ECE531: Advanced Operating Systems – Homework 2
Blinking an LED on a Bare-metal Raspberry Pi

Due: Friday, 15 September 2023, 5:00pm

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

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

   - On a machine running 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 MacOS or Windows you can try installing the cross-compile toolchain direct from ARM:
     `https://developer.arm.com/downloads/-/gnu-rm`
     - For MacOS 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. Possibly if you are using the Linux subsystem for Windows it might be easier to use that.

2. **Download the homework code template**

   - Download the code from:
     `https://web.eece.maine.edu/~vweaver/classes/ece531/ece531_hw2_code.tar.gz`
   - Uncompress the code (on Linux or Mac you can just `tar -xzvf ece531_hw2_code.tar.gz`)

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 C program to Blink the GPIO18 LED (5pts)**

   - 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_c.c` code to blink the GPIO18 pin/LED
   - First, find the `IO_BASE` defines near the beginning and uncomment the line corresponding to the type of Pi you have.
   - Next modify the code where necessary. As a reminder, you will need to enable the proper GPIO, turn on the GPIO, delay, turn off the GPIO, delay again, then loop back so the LED blinks forever.
   - Adjust the timing so it blinks at roughly 1Hz (500ms on/500ms off)

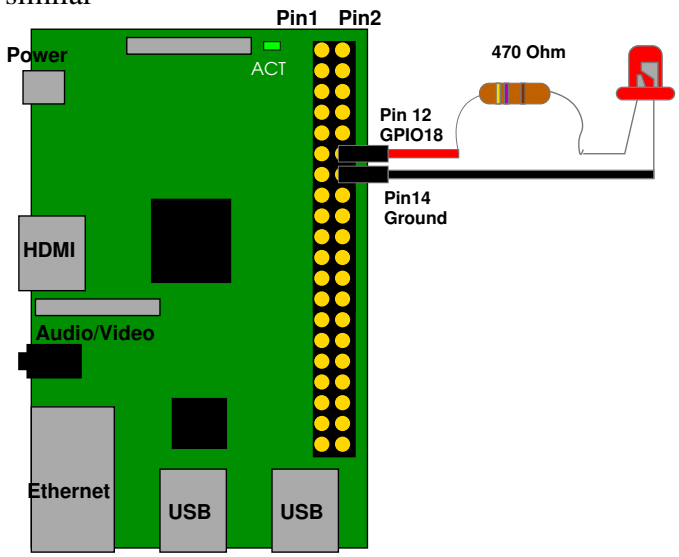

Figure 1: Raspberry Pi Layout

- Be sure to comment your code!

5. **Build the C code**

   - Run "make"
   - If it complains about not being able to find your C compiler you may need to modify `Makefile.inc` so that the CROSS variable is prepended with the directory where your cross compiler is installed. 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 a `blink_c.img` file will be created

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

   - Place the SD card into your development system. It should have a relatively recent Raspberry Pi OS on it as we will be re-using the firmware and bootloader on the `/boot` partition. (Note, to be safe you might want to use an SD card that doesn't have anything else important on it). If you need to install Linux on a blank SD card see `https://www.raspberrypi.com/software/` for directions.
   - Once the SD card is in your development system you will need to copy a file to the boot partition. It is a FAT filesystem so Linux/MacOS/Windows should all see it fine. Ideally your operating system will-auto mount this as "Boot" and you should see the existing kernel7.img files. 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.
   - The file we want to replace is either `kernel7.img` if you have a Pi3 or `kernel7l.img` (that's an L) on a Pi4. In the following directions I'll say `kernel7.img` for simplicity but if you have a Pi4 just use `kernel7l.img` instead

- ***IMPORTANT*** Backup your `kernel7.img` file if you ever want to boot back into Linux using this SD card again. Make a copy (call it `kernel7.img_good` or something like that). If you want to boot back to Linux you can just copy it back in place.
- If you are using a Pi4, you will need to force it to use 32-bit mode. (Note, you only have to do this once). Edit the file `config.txt` and add the line
  `arm_64bit=0`
  to the end of the file.
- Copy the `blink_c.img` file you built over top of `kernel7.img` on the SD card. (Make sure you over-write `kernel7.img`, you can't just copy blink_c.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. **Examine some build files**

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

8. **Something Cool: Modify the Assembly program to Blink the LED on GPIO18 (1pt)**

   - Once you get the C example working, try to see if you can get the assembly version working too.
   - Make sure the proper `IO_BASE` line is uncommented at the top of the file.
   - 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!
   - Run "`make`".
   - If all goes well it should create a `blink_asm.img` file.
   - Install on the SD card the same way you did for the assembly version. This time copy the `blink_asm.img` file overtop of `kernel7.img`. Be sure you are properly over-writing the file.

9. **Answer the following questions (4pts)**

   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 `blink_asm.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?

10. **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.