# Digital Computer Design Lab Manual

Dr. Melissa Smith
Edited by: Brad Green
Clemson University
Holcombe Department of Electrical and Computer Engineering

August 21, 2019

## Contents

## 1 Introduction

This document constitutes the lab manual associated with ECE3270 Lab. In it you will find helpful information as well as important policies for the lab portion of the course. Please refer to this for any questions you may have before contacting any of the TAs. None of the labs will be included with this manual, they will be posted on Canvas whenever assigned as they change each semester.

### 1.1 About this Lab

The purpose of this lab is to give you hands-on experience working with FPGAs for hardware prototyping and algorithm acceleration. You will gain proficiency in both VHDL and OpenCL during this semester.

### 1.2 Required Materials

There are no required textbooks for this lab, all required information will be given through Canvas. However, you must purchase a microSD card for use in the lab. 4GB is the minimum recommended size for these cards. You will also need a USB card reader as none of the computers in the labs have a built in reader. Below are some recommendations.

8GB SD Card: $5.88 at the time of this writing.

USB Card Reader: $7.77 at the time of this writing.

32GB card/reader combo: $20.72 at the time of thisUniversity UnionUniversity Union writing. This is a much faster card (U3 class) and includes the necessary USB card reader.

Anything UHS, U3, or “Designed for 4K video” will be much faster and more usable after this class (such as in a phone or other devices). If for any reason you are unable to purchase an SD card, please talk to one of the TAs and we will discuss alternative options.

You will be **UNABLE** to write to these cards via Windows or Mac, and you will be unable to write the image on these cards unless you have a Linux system (not a VM) with root privileges. Because of this, you will need to **give your card to a TA for imaging**. This must be done by the **END OF THE SECOND FULL WEEK OF CLASSES**. If you turn your card in later than this, it may not be able to imaged in time for you to complete your projects and this is not an acceptable excuse for incomplete projects.

If you do have a compatible machine (native Linux with root privileges), you can attempt to image your card yourself. To do this, you need to download the image file found HERE. Extract this file and browse to it’s location in a terminal window. Plug in your SD card and type `dmesg | tail` to determine which device the card mounted. You should expect to see something like `/dev/sdd` or `/dev/sdc`. This is **EXTREMELEY IMPORTANT!** Failure to note the correct drive can result in you accidentally wiping your HDD. After you complete this, you simply type `sudo dd if=8gbNew of=/dev/sdd bs=1M` and make sure to replace the output file with the location of your SD card found earlier. When this completes, type `sync` to make sure the buffer is empty and your card is written completely.

If you have problems with the above instructions, just give your card to a TA to allow them to image it.bbbo

## 2 Lab and Office Space

Room 309 has 20 computers and is where we will conduct demonstrations. This room is also shared with other classes, so please respect TAs or other students working in this lab. If another class needs to demo, please make sure to give them computers if necessary.

**DO NOT** disconnect Ethernet cables or mice from **ANY** computers in the lab space.

### 2.1 TA Offices

TAs can be found in Riggs 10 in the basement. At least one of the TAs is usually available in the lab during normal business hours. If neither are available, then make sure to send an email.

## 3 Lab Communication

For all lab material related questions, please post in the corresponding Discussion Board. Do not email one of the TAs or Dr. Smith unless you have a question not relating specifically to a lab.

### 3.1 Discussion Boards

For each lab, there will be an associated Discussion Board post on Canvas. This is where you should pose all questions regarding the labs, which includes any confusion about projects, clarifications about specifications, or anything else regarding assistance with a project. One of the TAs will monitor this discussion board and respond to questions there. Any questions sent via email to Dr. Smith or any of the TAs may go unanswered.

This process is to prevent duplicate questions and an overwhelming number of announcements from TAs. You may also use the discussion board to communicate with other students and browse previous questions. Always check previous posts before submitting a question of your own.

### 3.2 Email

The TA email addresses are listed below. You can email us directly or contact us through Canvas. Please reserve direct email communications for **non-project related questions only**. And please contact the TAs before you send an email to Dr. Smith, the email will be forwarded if necessary.

- Brad Green: wbgreen@g.clemson.edu
- Ankit Kulshrestha: akulshr@g.clemson.edu

### 3.3 Canvas

Any important information will be sent out via Canvas. If you are not receiving emails for announcements, first check your notification settings in Canvas.

## 4 Lab Expectations

You are expected to complete every lab during the semester. As specified in the syllabus, failure to do so will result in a failure for the lab and an **F** in the class. You will also be expected to demonstrate proper functionality of the project to a TA during a predetermined time period.

### 4.1 Attendance

Attendance is required only on Demonstration days for this lab (labs are considered incomplete without a demo unless specified otherwise). These demo times will be clearly relayed to the class via Canvas. Tutorial and help sessions will be given during demo times on some weeks when a lab is not due. While optional, these sessions are recommended as one of the TAs will walk through important lab information and assist with the currently assigned project.

If you are unable to make it to the determined demonstration time, please contact a TA with a valid excuse and schedule another meeting time.

## 5 General Lab Practices

### 5.1 Lab Reports

For each lab, you are required to complete a lab report using the given LaTeX template. You can find this here, https://www.overleaf.com/read/vtqywnmrtgjy or on Canvas. It is suggested to use Overleaf. If you sign up with your Clemson Email via the Google sign in, you will receive access to the Clemson license for overleaf.

All lab reports are submitted on Canvas via Turnitin as a PDF. **Any file format other than a PDF will not be accepted!** Project files will be submitted separately to the Assign server.

#### 5.1.1 Report Guidelines

Below are the report requirements. Please note, you must include every section, even if said section is empty for a given project.

1. Your lab report should consist of the following:
    - Cover Sheet
    - Abstract
    - Body of the Report (in appropriately named sections)
    - References
    - Appendices (multipage timing diagrams, etc.)
2. **Cover Sheet:** The cover sheet should follow the format similar to the sample cover sheet in the template.

3. **Abstract:** The abstract is a brief (50-150 words) summary of your report. The project goals, major results, and/or conclusions should be stated in the abstract. The abstract should occupy a single page with the centered title, “ABSTRACT”.

4. **Body of the Report:** The body of the report should begin with an INTRODUCTION and end with a CONCLUSIONS section that includes lessons learned. The remainder of the report should be divided into several named sections as appropriate. You should have sections titled according to what is described in that section, e.g. “Design of a 4-bit Comparator Circuit”, “Test of a 4-bit Comparator Circuit”, etc. Note: You should never have a section titled “Body of the Report”. Include block diagrams, circuit diagrams, state diagrams, truth tables, Boolean equations, etc. as appropriate (i.e. what you used to develop and explain your design) complete with discussion of these figures as appropriate to the project (you may choose to put figures in an appendix as discussed below). At a minimum, the report discussion should include theory of operation, testing strategy, problems encountered, lessons learned, etc. If things are not working correctly, also include a discussion describing what you think is wrong and your strategy for correcting.

5. **References:** As appropriate, include a list of reference material used in your design development and preparing your report (i.e. your textbook, any and ALL internet sources, conference/journal papers, etc.). References should be listed in the order they were cited in the report and should be in standard IEEE format (see the IEEE Information for Authors document on Canvas). The Preparation of Papers for IEEE TRANSACTIONS and JOURNALS document on the Canvas also provides good examples. The references should occupy one or more separate pages with the centered title “REFERENCES”. References that do not follow these conventions will be graded as completely incorrect.

6. **Appendices:** Diagrams, programs, tabular data, etc. may be included as appendices or may be contained within the body of the report. Any data that spans more than one page should be included in an appendix. Construct multiple, titled, and lettered appendices as appropriate, e.g. “Appendix A VHDL Code Segments for a 4-bit Comparator”, “Appendix B Timing diagram for a 4-bit up/down counter”, etc.

7. **Figures:** Figures should be legible and have an appropriate figure number and caption. Logic diagrams and timing diagrams may be screen captured and imported as figures where appropriate; all diagrams should be properly annotated in the figure and described in the text body. Logic and/or timing diagrams may appear in landscape mode to enhance readability. All information in logic diagrams and timing diagrams should be labeled appropriately. The IEEE Information for Authors document on Canvas is the source for describing proper formatting and referencing of figures (do not put a figure in your report that is not properly referenced and described in the text). Figures that do not follow these conventions or are unreadable will be graded as completely incorrect.

8. **VHDL Source Code:** All VHDL source code must follow coding conventions presented and discussed in class. Code that does not follow these conventions will be graded as completely incorrect. All code must be submitted to the Assign server as described in the following subsection. Do NOT include your entire code in your report but you may use code segments to describe implementation and operation where appropriate.

9. **Timing diagrams:** Remember, timing diagrams are meant to convey information to your reader about the functionality of a given design. Related signals should be grouped together to enhance readability. For example, clock and clock related signals should be grouped together and placed at the top of the timing diagram. Controls signals should also be grouped together with the data signals they control. Timing diagrams should be sized so that it is easy to read appropriate information from the timing diagram. Long timing diagrams should be split into multiple figures as appropriate. Annotations should be added to these diagrams to indicate important test vectors, modes of operation, etc. Timing diagrams that do not follow these conventions or are unreadable will be graded as completely incorrect.

10. All pages are to be numbered with the exception of the cover sheet and abstract page.
11. All reports should be free of grammatical and typographical errors. Proof read your report and use an appropriate LaTeX processor with spell and grammar check capabilities.

## 5.2 Code Submission

All code for each lab must be submitted to the Assign Server using the Assignment number given in the lab directions. This includes all code required to make and run your code (i.e. testbenches and any code written for testing). We must be able to compile your code; missing files will result in a penalization.

### 5.2.1 Submission Instructions

You **MUST** follow the below instructions carefully when submitting your code. Failure to do so correctly will result in a failed submission.

1. You will send your email to ece_assign@clemson.edu
2. You **MUST** use the following subject (updated for later projects).
    - <CourseName>-<Section>,#<AssignmentNumber>
    - For Example: ECE3270-1,#1
    - Note there are **NO SPACES!!!**
3. If you wish to leave a note with your submission, **do not write it in the body of the email**. Attach it as a .txt file!
4. You can only submit files of the following types: **.vhd, .vht, .c, .h, .cpp, .txt, .xml**

## 5.3 Grading

Each lab report will be graded as follows:

- Report: 50%
    - Proper Format
    - All sections included
    - Valid Images where applicable
    - Proper grammar, punctuation, and spelling
- Demo: 40%
    - Live Demonstration of working design
    - Questions posed by TA
    - Proper Modular Design, which includes thoughtful signal and I/O names
    - Commented code.
- Proper Assign Server Submission: 10%

This rubric may be modified for specific projects and only serves as a guideline for all projects in this lab.

## 6 Hardware

Below you will find instructions for setting up and programming the FPGA for use during the lab.

### 6.1 Standard Use

These instructions show how to program the FPGA for use with the on-board components without OpenCL.

1. Verify you have the USB cable plugged into the USB Blaster II port on the FPGA and power is connected and powered on (Figure 1. If this is the first time plugging in the device, it may take a few minutes for your computer to recognize the hardware and correctly install the driver. Be patient until the device installation pop-up either goes away or says the device is installed.

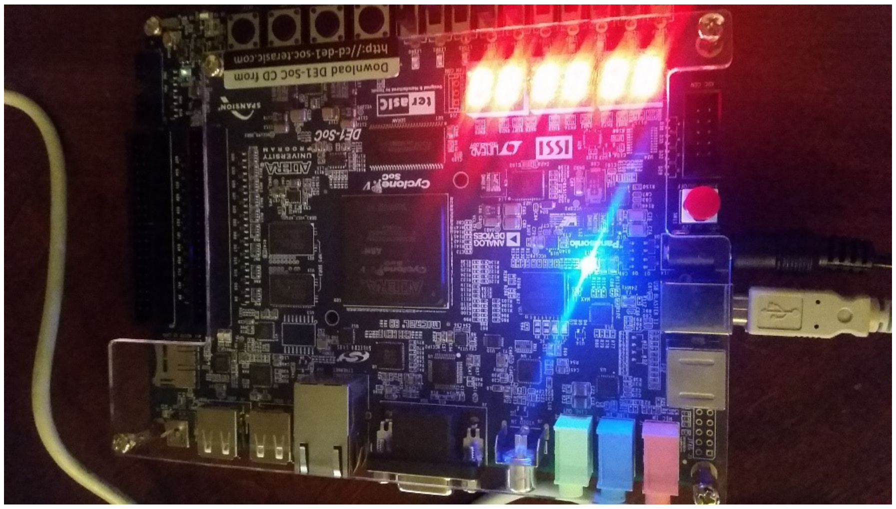


Figure 1: Correct installation of USB Blaster II Cable

2. You will need to verify the device is set up for JTAG programming. To do this, look on the bottom of the FPGA for a set of switches. You will set them to **110010** from the highest to lowest order, as shown in Figure 2.

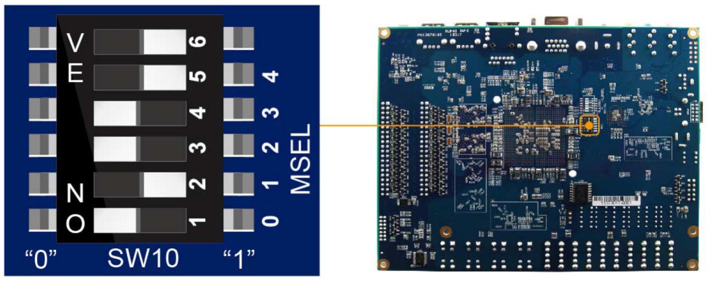


Figure 2: Correct switch setup for JTAG programming.

3. When installed, you can open the programmer, Figure 3, in Quartus and it should automatically detect the device when opening. Otherwise click the “Hardware Setup” Button shown in Figure 4 and then

select the DE-SoC device in the drop-down list, Figure 5.

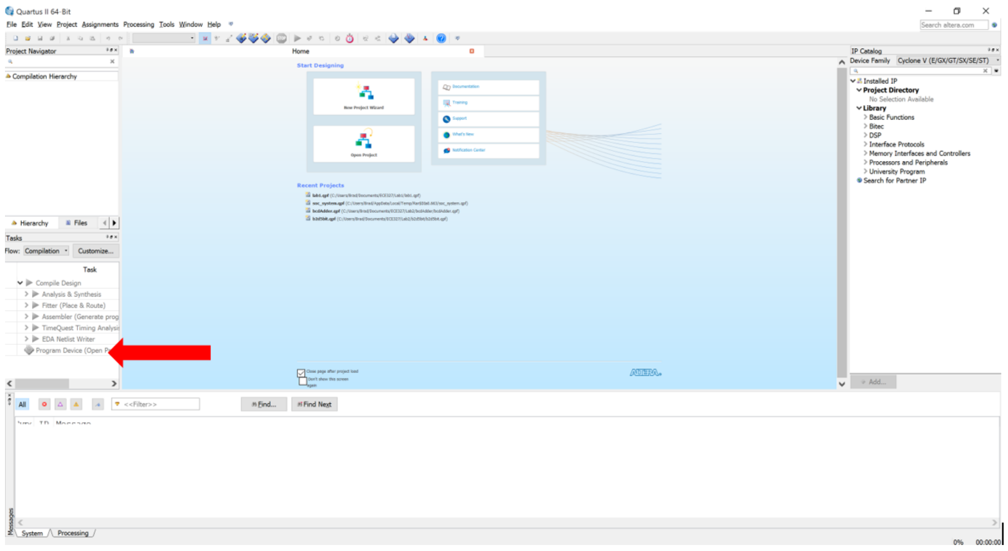


Figure 3: Double-click Program Device to open the Programmer

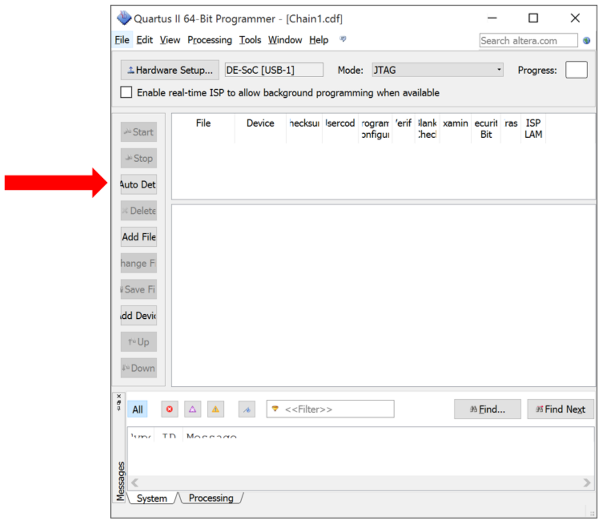
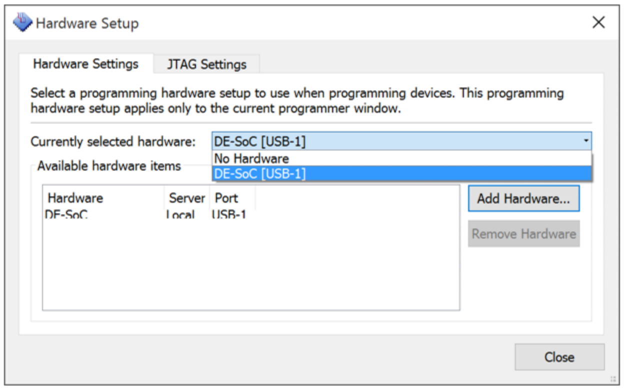


Figure 4: This is the programmer window. If you do not see the DE-SoC device as above, you will need to click "Hardware Setup"


Hardware Setup

Hardware Settings | JTAG Settings

Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: DE-SoC [USB-1]

No Hardware
DE-SoC [USB-1]

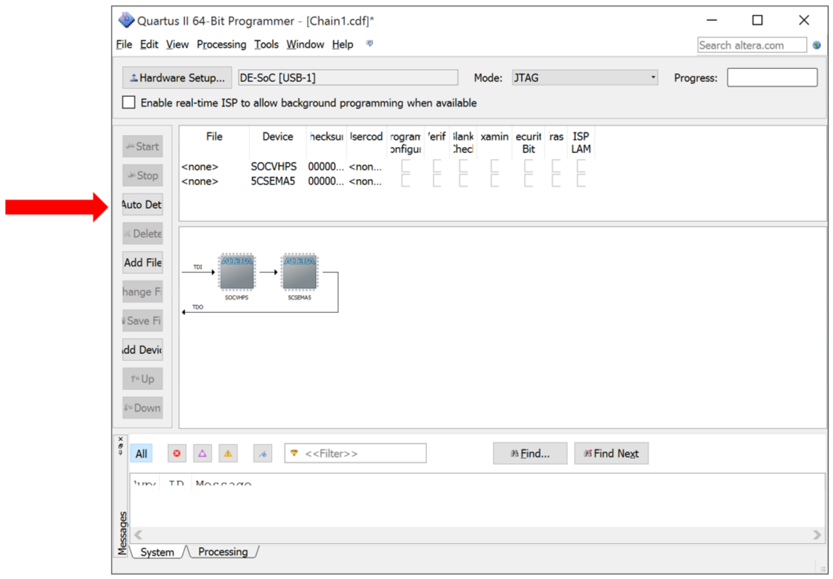
Available hardware items

| Hardware | Server | Port |
|---|---|---|
| DE-SoC | Local | USB-1 |

Add Hardware... | Remove Hardware | Close

Figure 5: Device you should select, leave other options as is.

4. Click “Auto Detect” in the programmer window once your device has been selected. You will need to select the 5CSEMA5 device. You should then see multiple blocks appear as in Figure 6. The 5CSEMA5 is the FPGA and the SOCVHPS is the ARM controller which you will not need to be concerned with as of now.


Quartus II 64-Bit Programmer - [Chain1.cdf]*

File Edit View Processing Tools Window Help

Hardware Setup... DE-SoC [USB-1] Mode: JTAG Progress:

Enable real-time ISP to allow background programming when available

| File | Device | hecksu | lsercod | rogran onfigu | 'erif | ilank Chec | xamin | ecurit Bit | ras | ISP LAM |
|---|---|---|---|---|---|---|---|---|---|---|
| <none> | SOCVHPS | 00000... | <non... | | | | | | | |
| <none> | 5CSEMA5 | 00000... | <non... | | | | | | | |

Figure 6: View after clicking ”Auto Detect” and selecting the correct device.

5. Next, you need to add your programming file to the 5CSEMA5 device. To do this, right click the 5CSEMA5 device and select “Edit->Change File”as shown in Figure 7.

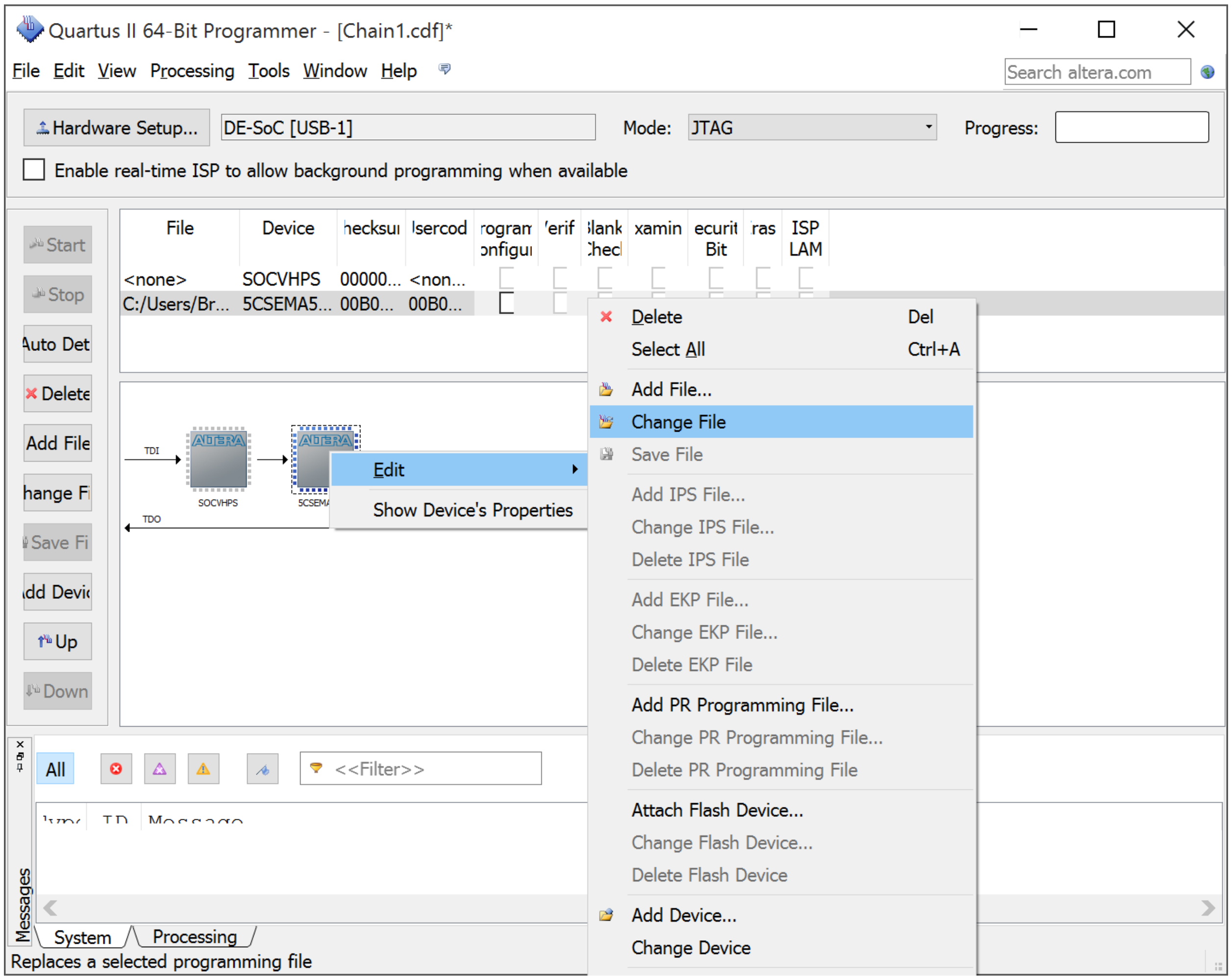
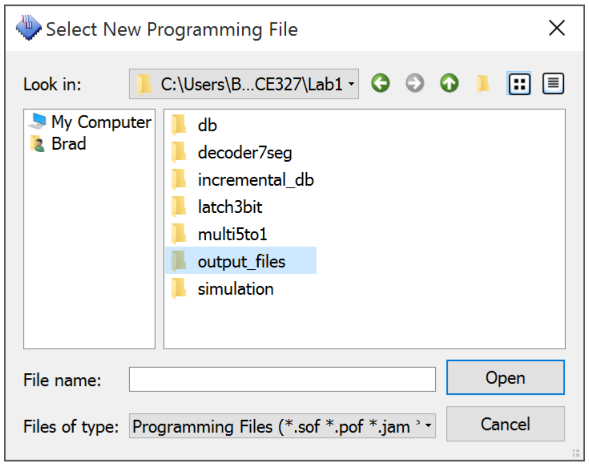


Figure 7: Adding a file to program the FPGA.

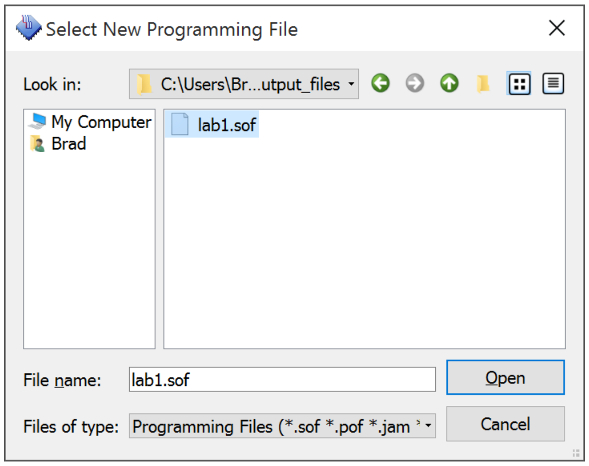
6. The file you are programming to the FPGA can be found in the “output_files” folder inside of your projects working directory, as shown in Figure 8. Select the .sof file within the folder as shown in Figure 9.



db
decoder7seg
incremental_db
latch3bit
multi5to1
output_files
simulation

File name:
Files of type: Programming Files (*.sof *.pof *.jam

Figure 8: Output file folder in working directory.

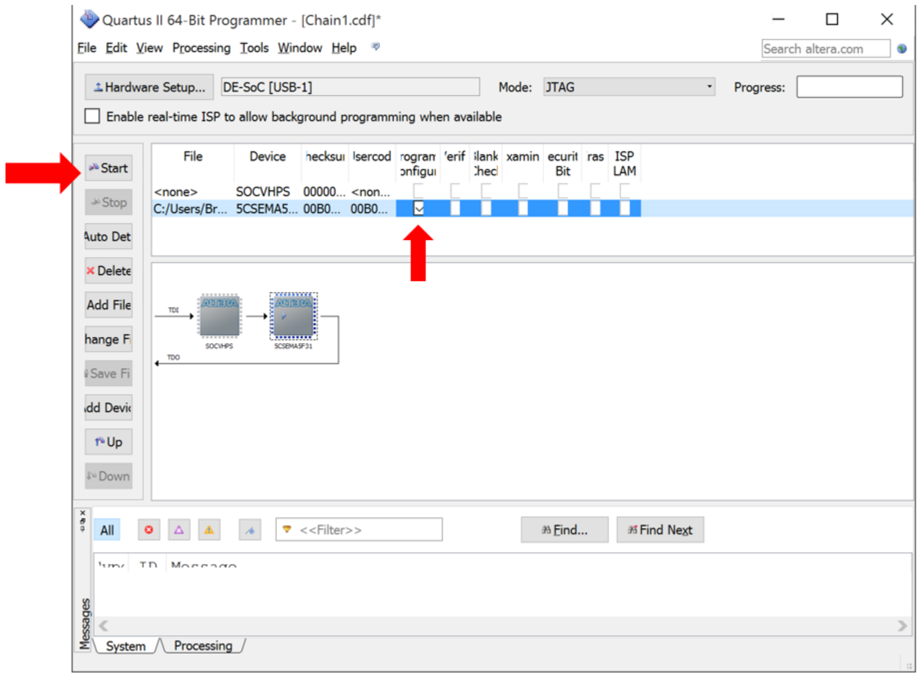

Select New Programming File
Look in: C:\Users\Br...utput_files
My Computer
Brad
Open
Cancel

lab1.sof

File name: lab1.sof
Files of type: Programming Files (*.sof *.pof *.jam

Figure 9: Example .sof file for programming.

7. You may need to select “Program/Configure” in the device chain as shown in Figure 10. Then select “Start” and your device should program!



DE-SoC [USB-1] Mode: JTAG Progress:

| File | Device | hecksu | lsercod | rogran onfigu | 'erif | lank hec | xamin | ecurit Bit | ras | ISP LAM |
|---|---|---|---|---|---|---|---|---|---|---|
| <none> | SOCVHPS | 00000... | <non... | | | | | | | |
| C:/Users/Br... | 5CSEMA5... | 00B0... | 00B0... | ✓ | | | | | | |

Figure 10: Verify ”Program/Configure” is selected and the click ”Start”.

## 6.2 OpenCL Use

Below you will find instructions for setting up the FPGA for OpenCL use as well as instructions for compiling programs and running them on the ARM core. You MUST use the ULLAB machines to properly use OpenCL. Please note these instructions are for editing and compiling a full OpenCL project. Some of the instructions in Section 6.2.1 are not applicable when only programming for libraries.

### 6.2.1 Programming and Compiling your Kernel

1. You will receive a folder structure containing a bunch of files, including a skeleton for your OpenCL Kernel. The kernel will be located at `aocl_PROJECT/device/PROJECT.cl` inside the root folder. **DO NOT MODIFY THE FOLDER STRUCTURE**.
2. Edit the *.cl file with your favorite text editor if the project requires it.
3. Browse to the emulation folder if you would like to test your project before spending time compiling. Here, modify the `run.sh` file, if necessary, to change running parameters if you prefer a different test case. Simply type `./run.sh` to make and execute your program for emulation. **DO NOT EXECUTE MAKE OUTSIDE OF THE RUN SCRIPT. IT WILL FAIL!** This will only test the functionality of the host with a C kernel.
   (a) Keep in mind emulation will not work for all projects. When using Libraries, your VHDL code **WILL NOT BE TESTED**.
   (b) Specifically, emulation is for testing the kernel and the host code ONLY. Library files using VHDL cannot be emulated!
4. Once you are ready to compile, you need to open a terminal window and browse to this location and type `make`.
   (a) It may be useful to note that the Makefile includes a `resource` option! You can use this first to verify your design will fit on the FPGA before wasting time compiling.
5. Below are a few tips that may be useful when compiling OpenCL.
   (a) Errors generated by OpenCL can be difficult to troubleshoot. Check if you are directed to a log file first. If not, then type the command `cat PROJECT/quartus_sh_compile.log | grep Error` to see the Quartus generated errors.
   (b) When working with Libraries projects, make sure you copy your your .vhd file into the device folder.
   (c) You will also need to make sure every file for libraries is included in the .xml file before compiling.

### 6.2.2 Programming and Compiling Host Code

1. Host code will be located at `aocl_PROJECT/host/src/main.cpp` inside the root folder. **DO NOT MODIFY THE FOLDER STRUCTURE**.
2. Edit the *.cpp file with your favorite text editor.
3. Once you are ready to compile, you need to open a terminal window and browse to `aocl_PROJECT` and type `make`. This will automatically compile the host code with the ARM cross compiler. The generated executable is required to run your code on the DE1 Board.

### 6.2.3 Copying Files to the SD Card

1. 1. Once you have compiled your kernel and host code, you need to copy the `PROJECT.aocx` and `PROJECT` binaries to the SD Card. To do this, you will need a micro-SD card reader that plugs into USB. Please keep in mind the format of these SD Cards is ext4. Neither Windows nor Apple computers recognize this format without Third-Party software, which is unreliable, even though easily found online.
2. Once you have the micro-SD card mounted, you can copy files into the `home/root/PROJECT` folder, which you may need to create for each project.
3. Anytime you are copying files to the SD card, **MAKE SURE TO EJECT SECURELY BEFORE REMOVING**. Some SD cards have a high read/write time, and some file transfers may appear complete, but the buffer is still writing data to the card. Removing early may corrupt files on the card and require the card be reimaged. If using Linux commands in terminal, type `sync` and wait for this to finish. If using the GUI to copy, be patient and wait until the OS tells you it is safe to eject.
4. You can now re-insert the card into the FPGA and test the program execution by browsing to `PROJECT` in the file structure and running `./PROJECT PARAMETERS` (these parameters may change with each different project). Please contact a TA if any of these instructions are incorrect, confusing, or if any issues arise when attempting to follow them.

### 6.2.4 Hardware Setup

1. First, you need to change the switches on the bottom of the board. You will set them to **001010** from highest to lowest order as shown in Figure 11.

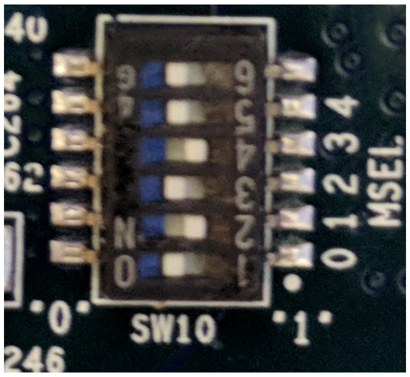


Figure 11: Correct switch setup for OpenCL.

2. From here, you must have a microSD card with Alteras Linux Console image built on it. Upon powering on the board, you should see the red LEDs begin a binary counting sequence. You will need to obtain your own microSD card and have it imaged by one of the TAs.
3. Verify the UART cable is connected to the host computer (mini-USB to USB Cable located near the SD Card slot).

#### 6.2.5 Software Setup

1. Open a terminal and type `minicom`. You should see an "Initializing Modem" message followed by a few messages from minicom. If this does not connect, or states that permission is denied or "ttyUSB0 is unavailable", then disconnect the and reconnect the UART cable on the FPGA board.
2. Power Cycle the FPGA. You should see boot messages appear from Terasic, dont press anything until prompted for login.
3. When prompted for "socfpga login:", type `root` and you should be logged in.
4. Next you **MUST** type `source ./init_opencl.sh` or you will not be able to run your OpenCL code.
5. You can now browse to the location of your executable and test files.

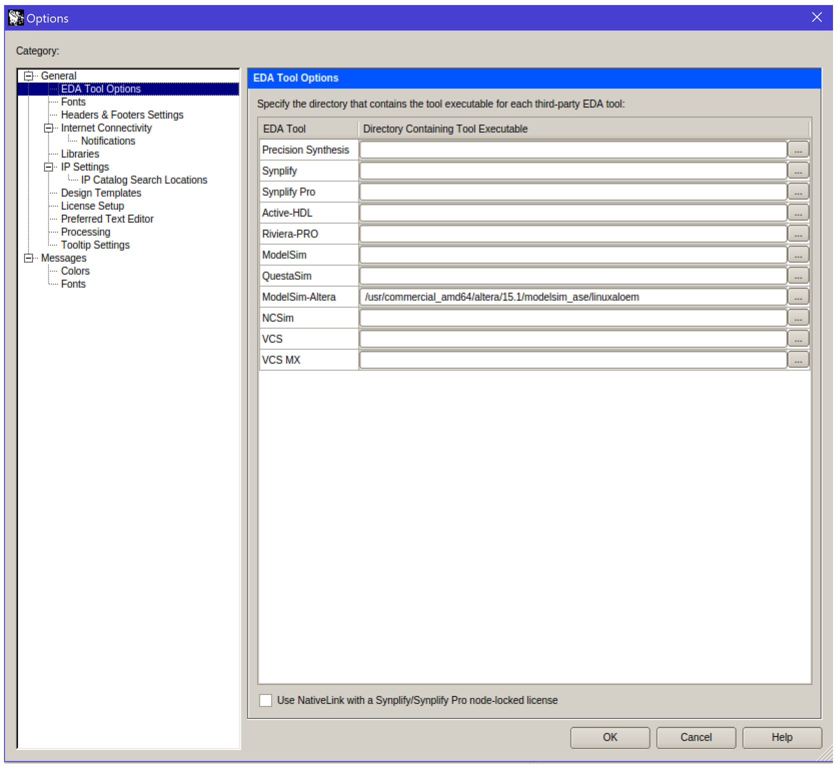
# 7 Lab 0: Tool install and Analysis

This lab is designed to walk you through installing the Quartus tools on your personal computer (optional) and teach you how to use them. This will also be taught in lab to reinforce the information in this manual.

## 7.1 Tool Install - Optional

As stated, it is not a requirement to install the tools on your personal computer. Many students do find it convenient, however, to be able to work on projects outside of the lab. Follow the instructions below to complete the installation:

1. Download the Quartus installer from Intel's page, here. You should select the Quartus Lite edition. The lab uses version 16.0, but you may use any newer version as long as it supports the **Cyclone V** device family.
   (a) When selecting which software to download/install, make sure to select **Cyclone V Device Support**, you may deselect all other devices.
   (b) Also make sure to select the **ModelSim-Intel FPGA Edition (includes Starter Edition)** option.
2. Once installed, you need to inform Quartus where ModelSim is installed. To do this, select **Tools-Options** and then select **EDA Tool Options** to get the window shown in figure 12. Select the 3 dots icon next to ModelSim-Altera and select the installation location for ModelSim-Altera Starter Edition.
   - For Windows, this is often in the root install location for Quartus, `C:\IntelFPGA-lite\16.0\`, or whichever version you installed. Inside of this directory, you will often find ModelSim in the `modelsim_ase` folder. You will then select the `win32aloem` folder as the location, as this folder contains the binary file for ModelSim-ASE.
   - For Linux, you will select a path very similar to that shown in figure 12. The only difference will be the install path you selected during the install.
3. For larger projects, the lab machines may compile quicker, as they are fully licensed and will use multiple cores. Personal installs with the free license do not have this option.



EDA Tool Options

Specify the directory that contains the tool executable for each third-party EDA tool:

| EDA Tool | Directory Containing Tool Executable |
|---|---|
| Precision Synthesis | |
| Synplify | |
| Synplify Pro | |
| Active-HDL | |
| Riviera-PRO | |
| ModelSim | |
| QuestaSim | |
| ModelSim-Altera | /usr/commercial_amd64/altera/15.1/modelsim_ase/linuxaloem |
| NCSim | |
| VCS | |
| VCS MX | |

☐ Use NativeLink with a Synplify/Synplify Pro node-locked license

Figure 12: EDA Tool Option Window.

## 7.2 Remote Desktop Connection

If you want to work from your own computer but do not want to install the tools, you may Remote Desktop into any of the lab machines. This will be useful during the OpenCL labs, as installing these tools to your computer is tedious and requires Linux (and TAs will not assist with this). There are 2 options to connect to:

- `ullabXX.ces.clemson.edu` and replace XX with `01` to `14`.
- `riggs309labXX.cecas.clemson.edu` and replace XX with `01` to `06`.
- `apolloXX.ces.clemson.edu` and replace XX with `01` to `14`.

When using a Remote Desktop Connection, you must be on Campus or connected via a VPN. Keep in mind that a VPN connection will result in laggy graphical performance if you run the connection at a high resolution. The computer will still process just as quickly, it just may take more time for updates to appear on the screen.

## 7.3 Tutorial

First, please refer to the information below.

- There are many tutorials available for the Intel Quartus CAD tools. You can find those here. Complete the below tutorials:
  - Getting Started with the DE-Series Boards

  - Quartus Introduction (Standard Edition)
  - Using ModelSim to Simulate Logic Circuits
  - Debugging of Hardware Designs

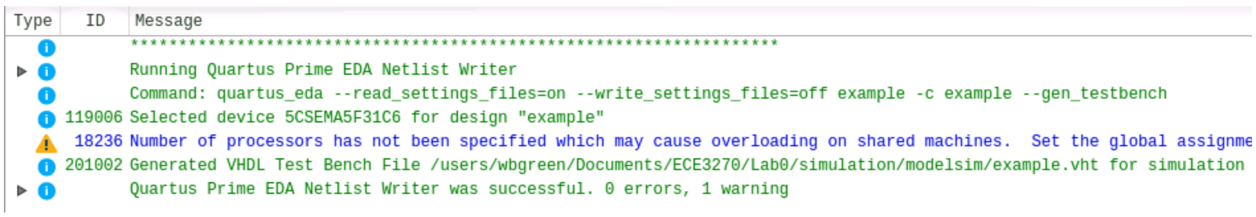
  And it is recommended you complete or at least read the below tutorials:

  - Using the Library of Parameterized Modules (LPM)
  - Using TimeQuest Timing Analyzer
  - Signal Tap II Logic Analyzer

  Some of these materials will be covered in lab, but not all.

This next section will use the code in Section 9. We will walk through the process of using this code to complete a project. Note all information in this manual is referring to Quartus 16.0, and specifically the installations on the ULLAB machines.

1. First, we need to create a project using the **New Project Wizard**. You can ignore the **Introduction** screen and just click **next** if it pops up for you.
2. Next, we need to select a working directory and name of our project. It is suggested you create a folder for this project (and future projects) and select this. For the name of the project and the **top-level design entity**, use **example**. Select **next**.
3. Choose **Empty Project** and click **next**.
4. On future projects, you may add files from previous projects if applicable. You will do that on this page. After you are finished adding files, click **Next**.
5. The next stage is selecting the appropriate device. Make sure **Cyclone V** is the selected device family, then search for **5CSEMA5F31C6** in the device list. Select this device and click **Next**.
6. On the EDA Tools settings page, you must make sure the **Simulation** line has **ModelSim-Altera** and **VHDL** selected. This will make sure the appropriate simulator is linked and it can understand the language we are using. Click **Next**, review your selections on the next page, and then click **Finish**.
7. You should see a blank project. We need to create a new VHDL file, copy and past the code from Section 9, and save the file with the name **example.vhd**. There is a possibility copy and paste will not save whitespace, so you may need to type the code out (which is good practice anyway!). Verify the program compiles by pressing the **Start Analysis & Synthesis** button.
   - Note the comments in this code. They tell you what the TAs expect, but they are not representative of what we expect.
8. Once **Analysis & Synthesis** is completed, you can now generate a testbench. To do this, you will click **Processing->Start->Start Test Bench Template Writer**. This will generate a testbench file and save it. By looking in the **Messages** window, you will see the saved file, as shown in figure 13.



```
Type    ID     Message
               ********************************************************************
               Running Quartus Prime EDA Netlist Writer
               Command: quartus_eda --read_settings_files=on --write_settings_files=off example -c example --gen_testbench
        119006 Selected device 5CSEMA5F31C6 for design "example"
         18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignme
        201002 Generated VHDL Test Bench File /users/wbgreen/Documents/ECE3270/Lab0/simulation/modelsim/example.vht for simulation
               Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
```

Figure 13: Messages window showing location of generated testbench file.

9. Add this file to your project by clicking **Assignments->Settings** and selecting **Files**. Simply browse to the location shown in the messages window and add the file.

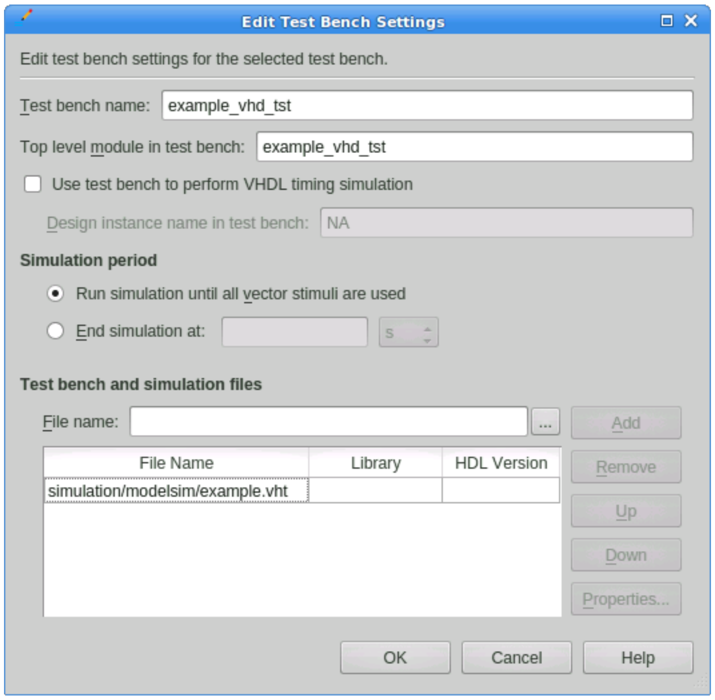
10. Now write a few lines of code to test the example project. This should be written inside the process called **always**. If you had a reset signal or setup required for the code to work correctly, you would write that in the process labeled **init**.

    - For any testbench you write for projects, you do not have to be exhaustive. However, the TAs will expect you to test specific instances for most designs. For example: boundary cases (all 0's or 1's) and any other interesting value for a given component (overflow cases for addition).

11. Next is adding the testbench to your project so that the simulator will compile it and test your device appropriately. Click **Assignments->Settings** and select **Simulation**. Select the **Compile test bench** option and click the **Test Benches** button. Now click **New** and you will be presented with a window asking for the specific testbench information. Fill it out as shown in figure 14, making sure to add your testbench file. Press **OK** on all of the windows.

    - When adding testbenches to a project, check the entity name of your testbench file. You **MUST** use that name as the testbench top level entity name.

    - When testing multiple design files in a testbench, you will notice there is a dropdown menu on the **Simulation** page of the **Assignment Settings** window. Make sure you have the correct design selected for the file you are planning on testing.



Edit test bench settings for the selected test bench.

Test bench name: example_vhd_tst

Top level module in test bench: example_vhd_tst

☐ Use test bench to perform VHDL timing simulation

Design instance name in test bench: NA

Simulation period

● Run simulation until all vector stimuli are used

○ End simulation at: s

Test bench and simulation files

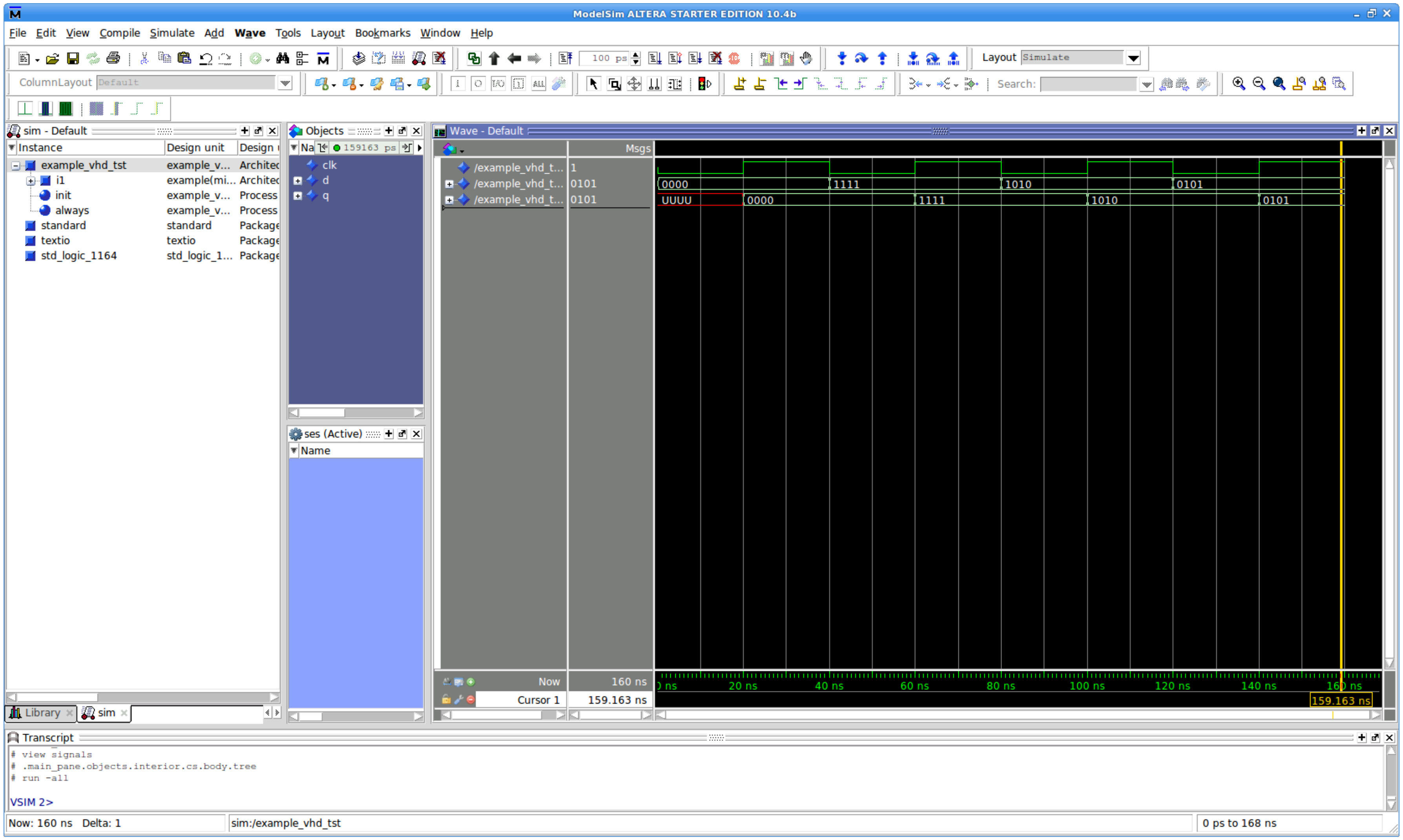
File name:

| File Name | Library | HDL Version |
|---|---|---|
| simulation/modelsim/example.vht | | |

Figure 14: Testbench Settings window.

12. Now you are ready to simulate. Simply click **Tools->Run Simulation Tool->RTL Simulation** to launch ModelSim. If you had no errors in your testbench, ModelSim will automatically run your file until it reaches the end of the always process. Click the waveform window, and then select the **Zoom Fit** option and it will zoom out the window so you can see all of your changes more clearly. Your final output should look similar to figure 15.

- If your waveform does not show up, but ModelSim does load, it likely means you had a compile error in your testbench. Check the messages window at the bottom of ModelSim and look for errors. You can double click these to highlight the error quickly.
- After fixing errors, you can either relaunch ModelSim, or recompile directly in the already launched ModelSim window. To do this, you will expand **Work** in the **Library** window of ModelSim. Right click your testbench file and click **Recompile**. Once compiled, double click the file and your signals should pop up in the **Objects Window**. Highlight all relevant objects, right click, and select **Add Wave**. At the top of the waveform window that pops up, there will be a time (default is 100ps). Change this to a value that will complete your testbench, then click **Run** which is just to the right of the time you modified. Refer to 12 to zoom your window.



| Instance | Design unit | Design |
|---|---|---|
| example_vhd_tst | example_v... | Architec |
| i1 | example(mi... | Architec |
| init | example_v... | Process |
| always | example_v... | Process |
| standard | standard | Package |
| textio | textio | Package |
| std_logic_1164 | std_logic_1... | Package |

| Name |
|---|
| clk |
| d |
| q |

| Signal | Msgs |
|---|---|
| /example_vhd_t... | 1 |
| /example_vhd_t... | 0101 |
| /example_vhd_t... | 0101 |

```
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all

VSIM 2>
```

Now: 160 ns   Delta: 1   sim:/example_vhd_tst   0 ps to 168 ns

Figure 15: Complete Simulation Window.

13. Once you have done all of this and verified functionality in simulation, you are ready to implement your design on the board. It is suggested you create a new VHDL file for this. Once done, copy the code from Section 10 into your new file. If you read carefully over this, it simply uses the on board components (SW and LEDR) and maps them to your component as the device under test. Save this file using the **same name** as the top level entity.

14. We also need to import our pin assignments so we can actually program the board and use the SW, KEY, LEDR, or HEX displays. Download the **DE1.qsf** file from Canvas. In Quartus, click **Assignments->Import Assignments** and select the **DE1.qsf** file.

15. Before compiling, we have to make sure the correct entity is set as the top level. To do this, change the **Project Navigator** view from **Hierarchy** to **Files**. Right click the new file, which should be titled **example_board.vhd**, and select **Set as Top-Level Entity**. Switch back to **Hierarchy** view and verify **example_board** is now your listed as the entity.

    - When working with larger projects, you may end up changing this top-level entity multiple times. It is **HIGHLY SUGGESTED** that you maintain the format of naming your .vhdl files using the same name as the entity in them. This simplifies this entire process.
    - If you run into problems attempting to compile, verify you have the correct top-level entity selected in the hierarchy view.
    - Also verify you have imported the pin assignments!

16. Click **Start Compilation** to complete the entire compilation chain.

17. Refer to Section 6.1 to program your device and verify functionality.

## 8 Random Tips

In this section you will find random tips to help you out when working in the lab in this class.

1. You can open the RTL Viewer (shows you the generated hardware schematic) by clicking **Tools->Netlist Viewers->RTL Viewer**

2. You can open the State Machine viewer (shows you the state machine diagram) by clicking **Tools->Netlist Viewers->State Machine Viewer**. If the state machine does not show up here, you did not describe your VHDL correctly.

## 9 Appendix A: Tutorial Example Code

```
--This is example.vhd, used for Lab0.
--Read over this code and determine the purpose.
--Further, what hardware does this represent?
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity example is
    port( d:    IN    std_logic_vector(3 downto 0);
          clk:  IN std_logic;
          q:    OUT std_logic_vector(3 downto 0));
end example;

--Begin the architecture, why is it called mixed?
architecture  mixed of example is
    signal internal_d: std_logic_vector(3 downto 0);

    begin
    --You should expect a comment explaining the process in your own code.
    --However, this is omitted here so you can figure it out!
    process(clk)
        begin
        if(rising_edge(clk)) then
            internal_d <= d;
        else
            internal_d <= internal_d;
        end if;
    end process;

    q <= internal_d;
end mixed;
```

## 10 Appendix B: Tutorial Board Code

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY example_board IS
port(SW:  IN STD_LOGIC_VECTOR(9 downto 0);
  LEDR: OUT STD_LOGIC_VECTOR(9 downto 0));
END example_board;

ARCHITECTURE struct OF example_board IS

COMPONENT example
PORT (
     clk : IN STD_LOGIC;
     d : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
     q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
);
END COMPONENT;

BEGIN
    --Create the device and map it to the appropriate pins
dut : example
    PORT MAP (
         clk => sw(4),
         d => sw(3 downto 0),
         q => ledr(3 downto 0)
    );

end struct;
```