EE194/EE290C HW2
Spring 2017

Sign up for a dev account at dev.sifive.com before getting started. Use c125m machines to complete this.

1. Complete the SiFive Freedom E310 Arty FPGA Dev Kit Getting Started Guide, located here: https://dev.sifive.com/develop/freedom-e310-arty-dev-kit-v1-0/, but with changes noted below. We have one "Arty" FPGA board so work out a schedule to share it.

Chapter 1: all the hardware has already been purchased
Chapter 2: the individual colored wires have already been set up
Chapter 3: To launch vivado, run:

```
/share/instsww/xilinx/Vivado/current/bin/vivado
```

Chapter 4: These steps have already been done on the c125m machines. To connect to the serial port of the Arty, run:

```
screen /dev/ttyUSB2 115200
```

Note it could be ttyUSB1/2/3/4... but probably not 0. The c125m systems have a built-in ttyUSB0 already. Connect to the one that appears when you plug in the Arty board. Press "reset" after you connect and you should see a logo printed out.

Chapter 5: Tools have already been downloaded from github and installed on the c125m machines. To set things up in your home directory, make a working directory and copy the demo files in. Make a symlink to the tools because duplicating them isn't necessary. For example:

```
cd ~
mkdir freedom-e-sdk
cd freedom-e-sdk
cd /opt/freedom-e-sdk
cp -R bsp LICENSE Makefile README.md software ~/freedom-e-sdk/
ln -s /opt/freedom-e-sdk/toolchain ~/freedom-e-sdk/toolchain
```

Then you can pick up on page 14, "Compiling Software Programs." Skip the Arduino stuff and the rest of the guide.

2. Write a simple program that uses some features on the board. The program should:

-- Toggle some LEDs (normal LED 7 and tri-color LEDs 0, 1, and 2 are what we've verified as working)
-- Print something over UART
-- Be responsive to a button (button 0 is what we've verified as working)
-- Run over and over forever

You'll want to consult the sample software directory you copied over for (1), particularly demo_gpio and hello. You'll also need some GPIO mapping that they don't mention in those files, written below in the form of #defines that describe where in the 32-bit GPIO register you can find the bits that correspond to these board peripherals. Note that the SiFive documentation has some of these wrong, and what's below is right!

```
#define R_LED0_OFFSET 1
#define G_LED0_OFFSET 2
#define B_LED0_OFFSET 3

#define R_LED1_OFFSET 19
#define G_LED1_OFFSET 21
#define B_LED1_OFFSET 22

#define R_LED2_OFFSET 11
#define G_LED2_OFFSET 12
#define B_LED2_OFFSET 13

#define LED7_OFFSET 14
#define BUTTON_0_OFFSET 0x10
```

To complete this program you'll probably want to:

-- Start with the "hello" sample program as a base and build up from there
-- Generate memory-mapped IO addresses for relevant GPIO features like value, input enable, output enable, pull-up enable
-- Use the OFFSET values above to figure out where in a given 32-bit register, e.g., input_en, the bit is for a given peripheral. For instance, the red part of tri-color LED0 is controlled with bit 1 (aka the second bit) of each GPIO register.
-- Setup the register appropriately and write it to the right address (lots of places doing this in demo_gpio involving bit shifting and bitwise OR operations)
-- For buttons, you work the other way around: set that GPIO up as an input, read out the memory at the appropriate address, mask off the bits you don't care about, and shift that bit down to position 0 so you get a 1 or a 0 instead of (for instance) a 32768 or a 0.
-- Write a loop to do all this at some regular rate

Hints and notes:

Chapter 11 of the SiFive platform manual discusses what goes on with GPIO registers in more detail:
https://dev.sifive.com/documentation/freedom-e300-platform-reference-manual/

Recompile and test very often!

To use an LED, make sure you set that GPIO pin to be an output AND not an input. To use a button, make sure you set that GPIO bin to be an input AND not an output AND to not have a pull-up (there's already a pull-down on the board, see Arty reference at the end of this page). The code for doing this is in the demo_gpio software.

FYI the board runs at 65MHz and the core can run approx. (65/2)x10^6 loops in one second.

Once UART output is working, you can use printf to print values for debugging, e.g.,

```
printf("My debug number blah is: %u\n", blah); // print unsigned int "blah"
```

The Arty board reference manual might also be helpful:
https://reference.digilentinc.com/reference/programmable-logic/arty/reference-manual

If you're totally lost about how to complete this assignment, work with someone in the class with microcontroller and/or digital experience!