Creating Small Standalone Raspberry Pi Images

ECE598: Advanced Operating Systems – Homework 1

Spring 2015

Due: Thursday, 29 January 2015, 9:30am

This homework is meant to get you started with the Raspberry Pi and writing simple self-contained programs.

1. **Install a Cross-Compile Toolchain**

   - A reference on how to do this is here: `http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html`
   - For Linux there are various alternate ways to install an ARM toolchain if you don't want to download the files as described in the link above. Feel free to use whatever works.
   - For OSX follow the instructions. You might have to install xcode to get `make` installed (on newer OSX it will prompt you to do this automatically if you try to run make at the command line).
   - I have not tested the Windows instructions. If there are any notes on getting it working that you discover, let me know.

2. **Download the homework code template**

   - Download the code from: `http://web.eece.maine.edu/~vweaver/classes/ece598_2015s/ece598_hw1_code.tar.gz`
   - Uncompress the code (on Linux or Mac you can just `tar -xzvf ece598_hw1_code.tar.gz`)

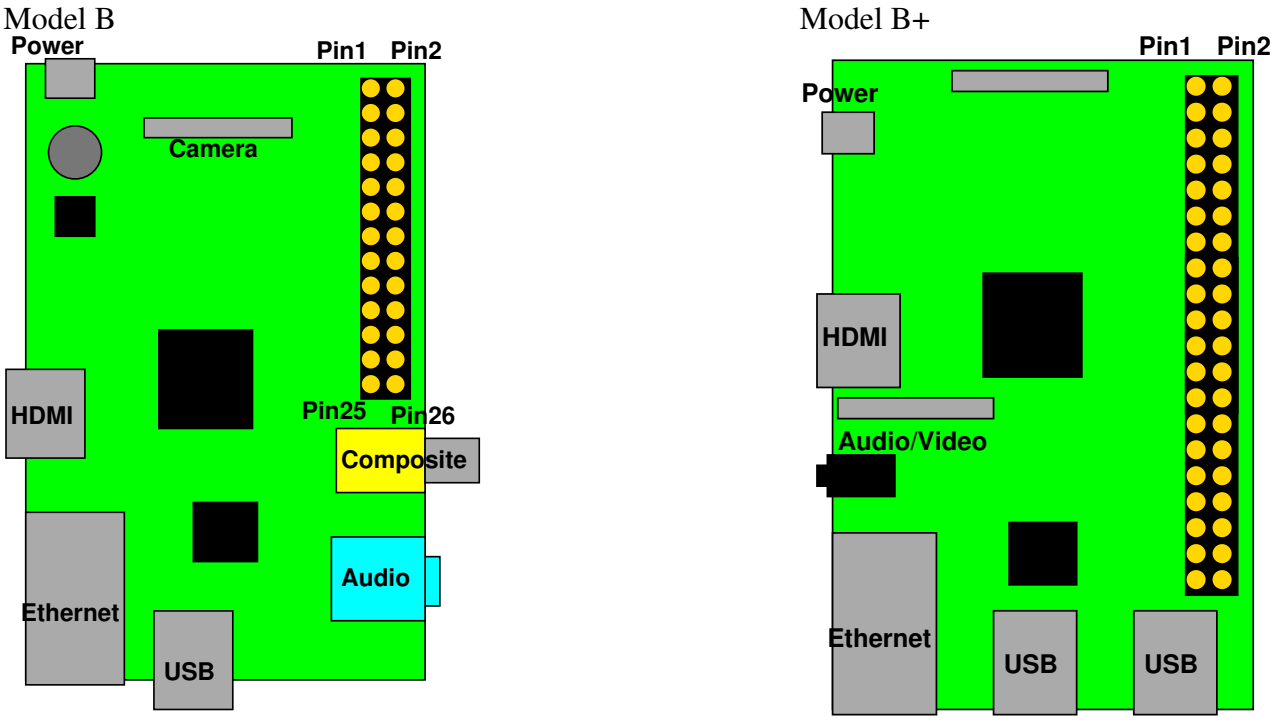

Figure 1: Raspberry Pi Layout

3. **Modify the Assembly program to Blink the ACT LED**

   - We went over how to do this in class. For reference view the GPIO related code in the BCM2835-ARM-Peripherals.pdf manual that can be found linked on the course website.
   - If you are using a Raspberry Pi Model B then you turn on the LED by setting GPIO16 Low. If you are using a Raspberry Pi Model B+ then you turn on the LED by setting GPIO47 High. You can look at Figure 1 to see what kind you have if you don't know. I you are using a Model A you're on your own.
   - Modify the `rpi_blink.s` file under the `part1_asm` directory so it blinks. (Look for "your code here" references). As a reminder, you will need to enable the proper GPIO, turn on the LED, delay (0x3f0000 is a good amount to loop), turn off the LED, delay again, then loop forever.
   - Be sure to comment your code!

4. **Build the Assembly Program**

   - Run "make". You may need to modify the Makefile so that the CROSS variable is prepended by the directory where you install the cross compiler. For example, if you you are using the yagarto cross-compiler, you will update the line to look something like:
     `CROSS = ~/yagarto/yagarto-4.7.2/bin/arm-none-eabi-`
   - If all goes well it should create a `kernel.img` file.

5. **Install on the SD card and Test**

   - Insert your SD card. I'm assuming you have one formatted with Raspbian. (Note, to be safe you might want to use an SD card that doesn't have anything else important on it).
   - You should be able to find the boot partition, which is a fat partition that should already have a kernel.img there. On Mac it helpfully automounts as a drive called "Boot". On Linux depending on what version you are running you may have to mount it by hand (it will be the first partition on the drive, something like /dev/sdb1) and you might have to be root or use sudo to copy the file into place.
   - *IMPORTANT* Backup your `kernel.img` file. Make a copy (call it `kernel.img_good` or something like that). You will need this if you ever want to boot your Pi back into Linux again.
   - Copy the `kernel.img` file you build over top of the one on the memory key.
   - Safely unmount the SD card.
   - Place it in your Raspberry Pi. Apply power to the Pi. If all goes well the ACT light should be blinking. If not, check your code!
   - To be safe you should probably remove power to your Pi before removing the SD card again.

6. **Modify the C program to Blink the ACT LED**

   - Modify the `rpi_blink.c` code in the `part2_c` directory to blink the ACT light.
   - Be sure to comment your code!
   - Running "make" should build the code (you might have to modify the `CROSS` value in the Makefile again).

7. **Install on the SD card and Test**

   - Do this the same way you did for the assembly version. Be sure you are properly over-writing the file.

8. **Answer the following questions**

   Put your answers in some sort of document (.txt, .pdf, .doc).

   (a) What is one of the benefits to using an operating system?

   (b) What is a downside to using an operating system?

   (c) Measure the size of your kernel.img files for the assembly and C versions (hint, you can use `ls -l` (that's a lower-case L). Which is bigger? Why? How does it compare in size to the Linux `kernel.img` that you backed up?

   (d) What is the purpose of the C `volatile` keyword?

   (e) In `rpi_blink.c` `GPIO_GPCLR0` is defined as 10 but in `rpi_blink.s` it is defined as 40. Why is this different?

   (f) Look at the BCM2835_ARM_Peripherals document, section 6.2. If we had accidentally set the GPIO_GPFSEL1 register for GPIO16 usage to type ALT4, what mode would that pin have been set to?

9. **Submit your work**

   - E-mail me your `rpi_blink.s`, `rpi_blink.c`, and the document with the answers to the questions. You can zip these up together if it makes it easier.