# **EEL4713**
# **Lab 0**
Spring 2018

**Overview:** The EEL-4713 assignments this semester will have 3 main sections: setup, laboratory, and deliverables. Setup (section 1) provides information on how to setup software tools and environments, when applicable. Laboratory (section 2) covers practical aspects of this class, including the design of a MIPS microprocessor, writing code for your processor, etc. Deliverables (section 3) covers the desired format of the report for the given and lab and a detailed list of deliverables including a point breakdown.

Please us by reporting inconsistencies or bugs you find in the lab to TAs so we can adjust and improve!

In this assignment, you will perform the following tasks:
Setup:
1. Install Quartus II and ModelSim
2. Quartus TimeQuest tutorial
3. Realterm
4. Verilog Tutorial

Lab:
1. Generic Delay
2. Clock Divider
3. UART driver and clock divider
4. Getting into the MIPS

Deliverables:
1. Lab report

**EEL4713**
**Lab 0**
Spring 2018

## SETUP

1. **Install Quartus and Modelsim**

Become familiar with what FPGA we are using and other technical specs of our system. Skim the manual to see what our system is capable of. Take note of the FPGA size, the FPGA model, AS programming, external memories, and clocking options.

### 2.1 Layout and Components

A photograph of the DE0 board is shown in Figure 2-1. It depicts the layout of the board and indicates the location of the connectors and key components.

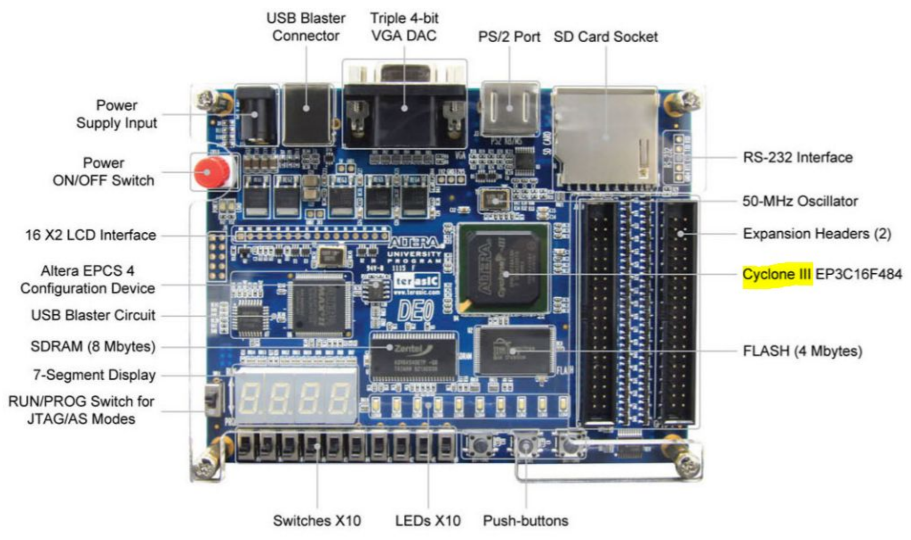
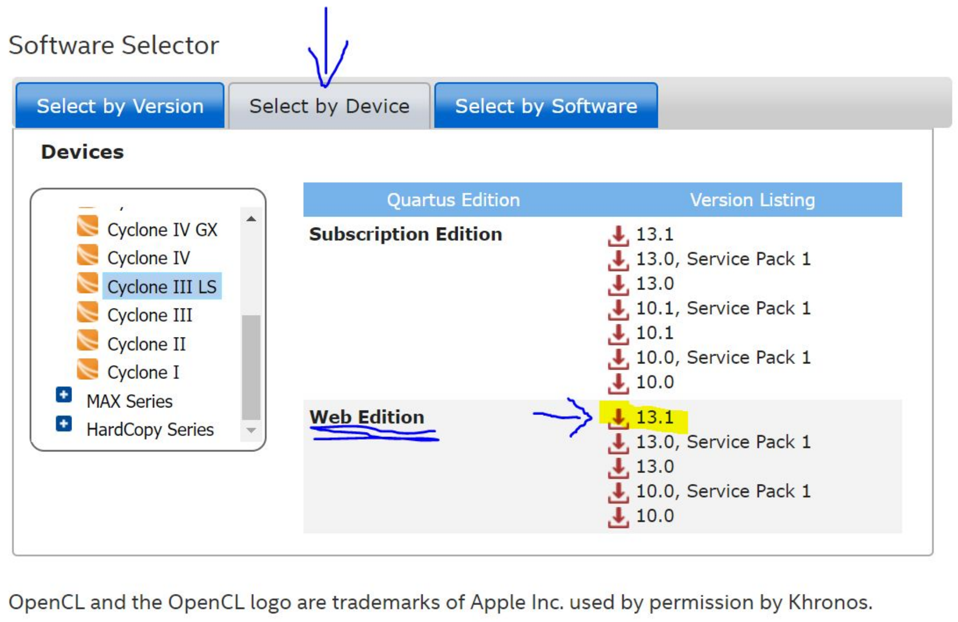


Figure 2-1 The DE0 board.

Go to the link follow image example. Make sure you select the web edition.
https://www.altera.com/downloads/download-center.html

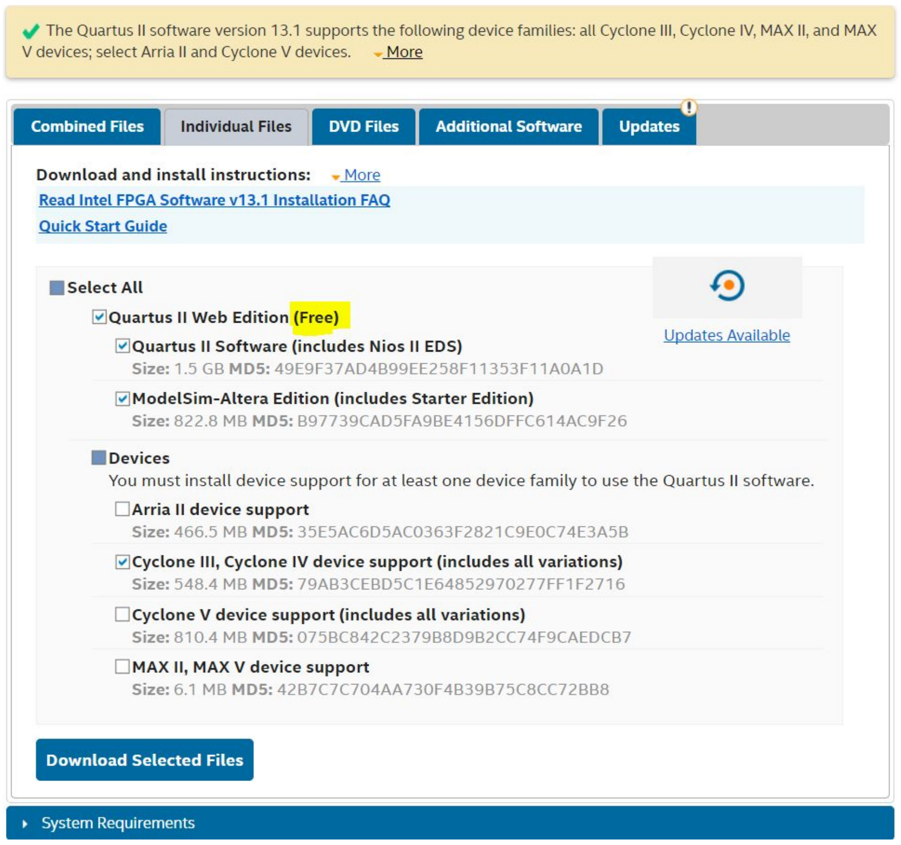
**EEL4713**
**Lab 0**
Spring 2018


Software Selector

| Quartus Edition | Version Listing |
|---|---|
| **Subscription Edition** | 13.1 |
| | 13.0, Service Pack 1 |
| | 13.0 |
| | 10.1, Service Pack 1 |
| | 10.1 |
| | 10.0, Service Pack 1 |
| | 10.0 |
| **Web Edition** | 13.1 |
| | 13.0, Service Pack 1 |
| | 13.0 |
| | 10.0, Service Pack 1 |
| | 10.0 |

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos.

This will take you to the install page. Configure as below.


The Quartus II software version 13.1 supports the following device families: all Cyclone III, Cyclone IV, MAX II, and MAX V devices; select Arria II and Cyclone V devices. More

Download and install instructions: More
Read Intel FPGA Software v13.1 Installation FAQ
Quick Start Guide

- ▨ Select All
  - ✓ Quartus II Web Edition (Free)
    - ✓ Quartus II Software (includes Nios II EDS) — **Size:** 1.5 GB **MD5:** 49E9F37AD4B99EE258F11353F11A0A1D
    - ✓ ModelSim-Altera Edition (includes Starter Edition) — **Size:** 822.8 MB **MD5:** B97739CAD5FA9BE4156DFFC614AC9F26
  - ▨ Devices
    You must install device support for at least one device family to use the Quartus II software.
    - ☐ Arria II device support — **Size:** 466.5 MB **MD5:** 35E5AC6D5AC0363F2821C9E0C74E3A5B
    - ✓ Cyclone III, Cyclone IV device support (includes all variations) — **Size:** 548.4 MB **MD5:** 79AB3CEBD5C1E64852970277FF1F2716
    - ☐ Cyclone V device support (includes all variations) — **Size:** 810.4 MB **MD5:** 075BC842C2379B8D9B2CC74F9CAEDCB7
    - ☐ MAX II, MAX V device support — **Size:** 6.1 MB **MD5:** 42B7C7C704AA730F4B39B75C8CC72BB8

Follow instructions to install. Note another application will need to get installed to install quartus.

I had issues with modelsim when I did a fresh install. Email me if you do as well. Other simulators exist such as Active and Vivado (Vivado is like quartus, but with a built in simulator). You can use these if you wish.

## 2. Quartus Timing Tutorial

This class will require synthesis and functioning hardware similar to Digital Design. For this class, meeting timing constraints will not always be simple. Become familiar with Quartus's TimeQuest Analyzer. FPGA timings are specified with Synopsis Design Constraints (.sdc) files. TimeQuest provides a gui that will write the file for you. You may come to find it isn't quite enough in the future, but it will be fine for now. You are not required to do the tutorial per-say but familiarize yourself with it as you will need to do this later in the lab.

The tutorial can found at https://www.altera.com/en_US/pdfs/literature/ug/ug_tq_tutorial.pdf and the pdf will be saved in the lab0 resources folder. Toward the end of the tutorial, you will come across "false paths". Become familiar with the concept of false paths. There is a seperate pdf explaining false paths in the lab0 resources folder.

In your labs, you will interface with I/O. Thus you will need to handle unconstrained I/O paths. TimeQuest will be able to do this. There is also a discussion thread saved in resources which you may find useful for this.

## 3. Realterm

Download and install realterm. This is a terminal application which allows very detailed control over a serial terminal. We will (ideally) use this to send and receive files to and from our FPGAs.
https://sourceforge.net/projects/realterm/

## 4. Verilog Tutorial

### Synthesis

In the resources folder, there is a pdf labeled introVerilog.pdf which is an older powerpoint from University of Wisconsin. This is a pretty decent reference for writing Verilog. Go through this powerpoint and familiarize yourself with Verilog syntax and data types. I recommend you think of VHDL when learning Verilog. There are many similar constructs which go by different names such as process statements (always blocks), sensitivity list, variables vs signals (reg vs wire), entity (module), etc…

### Testbenches

You can find my two testbenches in the lab 0 folder. These testbenches are really basic. You will need more advanced features for later parts of the course. I added two useful resources to the lab 0 folder.

### Editors

For an editor, I have two recommendations. Sigasi is great. It has a lot of shortcuts and auto-compilation to point out syntax errors, but I found it a lot more useful for VHDL than verilog. Notepad++ is also really nice once you learn the shortcuts. Many people also use vim or emacs because writing this code is done on remote servers via terminals. Emacs has a nice auto formatting feature (beautify) that works well on VHDL, but I haven't experimented with in on verilog.
http://www.sigasi.com/download-f/
https://notepad-plus-plus.org/download/v7.5.4.html

**Important note:**
Similar to C, a verilog if statement defaults on only including the line below it.
I.e.

```
if(rst)
        Example <= 1'b0;
        Temp <= 1'b0;
        …
```

Only "Example" will be assigned 0 on reset in the above code. In C we would use {} to encompass more than the following line. In verilog, we use "begin" and "end".

```
if(rst) begin
        Example <= 1'b0;
        Temp <= 1'b0;
        …
end
```

Now Example and Temp are assigned 0 on reset. Verilog uses "begin" and "end" for pretty much every construct.

## LAB:

We will begin creating some verilog components which will be useful for your future labs and processor design. File structure can be very important for this type of work. I have found a certain organization that is scalable to more complex projects. I strongly recommend you follow the below file structure for this class:

lab0
        ip //folder to hold ip source code, simulations, and sdc files
                src //source
                        delay //folder holding delay ip src
                                delay.v //verilog source for delay (this file is a wrapper that can call any implementation of delay; this is a good idea since it provides an abstraction layer between instantiation and implementation)

delay_r0.v //verilog source which holds the actual implementation of delay

sim //testbenches
delay_tb //folder for delay simulation test code, good because can be multiple files
delay_tb.v //testbench code for delay.v module

sdc //sdc files for the project
lab0.sdc

q //quartus project directory

ms //modelsim project directory

## 1. Generic Delay

In the lab0 folder you will find my implementation of delay_r0.v which is heavily commented. Familiarize yourself with this code as many of the concepts will be important for you to use in future labs. Specifically passing arrays, loop unrolling, and registering values.

## 2. UART Driver and clock divider

If you aren't familiar with UART, get familiar and be able to draw the timing diagram. There are resources in the lab0 resource folder.

Similar to the generic delay, you will find my implementation of a simple UART driver in the lab0 folder. Familiarize yourself with this code as well. Know how to do state machines in verilog.

Once you are comfortable with this code, you should have all the main tools of verilog synthesis under your belt. You should synthesize this code. To synthesis this code you will need to use the timequest analyzer to make an sdc file. You can find mine for reference in the lab0 folder for your reference on what it should look like. It is not intuitive. Be able to make an sdc file. Warnings will arise when there are errors in your sdc file.

When doing pin planning, reference your manual. I recommend putting the rx_reg on LEDs so you can see the values. I set my tx_reg to I/O pins, and my rst, send, and read to switches.

**Not required:** To test UART in hardware, use a TTL-Serial to FTDI USB converter board.
https://www.banggood.com/FT232RL-FTDI-USB-To-TTL-Serial-Converter-Adapter-Module-For-Arduino-p-917226.html?cur_warehouse=USA
https://www.amazon.com/HiLetgo-FT232RL-Converter-Adapter-Breakout/dp/B00IJXZQ7C/ref=pd_lpo_vtph_147_bs_lp_t_1?_encoding=UTF8&psc=1&refRID=XY4SY1FDN8YPKSX0FAM3

Note that my sdc file will still fail timing analysis (this might vary a little), but I have tested my code in hardware and it works.

**Q:** Where does timing analysis fail on my sdc file (see "Timing Closure Recommendations" in the quartus project)? Is that somewhat expected and why (hint what causes most timing analysis failures in well made FPGA designs)? Why does my project still work? What user action would make my project no longer work?

**Extra Credit:** My implementation doesn't implement parity. Implement 4 parity checks on rx and tx and test with a realterm terminal on your computer. Make a separate module than the uart_basic and conditionally instantiate it in uart.v module with ARCH_SEL. Rewrite as little as possible. Demo for extra credit.

## 3. Getting into the MIPS

In this section, you will start building useful components for your MIPS.

### a. Muxes

Your MIPS will do a lot of muxing so it'll be convenient to have a dedicated muxing component. Create a generic mux by passing arrays as long vectors as shown in the delay module. Your should take an array of generic bit width (BIT_WIDTH) and depth (DEPTH). Pass in the select width (SEL_WIDTH) but set the min value possible (based on depth) as the default. Do this with the log2 function used in the example code.  Of course make a testbench and test your rtl. Also look at the rtl viewer to see what is being synthesized. Name is component "mux.v".

This component will be particularly useful throughout your processor and future designs.

### b. Reg File

The MIPS has 32 distinct 32 bit registers with some basic control logic which is referred to as the register file. This should be made generic, i.e. passing vectorized arrays like with the mux.

See lab0 resource on the register file. Read about the register file. There is one register which has a special property. This must be implemented.

You should be able to read to both registers and write to a register. Of course make your own testbench and verify hardware with the rtl viewer. Name component "registerFile.v".

You may use my generic delay entity or you can make your own version (maybe without the array packing and unpacking, since it isn't really needed as mentioned already in example code) or use neither.

Use your generic mux which you made in part a for muxing. Making a decoder module isn't necessary. It can be as little as a single line of code.

Match the below module instantiation.

```
NOTES
        DATA_WIDTH //world length
        RD_DEPTH //number of parallel reads
        REG_DEPTH //depth of the register file
        ADDR_WIDTH //address width, for the MIPS the operands are addresses, so operand
width
        rst //synchronous reset
        wr //write enable
        rr //reg read address as vectorized array
        rw //reg write address
        d //input data to be written to reg
        q //output data from reg reads, vectorized array

        reads are always enabled

module registerFile #(
        DATA_WIDTH = 32,
        RD_DEPTH = 2,
        REG_DEPTH = 32,
        ADDR_WIDTH = log2(REG_DEPTH)
)(
        input clk,
        input rst,
        input wr,
        input [ADDR_WIDTH*RD_DEPTH-1:0] rr,
        input [ADDR_WIDTH-1:0] rw,
        input [DATA_WIDTH-1:0] d,
        output [DATA_WIDTH*RD_DEPTH-1:0] q
);
```

**Q:** What should happen if you read and write from the same register in the same clk cycle?

## DELIVERABLES:

Please note that this is a senior level elective class. You need to be doing all the work assigned even if there is no direct deliverable for it. It will be important later and I will assume you have done it. For instance, there is no Quartus deliverable. But if you don't have it up and working, you can't do the rest of the lab or future labs. If you are hardworking and don't take shortcuts, you will do fine on the labs. If you take shortcuts, you will be miserable when things don't work.

For this lab, you will submit a zipped file of your verilog code and a document with your answers to lab questions on them. Be prepared to talk about code and lab questions in the lab section. I.e. zip up src/ and sim/

| Deliverables | Grade |
|---|---|
| Questions throughout lab doc | 25% |
| Generic Mux | 25% |
| Register File | 25% |
| In person questions/tasks | 25% |

| | |
|---|---|
| Extra Credit | 15% |