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

Due: Friday, 19 September 2025, 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**

   - **Linux**: 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`
     Feel free to use whatever works.
   - **MacOS**: For MacOS you can try installing the cross-compile toolchain direct from ARM:
     `https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads`
     You will want the file
     `arm-gnu-toolchain-14.3rel1-darwin-arm64-arm-none-eabi.pkg`. Also the first time you run `make` it will prompt you to install the command line xcode development tools (which you will want to do).
   - **Windows** 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 you can use apt to install the cross compiler as with Linux. You can also get the official ARM windows toolchain from the same place as the MacOS versions.

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. **Modify the C program to Blink the ACT LED (GPIO47) (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 GPIO47 pin/LED
   - The different Pi models have different `IO_BASE` offsets for the start of the I/O region. This year we are all using Pi 1B+ models so there is no reason to change this value.
   - 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)
   - Be sure to comment your code!

4. **Build the C code**

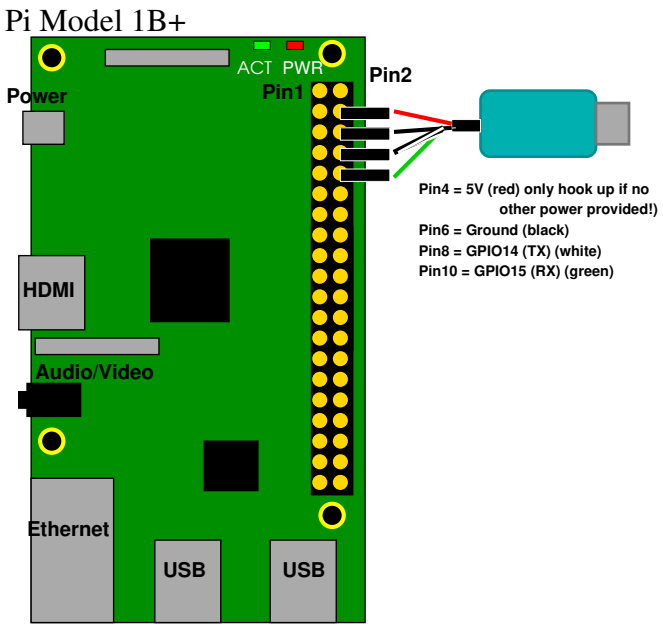

Figure 1: Raspberry Pi Layout

- 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, on MacOS using the ARM supplied tools you will need something like:
  `CROSS = /Applications/ArmGNUToolchain/14.3.rel1/arm-none-eabi/bin/arm-none-eabi-`
- If all goes well a `blink_c.img` file will be created

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

   - Place the SD card into your development system. Hopefully it has an SD-card slot, or alternately you have a USB/SD adapter.

     The SD card I provided should have a relatively recent Raspberry Pi OS on it; we will be re-using the firmware and bootloader on the `/boot` partition. (If for some reason you want to re-install yourself, you can find directions on how to do that at `https://www.raspberrypi.com/software/`).
   - 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 kernel.img file.

     (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 on the Pi-1B+ is `kernel.img`
   - ***IMPORTANT*** Backup your `kernel.img` file if you ever want to boot back into Linux using this SD card again. Make a copy (call it `kernel.good` or something like that). If you want to boot back to Linux you can just copy it back in place.
   - Copy the `blink_c.img` file you built over top of `kernel.img` on the SD card. (Make sure you over-write `kernel.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.
  (a) The normal way to power a Pi is via a USB-micro cable. You can do that for this assignment if you want
  (b) For all future assignments we will use the USB-serial adapter and you can power things with it (See Figure 1). I'd recommend you power things that way in this assignment just so you're used to it and it's fewer cables to worry about.
  (c) Don't power your device with both USB and the USB-serial at the same time. If you want to power with USB be sure the red wire on the USB-serial cable is not hooked up
- If all goes well first the red power (PWR) LED will come on, and a short time later the green ACT LED will blink as well.

  If this does not happen, check the notes at the end of the homework for troubleshooting hints.
- To be safe before inserting/removing the SD card remove the power first.
- Note! When removing the SD card, on the Model 1B+ don't just rip it out, it's the old-style slot where you push it in lightly and it will pop out.

6. **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.

7. **Something Cool: Modify the Assembly program to Blink the LED on GPIO47 (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 `kernel.img`. Be sure you are properly over-writing the file.

8. **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 `kernel.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 were using GPIO18 and had accidentally set the GPIO_GPFSEL1 register for GPIO18 usage to type ALT4, what mode would that pin have been set to?

9. **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.

10. **Troubleshooting:**

    - **If the red PWR LED comes on but then both the red / green LEDs blink a lot but not in a regular pattern.**

      This usually means you didn't over-write `kernel.img` properly and Linux is booting. Try copying your file over again.

    - **The red PWR LED comes on but the green ACT LED doesn't.**

      This usually means something is wrong with your code. Try to debug!

      Alternately, if somehow you remove your card from your development system w/o properly shutting down first then this can happen. On Linux if you put the SD card in and look at the `dmesg` look it will complain like this `Volume was not properly unmounted. Some data may be corrupt. Please run fsck.` In this case either run `chkdsk` or similar (on Windows) or run `fsck.fat` on it (Linux)