Blinking an LED on a Bare-metal Raspberry Pi

ECE598: Advanced Operating Systems – Homework 2

Spring 2018

Due: Thursday, 8 February 2018, 2:00pm

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

You may work in a group of two for this assignment, although it should be possible to do it on your own.

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

   - A good reference on how to do this can be found here:
     `http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html`
   - For Linux the easiest way (at least on Debian or Ubuntu) is to do something like:
     `apt-get install gcc-arm-none-eabi`
     but there are various other ways to install things 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 personally tested the Windows instructions, however I know people successfully used it in previous years.

2. **Download the homework code template**

   - Download the code from:
     `http://web.eece.maine.edu/~vweaver/classes/ece598/ece598_hw2_code.tar.gz`
   - Uncompress the code (on Linux or Mac you can just `tar -xzvf ece598_hw2_code.tar.gz`)

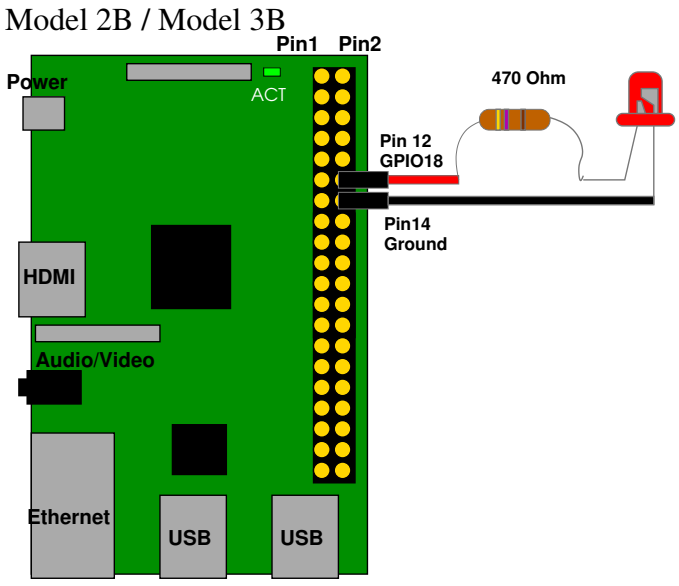

Figure 1: Raspberry Pi Layout

3. **Setup GPIO18 so you can Monitor the Output**

   - The easiest way to do this is hook a 470Ω resistor and LED to GPIO18 (Pin12) and Ground (Pin14) as shown in Figure 1.
   - You can also use a voltmeter or digilent for this instead.

4. **Modify the Assembly program to Blink the LED on GPIO18**

   - We went over how to do this in class. For reference view the GPIO related information found in Chapter 6 of the `BCM2835-ARM-Peripherals.pdf` manual that can be found linked on the course website.
   - Modify the `blink_as.s` file so that it blinks the LED. (Look for "your code here" references). As a reminder, you will need to enable the proper GPIO, turn on the LED, delay, turn off the LED, delay again, then loop back so the LED blinks forever.
   - Adjust the delay to see if you can get it close to 1Hz with 50% duty cycle (.5s on, .5s off).
   - Be sure to comment your code!

5. **Build the Assembly Program**

   - Run "make". You may need to modify `Makefile.inc` 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 `blink_asm.img` file.

6. **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 kernel7.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 `kernel7.img` file. Make a copy (call it `kernel7.img_good` or something like that). You will need this if you ever want to boot this SD card back into Linux again.
   - Copy the `blink_asm.img` file you build over top of `kernel7.img` on the SD card. (Make sure you over-write kernel7.img, you can't just copy blink_asm.img there.)
   - Safely unmount the SD card.
   - Place it in your Raspberry Pi. Apply power to the Pi. If all goes well the the LED on GPIO18 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.

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

   - Modify the `blink_c.c` code to blink the ACT light.
   - Be sure to comment your code!
   - Running "make" should build the code

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

   - Do this the same way you did for the assembly version. This time copy the `blink_c.img` file overtop of `kernel7.img`. Be sure you are properly over-writing the file.

9. **Examine some build files**

   - This isn't worthy any points, but take a look at the `boot.s` file used to setup for the C code, as well as the `kernnel.ld` linker script just so you are aware of what they are and what they are doing.

10. **Answer the following questions**

    Put your answers to the following questions in the README text file.

    (a) Measure the size of your .img files for the assembly and C versions (If you're on Linux/OSX you can use `ls -l` (that's a lower-case L) to get filesize). Which is bigger? Why? How does it compare in size to the Linux `kernel7.img` that you backed up?

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

    (c) In `blink_c.c` `GPIO_GPCLR0` is defined as 10 but in `rpi_blink.s` it is defined as 40 (0x28). Why is this different?

    (d) Look at the BCM2835_ARM_Peripherals document, section 6.2. If we had accidentally set the GPIO_GPFSEL1 register for GPIO18 usage to type ALT4, what mode would that pin have been set to?

11. **Submit your work**

    - E-mail me your solution. On Linux/OSX you can run `make submit` which will create a `hw2_submit.tar.gz` file containing all of the source files plus the README with your answers. Alternately, if you are on Windows and make submit doesn't work then just create a zip file of your solution directory.
    - If you worked in a group, you only need to submit one submission. Be sure to put both group members names in the README document.