Project 1: Basic Assembly and C Programming

This project has 3 parts. Part 0 will be due on **January 30, 2019** at **10 a.m.** (as Assignment 1a on D2L). Parts 1 and 2 will be due on **February 13, 2019** at **10 a.m.** (as Assignment 1b on D2L). Together (Part 1a and 1b), this project counts for 5% of your course grade. Late submissions are not accepted. If you have a conflict due to travel, interviews, etc., please plan accordingly and turn in your project early.

The code and other answers you submit must be entirely your own work, and you are bound by the Honor Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with others to develop a solution. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

Solutions must be submitted electronically via Dropbox for each part on **D2L**, following the submission checklist below.

Introduction

In this project, you will program a simple assembly program, and run it on the DE10-Lite board. We will extend the Simple Program from the DE10-Lite setup tutorial. If you have not already done so, we recommend you program this on your board, and verify that toggling the switches toggles the corresponding LED above the switch.

Objectives:

- Understand how to create projects and program the DE10-Lite board.
- Use Nios II assembly to do basic bit manipulation of registers.
- Understand how a CPU interprets and executes instructions.

Part 0. Installing Software (Due January 30, 2019 as part of Assignment 1a)

We will be using the DE10-Lite board for this course (`http://de10-lite.terasic.com/`). You will need to purchase one yourself or borrow from another student (though your work should be your own).

Operating System

The software we will use only runs on Windows or Linux (I'm using it on Ubuntu 16.04). You should consider using a virtual machine if you have a Mac—VirtualBox is highly recommended (`https://www.virtualbox.org/`)—or you can use Parallels or dual boot using Apple Bootcamp.

Utilities

We will be editing code, so you'll need a text editor. On Linux, you can use `vim` or `emacs`. For Windows, NotePad++ (`https://notepad-plus-plus.org/`) is recommended. We will also need to compress (tarball) files. This utility (`tar`) will already exist on Linux, but for Windows we recommend 7-zip (`http://www.7-zip.org/`).

Install the Quartus Software

Download the Quartus Lite 16.1 Software: `http://dl.altera.com/16.1/?edition=lite`. Note: this will require you to register/create an Intel account.

This is a 5.8 GB file, so I recommend downloading from campus or somewhere with fast Internet.

Linux

For Linux, extract the `tar` file (`$ tar xf Quartus-lite-16.1.0.196-linux.tar`), and run the setup (`$ ./setup.sh`). Follow the prompts, making note of the directory it installs to (mine installs to `~/intelFPGA_lite/16.1/`).

Windows

Run the EXE and install to `C:\intelFPGA_lite\16.1`.

Install the Monitor Program

Download the University Monitor Program for your operating system, for Quartus 16.1: `https://www.intel.com/content/www/us/en/programmable/support/training/university/materials-softw html?&ifup_version=16.1#University-Program-Installer`.

You should be able to click on the link above. If not, the below is all 1 URL, split across lines for this pdf.

`https://www.intel.com/content/www/us/en/programmable/support/training/university/materials-software.html?&ifup_version=16.1#University-Program-Installer`

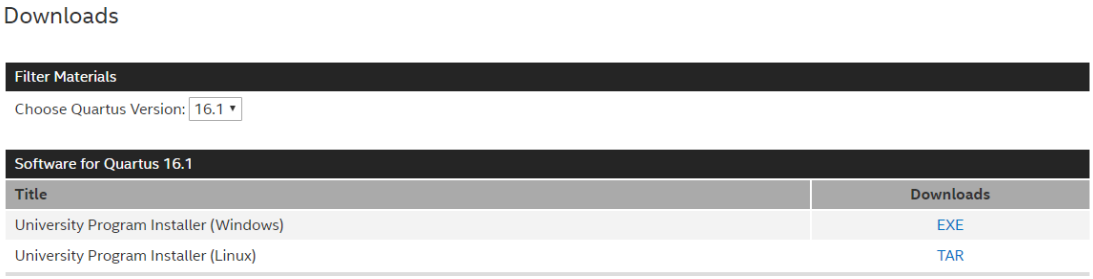

Figure 1: **Monitor Program download page** — Make sure you're downloading version 16.1.

Linux

Extract the tar (`$ tar xf ./intel_fpga_upds_setup.tar`) and
run the installer (`$ ./install_intel_fpga_upds`).
You may need to provide paths to Quartus and to nios2eds. These are in your `~/intelFPGA_lite` directory.
For example, for me I'd provide:
`/home/ekeller/intelFPGA_lite/16.1/quartus/`
`/home/ekeller/intelFPGA_lite/16.1/nios2eds/`
when prompted by the installer.

As described in `usb_blaster.txt`, copy the following into a new file in `/etc/udev/rules.d/51-usbblaster.rules` (as root):

```
# USB-Blaster
SUBSYSTEM=="usb", ATTRS{idVendor}=="09fb", ATTRS{idProduct}=="6001", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="09fb", ATTRS{idProduct}=="6002", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="09fb", ATTRS{idProduct}=="6003", MODE="0666"

# USB-Blaster II
SUBSYSTEM=="usb", ATTRS{idVendor}=="09fb", ATTRS{idProduct}=="6010", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="09fb", ATTRS{idProduct}=="6810", MODE="0666"
```

You'll then need to restart udev (or reboot your computer):

1. `$ sudo service udev restart`

2. `$ sudo udevadm control -reload-rules`

3. `$ sudo udevadm trigger`

You should be able to run the Monitor program:
`$ cd ~/intelFPGA_lite/16.1/University_Program/Monitor_Program/`
`$ ./bin/altera-monitor-program`.

Windows

Run the downloaded EXE, and follow the provided install instructions. If needed, consult Section 2 of the Monitor Program FPGA Nios II Tutorial: `ftp://ftp.intel.com/Pub/fpgaup/pub/Intel_Material/16.1/Tutorials/Intel_FPGA_Monitor_Program_NiosII.pdf`.

Running the Program

Once you have installed both Quartus and the Monitor Program, follow the tutorial in Section 3 of the Monitor Program FPGA Nios II Tutorial to learn how to create a new project: `ftp://ftp.intel.com/Pub/fpgaup/pub/Intel_Material/16.1/Tutorials/Intel_FPGA_Monitor_Program_NiosII.pdf`.

Follow the tutorial and create a new Simple Program (in Assembly), and verify that it runs on your DE10-Lite, and that you can flip the switches and see the corresponding LEDs change. You will modify the Simple Program for Part 1b of your project next.

If you have trouble with the USB driver (on either Windows or Linux), please consult the Troubleshooting guide on D2L: under Assignments/ProgrammingProject1/usb-blaster.pdf

What to submit Please take a screen capture of the Intel FPGA Monitor Program window (the windows snipping tool is good for this, or search Google for your specific platform). To include your name in the submission, there is a search box (as shown below) – please write your name in that, and then take the capture (the whole program window, not just the crop as I have done). Save the file as screecap.png.
Submit as a single file on D2L, under Assignment 1a.

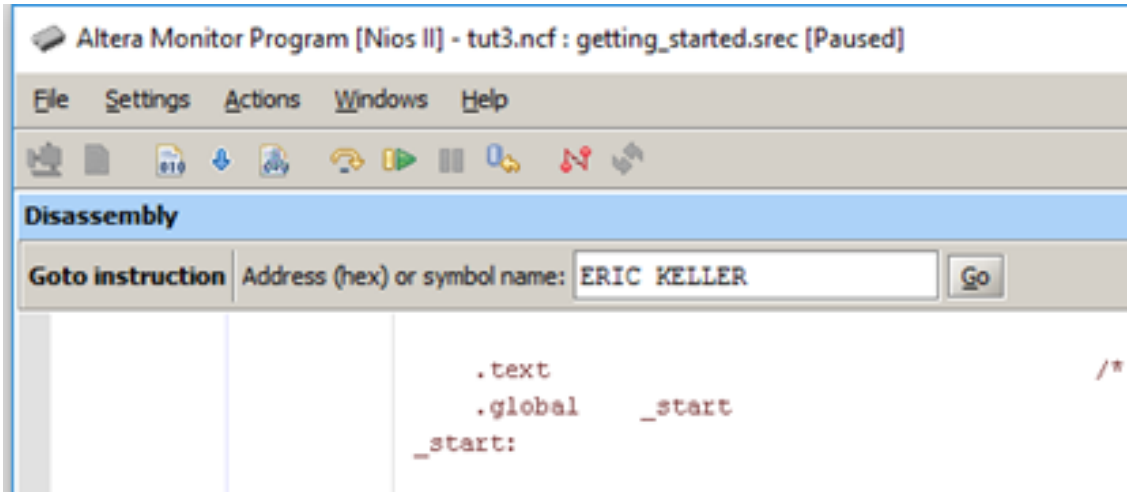

Figure 2: **Screen Shot of Intel FPGA Monitor Program with my Name Added**.

Part 1. Basic Adder (Due February 13, 2019 as part of Assignment 1b)

The DE10-Lite has 10 switches along the bottom right of the board. In the Simple Program, changing the switches directly changes the LEDs. In this project, we will extend this program to implement an *add* operation.
We will split the switches in half: the leftmost 5 switches will represent a binary number, while the rightmost 5 switches represent a second binary number. The LEDs will display a binary representation of the *sum* of these two numbers.

For example, if we wanted to add the numbers 3 and 5, the board should look as follows:

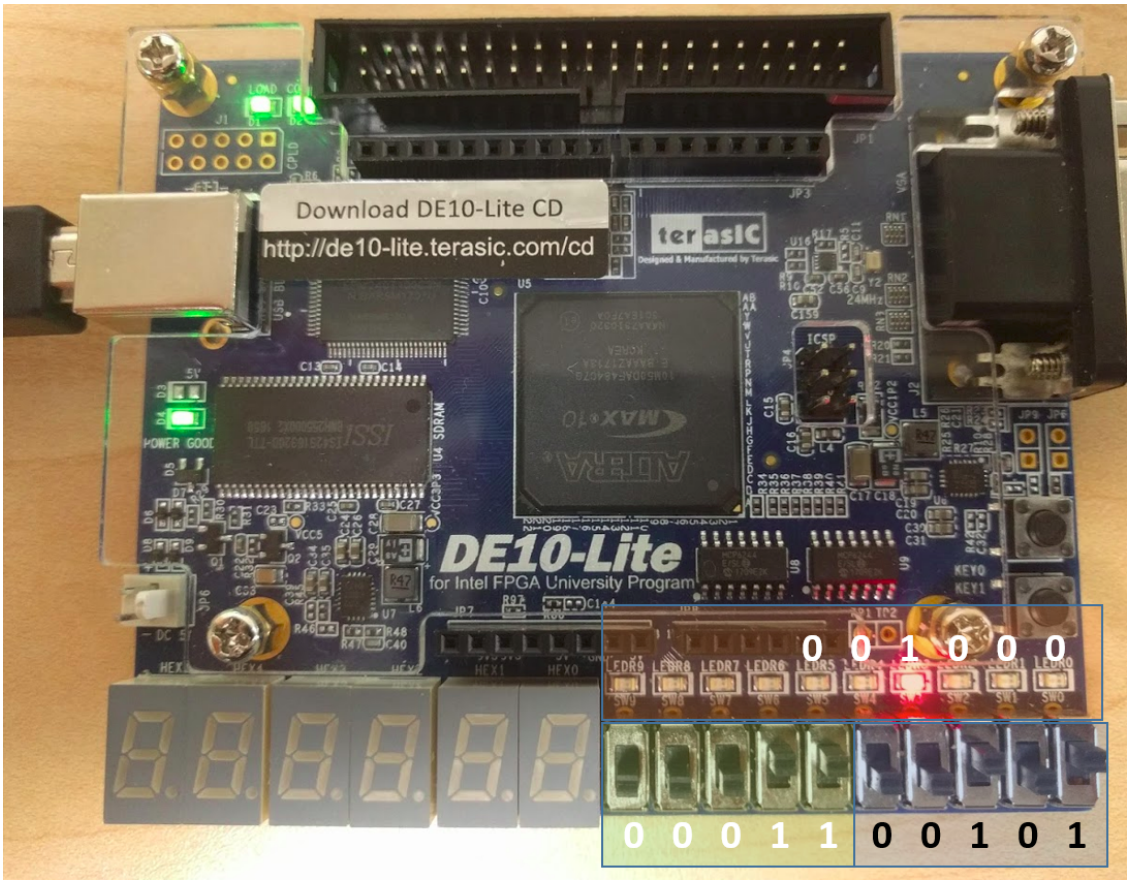

Figure 3: **Adding 3+5** — The leftmost five switches encode the binary number 3 (0b00011) and the rightmost five encode the number 5 (0b00101). The sum of these numbers (8) is displayed in binary on the LEDs: 0b001000.

You should write this program in assembly. You may use the Simple Program as a starting point - don't worry about how the program is reading the state of the switches for now (or writing the LEDs), that code is provided for you in the Simple Program (reproduced below).

You should assume `r4` contains the value of the switches (in a single register), and you must modify it accordingly to treat it as two 5-bit numbers that you then add together. Interpret the switches and LEDs in big endian, with the most significant bit on the left.
Try different switch configurations to verify that your interpretation of the switches and adding is correct.

You can base your code on the following example (from simple_program.s):

```
    .text
    .equ    LEDs,       0xFF200000
    .equ    SWITCHES,   0xFF200040
    .global _start
_start:
    movia   r2, LEDs        # Address of LEDs
    movia   r3, SWITCHES    # Address of switches

LOOP:
    ldwio   r4, (r3)        # Read the state of switches

    # <Your code to modify r4 here>

    stwio   r4, (r2)        # Display the state on LEDs
    br      LOOP

    .end
```

What to submit Submit on D2L in Dropbox (under Assignment 1b) a `adder.s` file that:

1. Takes two 5-bit binary numbers from the switches and adds them
2. Displays the resulting binary number on the LEDs

Part 2. Understanding assembly (Due February 13, 2019 as part of Assignment 1b)

2.1 Interpreting code

Below is some assembly code. Describe what it does, both in plain English, as well as provide some C code which performs the same functionality. We're not looking for C code that will compile to this assembly (it won't, as there is no main function). Instead, provide C code that shows the variables, how they are initialized, and what the code looks like.
The below program is a fully working assembly program—you can load it into the Altera Monitor Program to see how it works. To do that, create a new project of type assembly, and then instead of including a getting started example, you'd skip that part and when it gets to add a file, add this file (you have to save it as a .S file and put it into your project's directory first). You can download this file from Assignment/ProgrammingProject1/p1q2.S.

```
    .include "nios_macros.s"
    .global _start
_start:
    /* the following two instr. (orhi and ori)
       are what movia converts to */
    orhi    r2, r0, %hi(X)
    ori     r2, r2, %lo(X)
    movia   r3, Y
    movia   r4, N
    ldw     r4, 0(r4)
    add     r5, r0, r0
LABEL:
    ldw     r6, 0(r2)
    stw     r6, 0(r3)
    addi    r2, r2, 4
    addi    r3, r3, 4
    subi    r4, r4, 1
    bgt     r4, r0, LABEL
STOP:
    br STOP
.data
N:
    .word 6
X:
    .word 5, 3, -6, 19, 8, 12
Y:
    .word 0, 0, 0, 0, 0, 0
```

2.2 Decoding Instructions

Translate the following hex value to Nios II assembly code.

`993ff915`

2.3 Encoding Instructions

Convert the following assembly into machine code. Your answer should be an 8-digit hex number.

`divu r14,r5,r22`

What to submit Submit on D2L in Dropbox (under Assignment 1b) a plain text file (not a word document or pdf) named `writeup.txt` containing your answers to part 2. You should format this file using this template:

```
1.
   English/C description of assembly
---
2. decoded instruction (e.g. addi r3, r4, 8)
---
3. encoded instruction (e.g. 0xDEADBEEF)
```

Submission Checklist

Assignment 1a

Upload to **D2L** (under Assignment 1a) a gzipped tarball (`.tar.gz`) named `project1a.`*`identikey`*`.tar.gz`. You can make a tarball in Linux with `$ tar czf project1a.identikey.tar.gz ./screencap.png`, or in Windows with 7-zip. The tarball should contain only the following files:

Part 0

An image file named `screencap.png` that is a screen capture of the Intel FPGA Monitor Program running, as described.

Assignment 1b

Upload to **D2L** (under Assignment 1b) a gzipped tarball (`.tar.gz`) named `project1b.`*`identikey`*`.tar.gz`. You can make a tarball in Linux with `$ tar czf project1b.identikey.tar.gz ./writeup.txt ./adder.s`, or in Windows with 7-zip. The tarball should contain only the following files:

Part 1

An assembly file named `adder.s` that adds the left/right switches and outputs the result (in binary) to the LEDs.

Part 2

A text file named `writeup.txt` with your answers to the three short questions, using the template described previously.