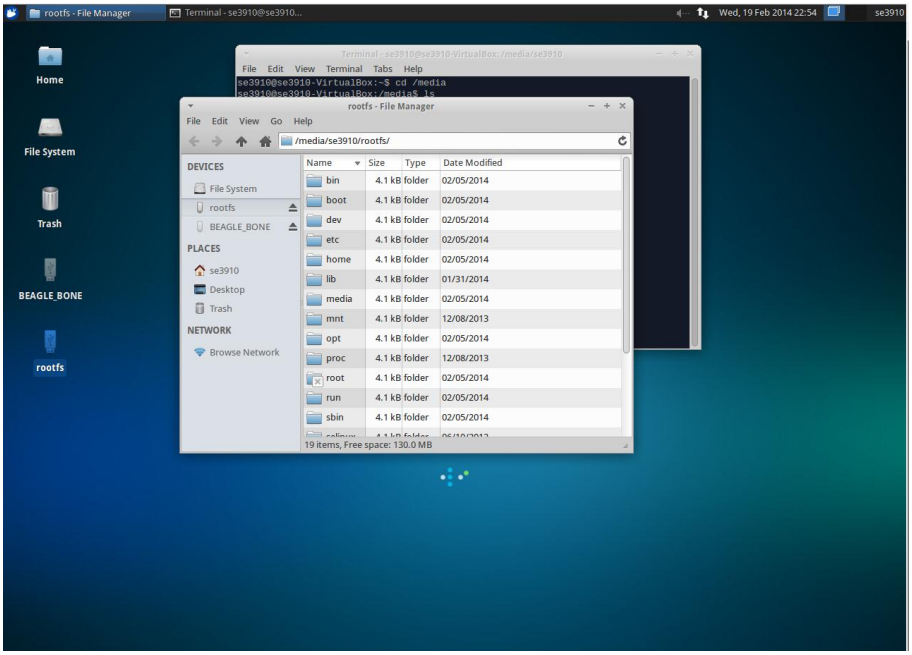
You also will need to download the Debian Operatign System for the Beaglebone. This operating system will be installed on the microsd card and allow the OS to boot. You will download a compressed version of the file. Once this is done, extract out the file with the extension .img.

To burn the image, first insert the SD Card into a MicroSD card adapter or a USB to micro SD card reader. Pay extreme attention to where the drive is mounted. IT IS HIGHLY RECCOMENDED THAT YOU REMOVE ANY AND ALL EXTRA STORAGE MEDIA FROM YOUR MACHINE. Start up the Win32Disk Imager software and select the image file that you want to burn as well as the destination drive. When you are ready, press write and the disk image will be written. Writing the image will take about 5 minutes.

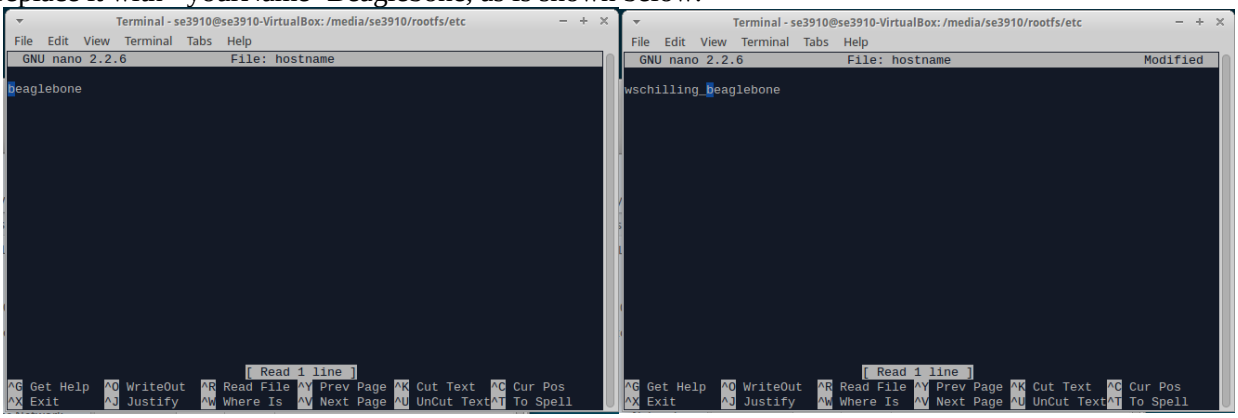


Once writing the image has completed, you will have a burned image of the operating system on the SD card. We will need to change the nane of your Beaglebone by modifying the host files on the sd card to have a custom name.

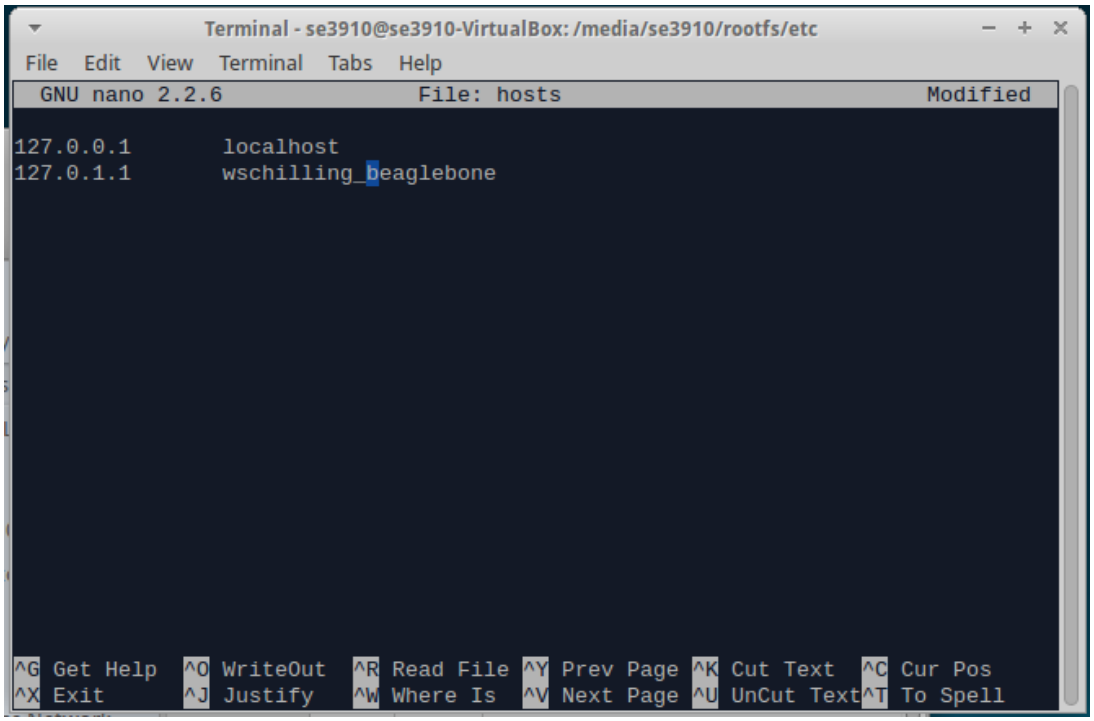
First, remove the card from your SD card reader slot. Next, start up the virtual machine. With the virtual machine running, select "devices" from the top of the Virtual Box menu, "USB Devices", and the name of the USB device on your machine. Now insert the card into the sd card reader. After a few seconds, two icons should appear on your virtual machine. Right click on the "rootfs" partition and selection "mount" to mount it into your file system. This should result in an image similar to that shown below.



Open up a terminal window and change to the directory /media/se3910/rootfs/etc. Open the file "hostname" file in the nano text editor. Find the line where the host name is defined as "Beaglebone" and replace it with <yourName>Beaglebone, as is shown below.



When this is done, edit the hosts file, again replacing the text Beaglebone with <yourName>Beaglebone.



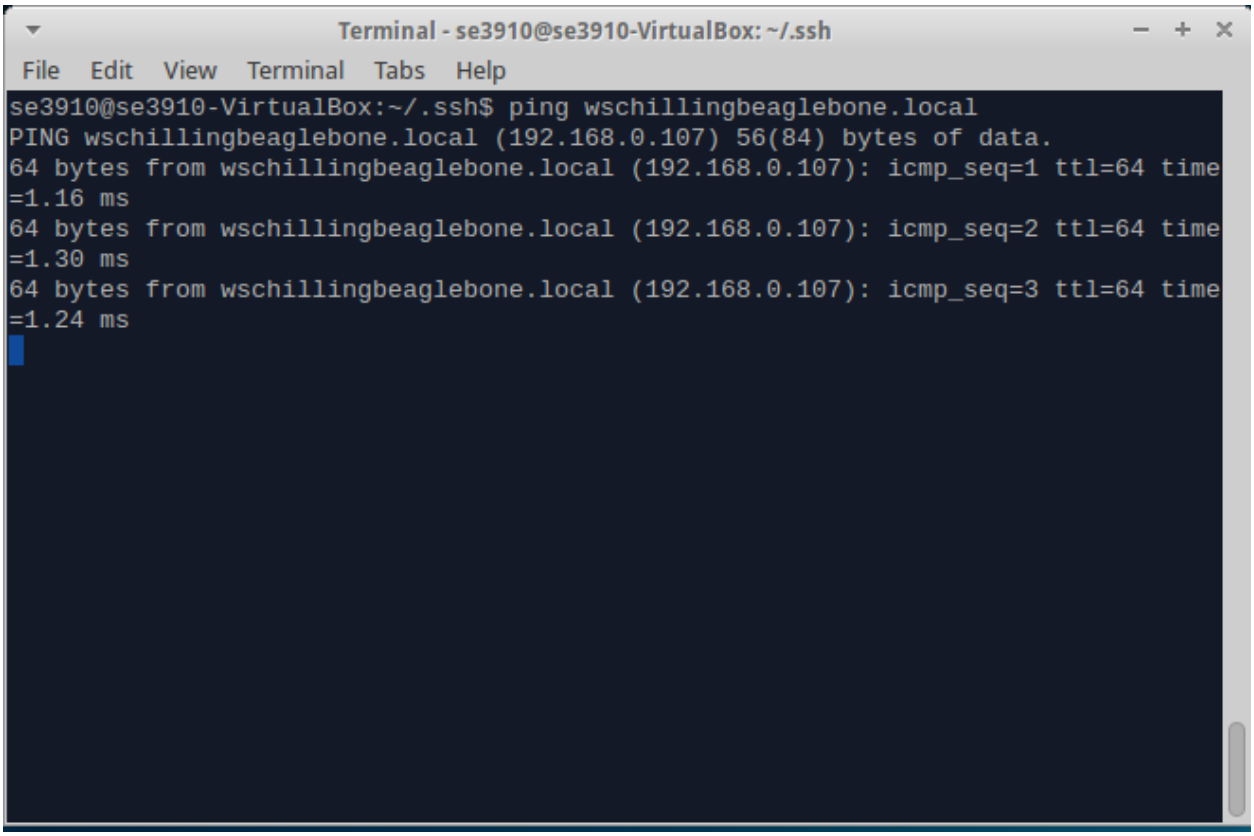
With this being done, exit out of the terminal window. Right click on the rootfs icon and select "eject volume". It is now safe to remove your SD card from the reader.

5 Booting the Beaglebone and finding it on the network.

With the card safely programming, take out your Beaglebone and put the card into the microSD card slot on the Beaglebone. Connect the Ethernet port to the router. While holding down the button labeled S2 (located above the micro sd card), plug the board in to the power supply. After a few seconds release the button. You should see the leds next to the network jack begin to flash.

Open up a terminal window on your virtual machine and enter the command "ping <yourName>Beaglebone.local". This will result in your Linux virtual machine trying to ping the Beaglebone and returning the ip address that has been assigned by the dhcp server to the device. The screen capture below shows an example. Notice that in this case, the router has assigned the ip address 192.168.0.107 to the Beaglebone board. Yours will most likely be different.

If this does not work, you can run wireshark on your machine and look for the mDNS entries. This will show the IP traffic going to your board.



6 Expanding the image

The image that you are provided with is designed to be small, but to maximize its effectiveness, you will need to expand it to fill your entire micro sd card. To do this, log onto the device as root via ssh and issue the following commands:

```
cd /opt/scripts/tools
git pull
./grow_partition.sh
sudo reboot
```

This will expand the partition to fill your entire micro-sd card, providing enough space to complete the lab assignments.

7 Writing the first program

Once the tools have been installed, open up a emacs (or other editor) window and enter the program shown in Figure 3. This program will display assorted pieces of information about the machine on which it is running.

Once you have typed the program in locally, run it under the virtual machine and verify that it works. To do this you will use the gcc compiler. Take a snapshot of the screen showing the program executing for your lab report.

When you are finished, cross compile the program for the Beaglebone. When you are done with the cross compilation, download the program to the target and execute it on the Beaglebone. To do this, first ping your machine to determine its ip address. Once you have done this, open a secure ftp session with the Beaglebone to transfer the program to it. This is done by typing sftp root@192.168.0.xxx where xxx is the unique part of your ip address. This will make a connection to the remote machine, and the program can be transferred by typing put a.out. Exit out of the sftp session by typing exit.

Now open up a secure shell session with the Beaglebone. This is done by issuing the command ssh root@192.168.0.xxx . This will open a terminal session that you can use to execute the program on the Beaglebone. As you are executing on the Beaglebone, take a screen capture of your lab report showing it functioning properly.

```
#include <stdio.h>
#include <stdlib.h>

int bss_var;              /* This is an uninitialized global variable. */
int data_var = 1;/* This is an initialized global variable. */
static int  lv;
static int lv2 = 1;

int main(int argc, char* argv[])
{
  void *stack_var; /* This is a local variable declared on the stack.*/

  char hostname[1024];  // This is an array declared on the stack which will hold the hostname.

  stack_var = (void*) main;

  printf("Hello WOrld!  Main is executing at address %p\n", stack_var);
  printf("This address (%p) is in our stack frame\n", &stack_var);

  printf("The printf function is located at location %p.\n", printf);

  /* The bss section contains uninitialzed data. */
  printf("This address (%p) is our bss section\n", &bss_var);

  /* Data section contains initialized data */
  printf("This address (%p) is in our data section\n", &data_var);

  /* This is a static variable which is uninitialized. */
  printf("The address of the static variable which is uninitialized is %p.\n",  &lv);

  /* This is a static variable which is initialized. */
  printf("The address of an initialized lv variable is %p\n", &lv2);

  /* read the hostname. */
  gethostname(hostname, 1024);

  /* Print out the hostname to the console.*/
  printf("Hostname: %s\n", hostname);
}
```

Figure 1 Our first program (extended from Embedded Linux Primer: Second Edition)

8 Expanding

Now that you have been successful, you want to try and use makefiles to generate your code for you. Makefiles allow great automation when creating large projects.

At the end of the document, there are two makefiles. One will work for a Beaglebone target and one will work for the host platform. They also are available on the course website. Download them to your development directory and issue the command "make all" or "make –f makefile.bb all" to make the host code or the target code respectively. "make clean" will clean the directory.

9 Building our first calculator

Now that we have completed the first program and made it with a makefile, it is time to move on. On the course website there is an archive file for the calculator project. This project will compile and execute a simple calculator. Download the tar file and extract it into a development directory. (To extract the tar file, use the command "tar –xvzf name.tar.gz". Details about tar can be found by issuing the command man tar".

Make the calculator program and execute it on the Beaglebone. Make a screen capture showing it working properly on your remote device.

10 Deliverables / Submission

This lab has primarily been focused on learning about the environment. The lab report is therefore brief about what you found and what you learned.

Each person will be responsible for submitting one report with the following contents:

1. Introduction -> What are you trying to accomplish with this lab? This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment.
2. Screen captures
   a. Include screen captures showing the execution of the program on both the development host and the target Beaglebone.
   b. Include a screen capture showing the calculator working on the Beaglebone.
3. Questions
   a. What is different about the addresses given to the variables on the Beaglebone versus the host?
   b. How big is the executable for the first program when compiled for the host?
   c. How big is the executable for the first program when compiled for the target?
4. Conclusions -> What have you learned with this experience? What improvements can be made in this experience in the future?

This material should be submitted as a single pdf file.
If you have any questions, consult your instructor.

Makefile for Linux

```
SHELL = /bin/sh
SRCDIR = .
CC = gcc
YACC = bison -y
CDEBUG = -g
COMPLIANCE_FLAGS = -save-temps
CFLAGS = $(COMPLIANCE_FLAGS) $(CDEBUG) -I. -I$(SRCDIR)
LDFLAGS = -g -lm

########################################################################################################
# List your sources here.
SOURCES = hello.c
########################################################################################################

########################################################################################################
# list the name of your output program here.
EXECUTABLE = hello
########################################################################################################
# Create the names of the object files (each .c file becomes a .o file)
OBJS = $(patsubst %.c, %.o, $(SOURCES))

include $(SOURCES:.c=.d)

all : $(OBJS) $(EXECUTABLE)

$(EXECUTABLE) : $(OBJS)
        $(CC) -o $(EXECUTABLE)  $(OBJS) $(LDFLAGS)

%.o : %.c #Defines how to translate a single c file into an object file.
        echo compiling $<
        $(CC) $(CFLAGS) -c $<
        echo done compiling $<

%.d : %.c #Defines how to generate the dependencies for the given files.  -M gcc option generates
dependencies.
        @set -e; rm -f $@; \
        $(CC) $(COMPLIANCE_FLAGS ) -M $< > $@.$$$$; \
        sed 's,\($*\)\.o[ :]*,\1.o $@ : ,g' < $@.$$$$ > $@; \
        rm -f $@.$$$$

clean :    # Delete any and all artefacts from the build.  The only thing which is kept is the
source code.
        rm -f *.o
        rm -f *.i
        rm -f *.s
        rm -f *.d
        rm -f $(EXECUTABLE)
```

11 Cross compile makefile for the Beagleboard

```
SHELL = /bin/sh
SRCDIR = .
CC = arm-linux-gnueabi-gcc
YACC = bison -y
CDEBUG = -g
COMPLIANCE_FLAGS = -save-temps
CFLAGS = $(COMPLIANCE_FLAGS) $(CDEBUG) -I. -I$(SRCDIR)
LDFLAGS = -g -lm

########################################################################################################
# List your sources here.
SOURCES = hello.c
########################################################################################################

########################################################################################################
# list the name of your output program here.
EXECUTABLE = hello
########################################################################################################
# Create the names of the object files (each .c file becomes a .o file)
OBJS = $(patsubst %.c, %.o, $(SOURCES))

include $(SOURCES:.c=.d)

all : $(OBJS) $(EXECUTABLE)

$(EXECUTABLE) : $(OBJS)
        $(CC) -o $(EXECUTABLE)  $(OBJS) $(LDFLAGS)

%.o : %.c #Defines how to translate a single c file into an object file.
        echo compiling $<
        $(CC) $(CFLAGS) -c $<
        echo done compiling $<

%.d : %.c #Defines how to generate the dependencies for the given files.  -M gcc option generates
dependencies.
        @set -e; rm -f $@; \
        $(CC) $(COMPLIANCE_FLAGS ) -M $< > $@.$$$$; \
        sed 's,\($*\)\.o[ :]*,\1.o $@ : ,g' < $@.$$$$ > $@; \
        rm -f $@.$$$$

clean :    # Delete any and all artefacts from the build.  The only thing which is kept is the
source code.
        rm -f *.o
        rm -f *.i
        rm -f *.s
        rm -f *.d
        rm -f $(EXECUTABLE)
```