CS466 Lab 1 -- Hardware, Development Tools and Blinking the LED
Due by Midnight Friday 2-4-2021.

!!! Must use provided lab format on Blackboard !!!

Note: This is an individual lab, you are free to collaborate but every student must perform the lab and hand in a lab report. It will be critical that each student is able to develop to the target platform..
I will be checking out a development board to each student. If you lose it, soak it, step on it, or let the smoke out of it during class you will be expected to replace it. They cost around $15 and I will collect them at the end of the semester.

Overview:

We will be using a TI Launchpad TM4C123G board. It's essentially a minimal development board to showcase the TI ASIC TM4C123GH6PMI microcontroller. The ASIC has a processor and a bevy of integrated peripherals that the developer can combine and use to control an arbitrary device.

Benefits of the hardware:

- Very Affordable, less than $15.00
- Industry standard 32 bit ARM Cortex-M4F processor running at 80MHz
- Open source Gnu tool chain, for Windows, Linux, OS-X
- Small and versatile.

Cons of the new hardware:

- No Snazzy IDE, command line tools and debugger.
- Very little electrical protection on I/O pins (We need to be careful)
  - Can damage dev-board (we've roasted a couple)
  - Potentially could damage host computer USB hardware (we've not done this yet)

Resources:

- Tiva™ C Series TM4C123G LaunchPad Evaluation Board User's Guide
- Tiva™ TM4C123GH6PM Microcontroller DATA SHEET

Lab Preparation:

- Take a short look at the board Users Guide. The guide is strewn with links back to TI for things to look at and install.. I suggest that you wait till Lab to decide what software to install, I will provide a minimal set. We will be using a command line GCC toolchain (not outlined in the guide).

- Take a slightly longer look at the microcontroller datasheet. This is a huge pdf (1409 Pages).. While we will be looking into specific sections in detail for now I want you to read sections 1, 1.1, 1.2, 1.3, 10, and 10.1.

  - If you are not familiar with data sheets this can be a daunting document.. Part of what we will be discussing in lab is how to not freak out when presented with all the data. The EE students have an early edge here having been exposed to data sheets before but the CS guys get it back later in the class.

Objective:

This lab is mostly to familiarize students with the LaunchPad development board and development tools. Making use of additional GPIO pins will require some basic understanding of the schematic and understanding some of the GPIO section of the microcontroller reference manual.

Note that a popular convention is used on the blinky.c code to write to a fixed address (where controlling registers in the ASIC are located.) The code has the line

```
GPIO_PORTF_DIR_R = (1<<3);
```

And if you follow the header files back we see the definition

```
#define GPIO_PORTF_DIR_R        (*((volatile uint32_t *)0x40025400))
```

In short `GPIO_PORTF_DIR_R` is a literal that is forced to resolve as a pointer to an address. The assignment of (1<<3) to the token results in the value `0x00000008` being written to address `0x40025400`. I will use this notation a lot during class so be sure that you understand it.

The 'volatile' storage type qualifier informs the compiler not to optimize or otherwise play with the storage location or method.

Since all 32 bits of the register may have some behavior this instruction would set our interesting bit but clear the other 31 bits. The quick and dirty way for us to prevent stepping on the register contents is to perform a read-modify-write operation which in C can be coded as

```
GPIO_PORTF_DATA_R |= (1<<3);
```

Which would read all 32 bits, set the `0x00000008` bit, and then write the whole 32 bit word. The other 31 bits retain their pervious state. We have to perform some logic tricks to clear the same bit yet remain readable in the code.

```
GPIO_PORTF_DATA_R &= ~(1<<3);
```

Lab Work

1. ☐ For LAB 1 I would like everyone to complete their work on Linux systems.

2. ☐ All of the lab material will be available on a my git repo. Choose a directory to use to locate your cs466 project in (I recommend ~/src). From that directory run the command:

```
$ git clone https://github.com/milhead2/cs466_s21
```

Using my standard this should create a directory ~/src/cs466_s21 with the first set of class files. Keep the directory tree in this format and support will be easier for your labs. If you fork the repo you will be able to update and preserve your class work. If you just clone my repo you will not be able to 'push' your changes. As a student you should be able to get a free github account

The README.md file on the class repo gives instruction on how to setup the directory.

To make me solve fewer issues install the compiler, and the TivaDriver, directories. In the end you should have the following directories together on your disk

<your-parent>/cs466_s21/
<your-parent>/TivaDriver/
<your-parent>/<arm-none-compiler-dir>

And in Lab2 we will be adding

<your-parent>/FreeRTOSv202012/

I only 'add' or 'modify' my files in this repo so you are free to work in your copy of the repo. It would be 'more proper' to fork the repo. As I only add files we should not see conflicts if you pull to get added files.

3. ☐ Open the box and familiarize yourself with the Tiva Launchpad development board.

4. ☐ Check that the complier and flash tools are available on the system by running the commands that all should give a reasonable response. I have provided setup text files for Linux, Windows, and OSX. Keep in mind that the Windows solution will last for more than a few weeks. We have never really gotten gdb and to operate properly on a windows PC.

```
$ arm-none-eabi-gcc -version
$ lm4flash -v
$ openocd -version
```

5. ☐ Verify that when you plug in the development board on Linux it is recognized by the OS. After you have the board plugged in and running, type a dmesg command to display the latest system stuff that occurred on the workstation;

```
$ dmesg
```

'dmesg' will display a gob of stuff but the last few lines are most interesting. You should see something like…

```
[271134.980843] usb 3-1.2: new full-speed USB device number 109 using xhci_hcd
[271135.086354] usb 3-1.2: New USB device found, idVendor=1cbe, idProduct=00fd
[271135.086361] usb 3-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[271135.086365] usb 3-1.2: Product: In-Circuit Debug Interface
[271135.086368] usb 3-1.2: Manufacturer: Texas Instruments
[271135.086371] usb 3-1.2: SerialNumber: 0E218406
[271135.114239] cdc_acm 3-1.2:1.0: ttyACM1: USB ACM device
```

This shows that Linux has seen the device and understands what it is.

6. ☐ The Launchpad board comes pre-loaded with a default program that will showcase the RGB LED on the board. Connect the board to your host computer then set the board 'Power Select' switch to 'Debug'. You may need to press the 'Reset' button to start the program.

a) (#1) How would you describe the default program
b) (#2) What do SW1 and SW2 do to the default program

7. ☐ Get your development environment setup so that you can build the oversimplified Blinky program that will be provided in lab.

a) Extract the lab01_blinky archive so that the 'lab01_blinky' directory resides in the 'labs' directory of your directory tree
b) Download 'makedefs' and install it in your lab directory.

8. ☐ Use the LM Flash program to upload the gcc/blinky.bin image to the Tiva board. The LED should flash bright green if the blinky program is working.

9. ☐ Use the schematic in the Users Guide and the existing code to examine how the GPIO pins of the LaunchPad are driven to light the LED. Alter the program to blink the Red or Blue LED in place of the Green one. Compare the GPIO register settings to the Microcontroller reference manual (GPIO section).

10. ☐ Switch the blinking LED back to green and measure the frequency of the blinking LED. Modify the high speed delay routines to get as close as possible to a 1Hz blink frequency.

11. ☐ Add code to read the status of SW1 and SW2. You will need to enable new GPIO pins for input and internal pull-up resistors.

a) (#3) Do both switches follow the same rules (hint, hint, maybe the datasheet says something).

12. ☐ Modify the code so that:

a) The green LED always blinks at 1.0Hz

b) By default the red and blue LED, are off.
c) Pressing SW1 will cause the red led to flash 20 times at 15Hz
d) Pressing SW2 will cause the blue led to flash 10 times at 13Hz
e) This should work with any combination of SW1 and SW2 (ugly blinkin!)
f) Use only the button press as a trigger, the led should blink their required number of times no matter how long the button is pressed.

13. ☐ Verify the frequency (or as close as you can get) by writing some code that can verify the frequency of the three LED blinks.

a) (#4) What is the purpose of the blinky.ld file?
b) (#5) What is the purpose of the startup_gcc.c file?

14. ☐ Without a friendly operating system to startup our code this environment is a bit more bare than you are 'probably' used to.
a) (#6): Describe the execution path from processor reset until main is called.

15. Write up your lab using the lab format provided on Blackboard. Include your program as a fixed spaced (`I recommend Lucida Console`) addendum to your lab. I will cut points for proportionally spaced code pasted in the end of the lab.

Submit your lab as a 'SINGLE' PDF file on Blackboard. Name the file
`CS466L01_Name(s)_Report.pdf`

…. Not a .zip File
…. Not a set of files
…. Not a set of .jpg files from your phone..

(I'm not too much a jerk.. Using a single PDF makes grading, feedback and organization of reports easier for everyone)