CS466 Lab 1 -- Hardware, Development Tools and Blinking the LED
Due by Midnight Thursday 1-19-2017.

!!! 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 loose 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
  - I want this lab only completed on the linux platform, If you don't have a linux laptop use one of the lab machines.. One of the benefits of the tool chain it that it can run on Linux, PC's and Mac's but I think the best supported platform is Linux. In the past students have wasted the whole lab getting the tools installed and running.
- 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 I 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 the Lab Linux systems.

2. Open the box and familiarize yourself with the Launchpad dev board.

3. This year I have all the handed out files on a git repo. Choose a directory to use to locate your cs466 project in (I use ~/src). From that directory run the command:

   $ git clone https://github.com/milhead2/cs466_s17

   using my standard this should create a directory ~/src/cs466_s17 with the first set of class files. Keep the directory tree in this format and support will be easier for your labs.

4. 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) How would you describe the default program
   b) What do SW1 and SW2 do to the default program

5. 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.

6. 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.

7. 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).

8. 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.

9. 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. Do both switches follow the same rules (hint, hint).

10. 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.

11. Verify the frequency (or as close as you can get) with the scope.

12. What do you think is the purpose of the blinky.ld file is?

13. What do you think is the purpose of the startup_gcc.c file is?

14. Without an OS to startup our code this environment is a bit more bare than you are ‘probably’ used to. 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.

16. Submit your lab as a PDF file on Blackboard. I do not want printed labs this semester.

17. Due by Midnight Thursday 1-21-2016.