Be Boulder.
University of Colorado Boulder

CAPSTONE PROJECT GUIDE

MODULE I

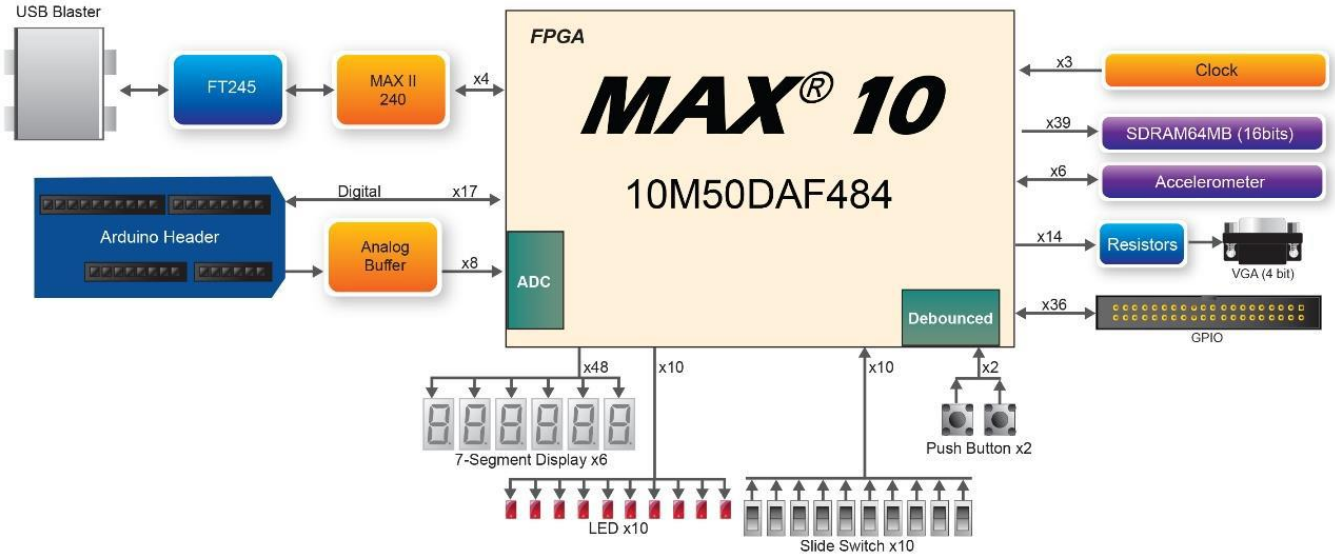
FPGA CAPSTONE PROJECT MODULE I: ALTERA MAX10 HARDWARE SETUP

TABLE OF CONTENTS

INTRODUCTION

The DE10-Lite is a FPGA evaluation kit that is designed to get you started with using an FPGA. The DE10-Lite adopts Altera's non-volatile MAX® 10 FPGA built on 55-nm flash process. MAX 10 FPGAs enhance non-volatile integration by delivering advanced processing capabilities in a low-cost, instant-on, small form factor programmable logic device. The devices also include full-featured FPGA capabilities such as digital signal processing, analog functionality, Nios II embedded processor support, and memory controllers.

The DE10-Lite includes a variety of peripherals connected to the FPGA device, such as 8MB SDRAM, accelerometer, digital-to-analog converter (DAC), temperature sensor, thermal resistor, photo resistor, LEDs, pushbuttons and several different options for expansion connectivity.



PROJECT LEARNING OBJECTIVES

For this project, the objective is for students to:

- Become familiar with the FGPA development flow, particularly in the case of a SoC with software development flow included.
- Appreciate the capability of the MAX10 to create whole systems on a chip.
- Learn how to build systems using the Qsys (Platform Designer) system design tool.
- Learn to create hardware using schematic capture input.
- Learn how to integrate software with hardware in the same device.
- Understand the rationale for each phase of the hardware development flow, including timing constraints, simulation, and programming.
- Design and build a several hardware examples using the MAX10.
- Consider hardware and software tradeoff possibilities available in the SoC architecture.

Do not be afraid to ask questions in the discussion forums as you work on this project. It involves many processes and actions that may seem complicated and confusing at first, but will become clearer as you work through the modules.

The modules will get progressively more difficult and take more time. This is Module 1.

MODULE GOAL

The goal of this module is to setup the MAX10 board, and create a series of simple working designs. In this module you will

- Setup and test the MAX10 board using the FPGA design tool Quartus Prime and the System Builder.
- Design and test a Binary Coded Decimal Adder
- Record all your observations in a lab notebook pdf
- Submit your project files and lab notebook for grading

Completing this module will help prepare you for the work to be done in the modules that follow.

I. GENERAL PROCEDURE

Caution:

Do not continue until you have read the following:

The names that this document directs you to choose for files, components, and other objects in this exercise **must be spelled *exactly* as directed.**

SETUP AND TEST MAX10 FPGA BOARD

1. If you have not already done this, download and install Quartus Prime FPGA development software from Intel Altera. Follow the directions in the installation video in Course 1 of the specialization if you need help. Install version 16.1.
2. Create a Lab Notebook document that can eventually be saved as a pdf file. Typically this will start as a Microsoft Word or Google Docs file.
3. Download the pdf document De10-Lite_User_Manual.pdf from the Course Resources, and read chapters 1, 3 and 4. Recognize that Quartus II now means Quartus Prime.
4. Download and unzip the DE10-Lite_v.2.0.0_SystemCD.zip.
5. Remove your DE10-lite evaluation board and plug the USB cable into a computer. Record your observations in your lab notebook - Describe how the device behaves – what do you see on the LED display and what is the light pattern?
6. Create a System using the System Builder using the instructions given in the detailed design section that follows, and record your observations in your lab notebook.

MODULE I: LIGHT THE LIGHTS, HIT THE HEIGHTS

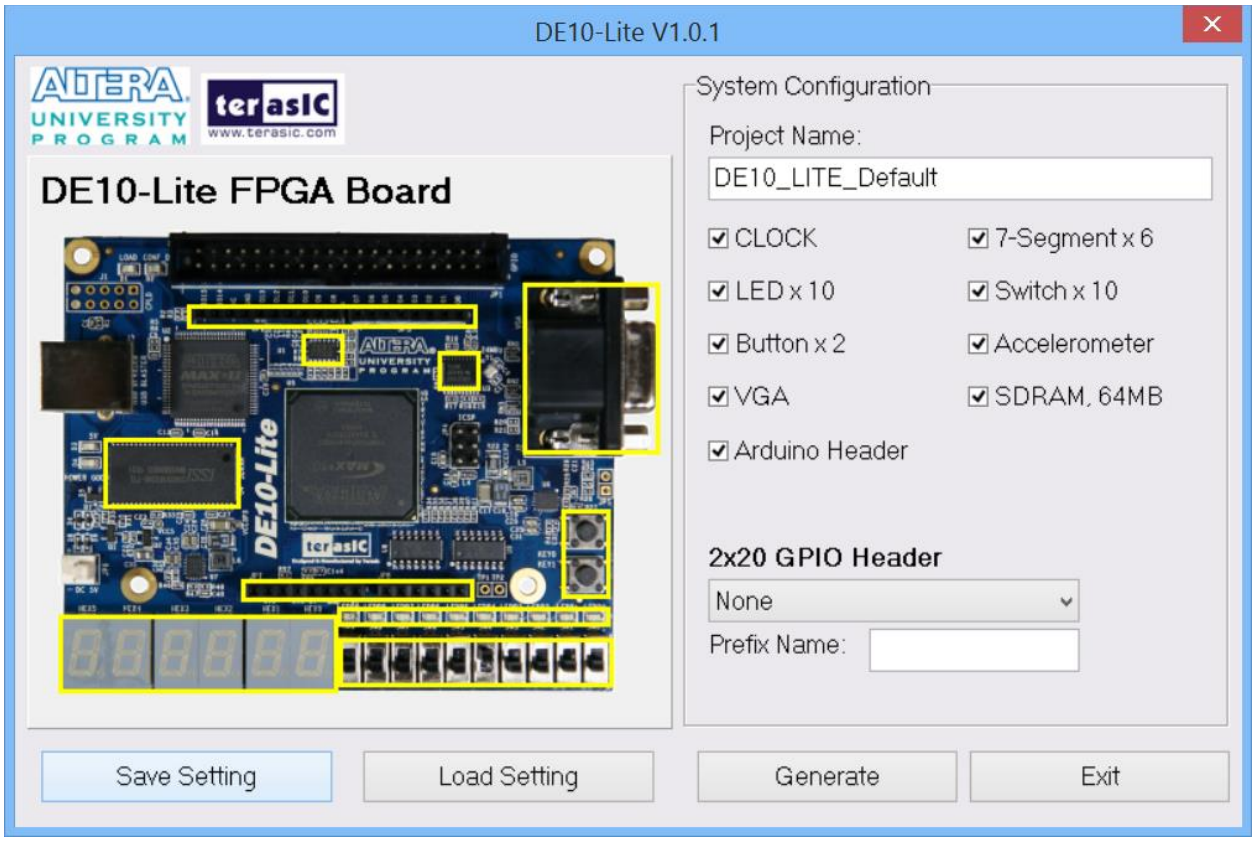
1. Follow the instructions in the detailed design section that follows for Module 1. Be sure to record your observations as you go.
   a. If your results look different than the project guide, do not be alarmed, as there may be some differences between the tools used in the guide and your particular installation of the tools.
   b. Do not be surprised when the initial and even subsequent compiles of the FPGA have errors. This will point you to the additional work you need to do.
2. Record the Fmax for this lab in your lab notebook. If an Fmax is not listed, use the inverted highest clock frequency for this number.
3. Estimate the % utilization of the FPGA logic for this lab at completion.
4. Record your observations of the board behavior once the FPGA is programmed. Does it behave as you expected?
5. Submit a zipped up project file for auto-grading as well as your lab notebook.

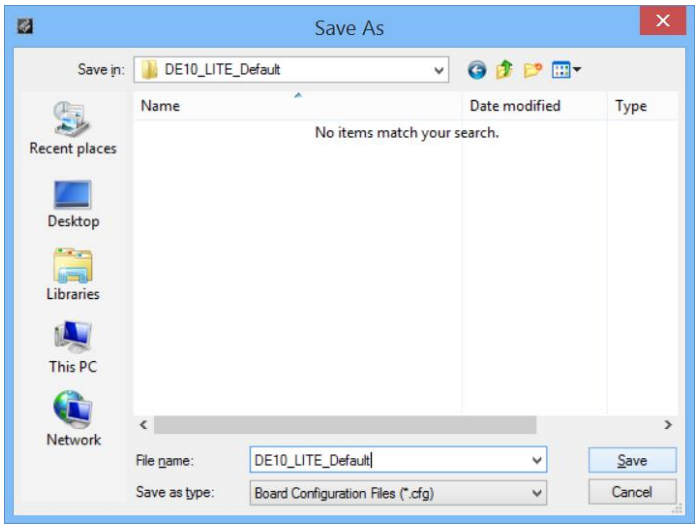
II. DETAILED DESIGN

SETUP AND TEST THE DE10-LITE DEVELOPMENT KIT

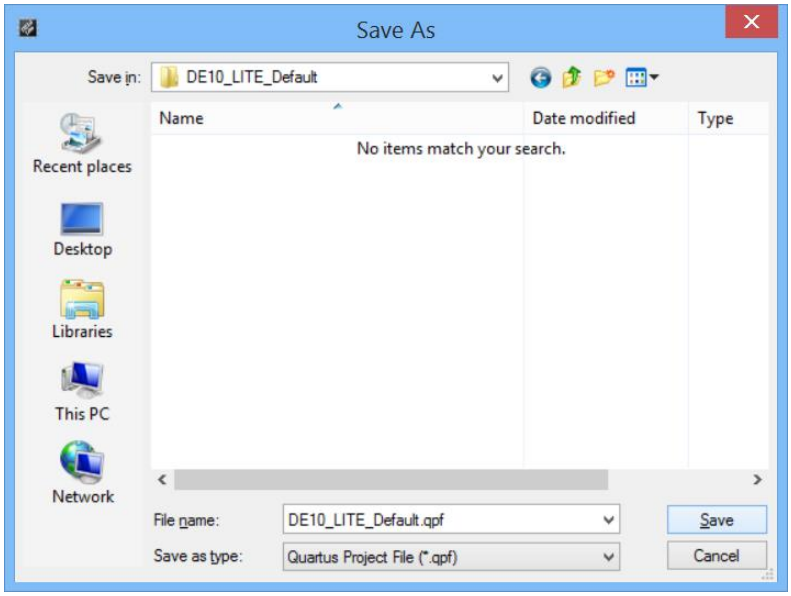
I. Create a System using System Builder

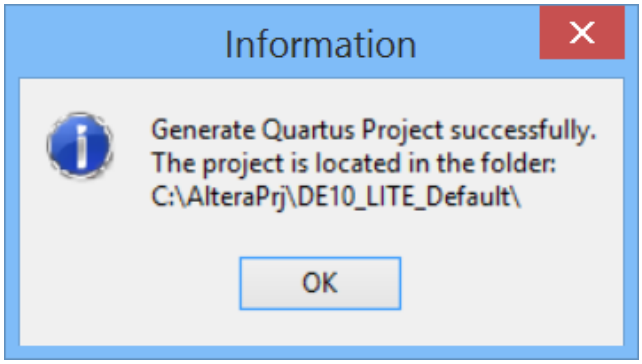
a) In your main hard drive, typically C:\, create an AlteraPrj folder. Within this folder, create DE10_LITE_Default folder, and a DE10_LITE_Small folder.
b) From the SystemCD.zip directory, extract, find (in the Tools directory) and run the System Builder utility.
c) Change the Project Name to DE10_LITE_Default. Confirm all the checkboxes are checked, including Clock, LED, Button, VGA, Arduino, 7-segment, Switch, Accelerometer, and SDRAM. Click Save Settings, and navigate to the DE10_LITE_Default folder you just created and save the configuration file there, name it DE10_LITE_Default.cfg.



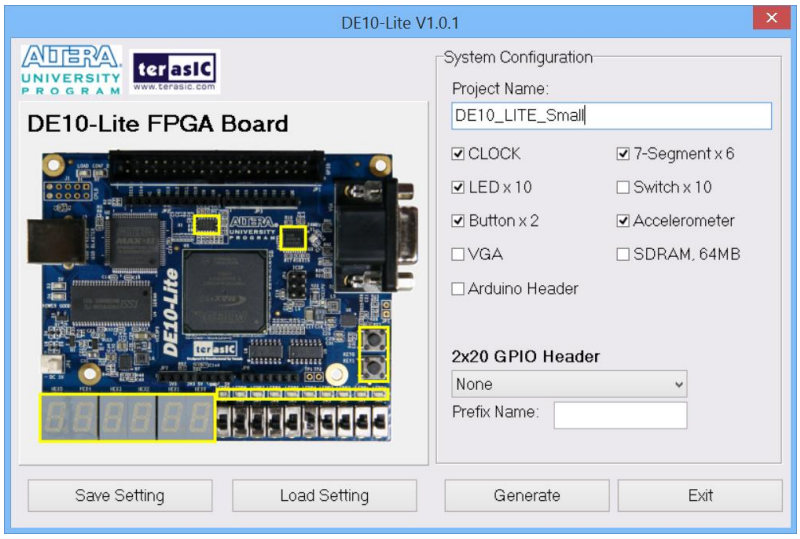


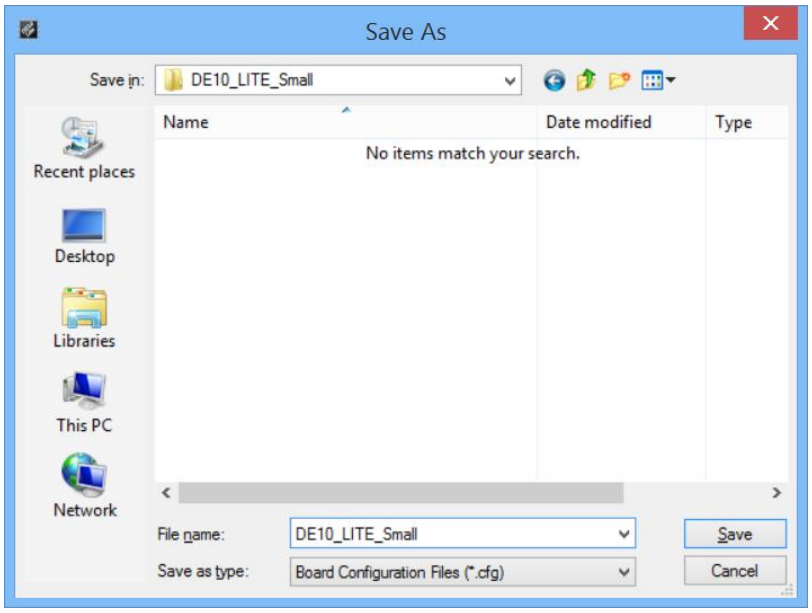
d) Press Generate, and save the project in the DE10_LITE_Default folder you just created. Be sure to navigate to the correct folder.



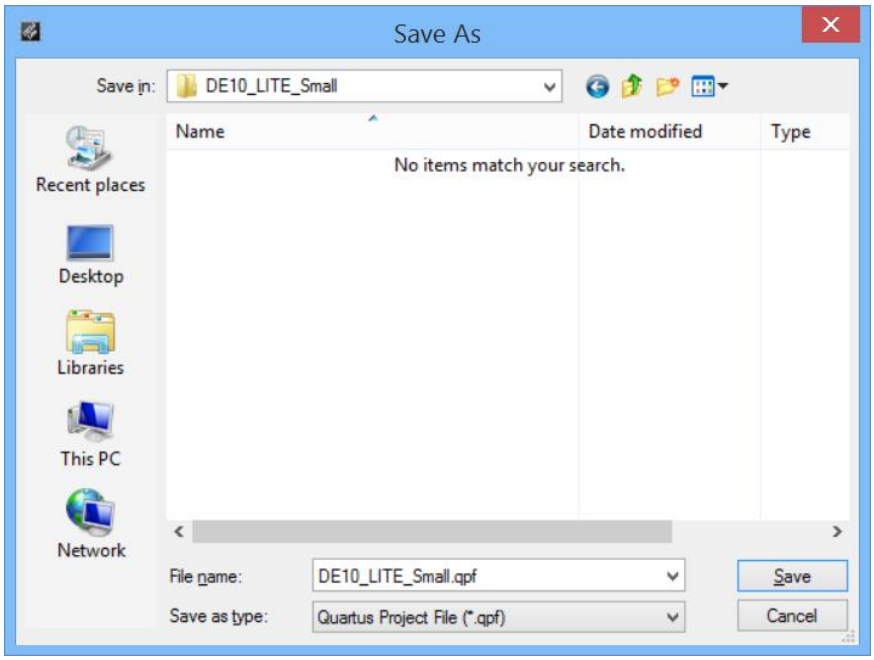


e) Uncheck the boxes VGA, Arduino, SDRAM, and Switch. Change the project name to DE10_LITE_Small. Click Save Settings, and navigate to the DE10_LITE_Small folder you just created and save the configuration file there. Be sure to use the DE10_LITE_Small file name.





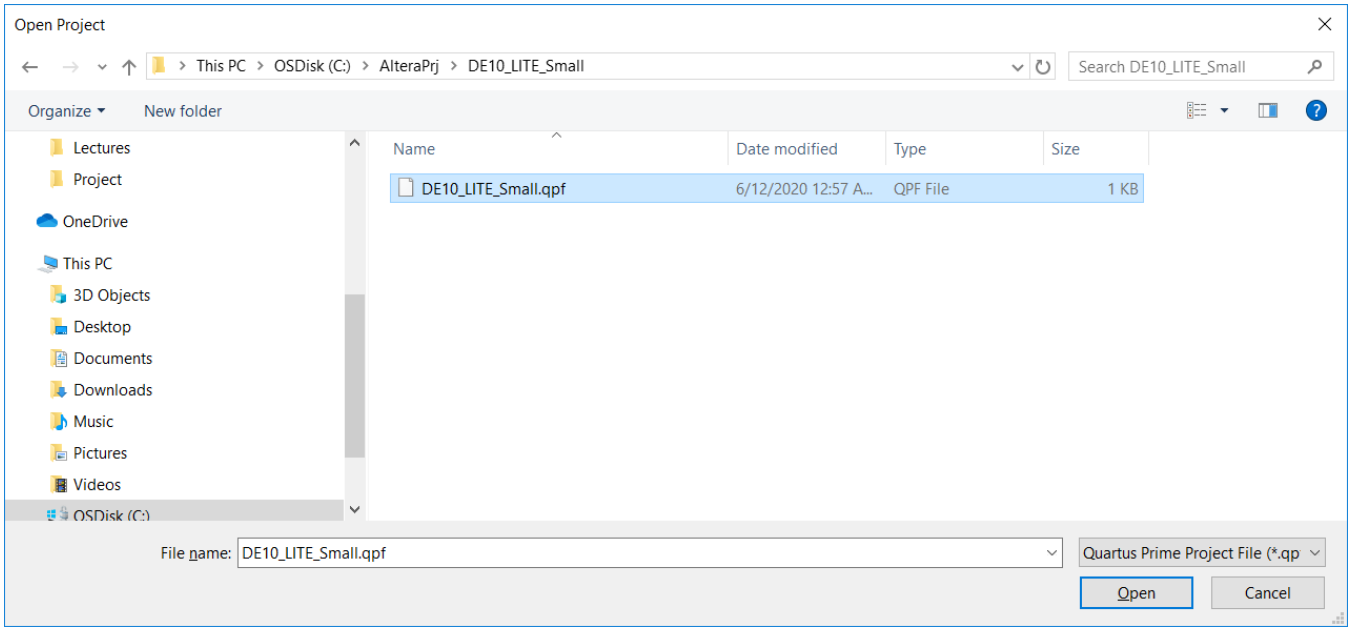
f) Press Generate, and save the small project in the DE10_LITE_Small folder you just created. Be sure to navigate to this folder.



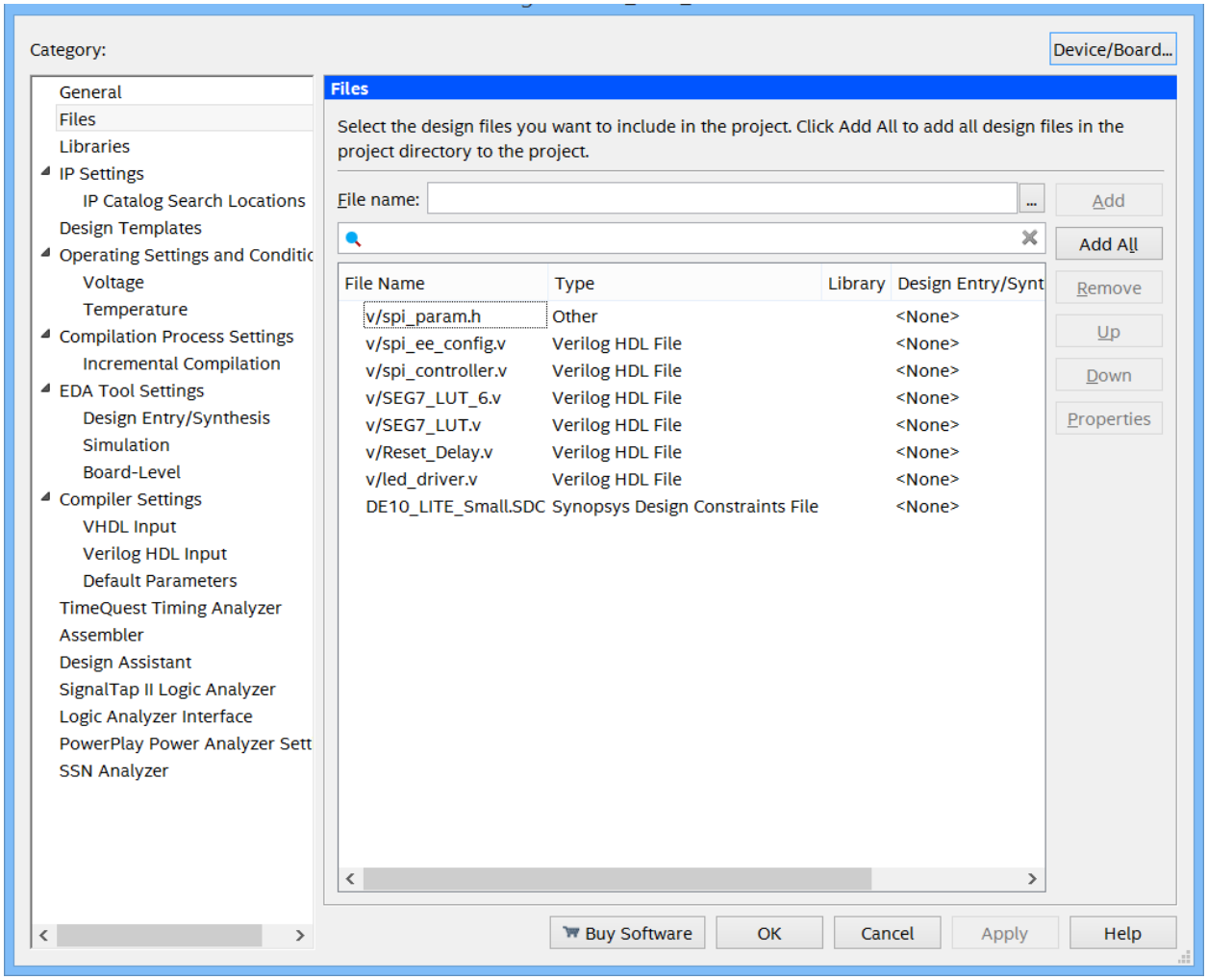
g) From Course Resources, get the DE10_LITE_Default.zip file and unzip it into the AlteraPrj/DE10_LITE_Default directory. Also from Course Resources, copy DE10_LITE_Default.v to this directory, say OK to overwrite.

h) From Course Resources, get the DE10_LITE_Small.zip file and unzip it into the AlteraPrj/DE10_LITE_Small directory. Also from Course Resources, copy DE10_LITE_Small.v to this directory, say OK to overwrite.

i) Launch Quartus Prime, and open the small project you just created.



Run full compilation on this project after using Projects-> Add/Remove files to get a file list like this:



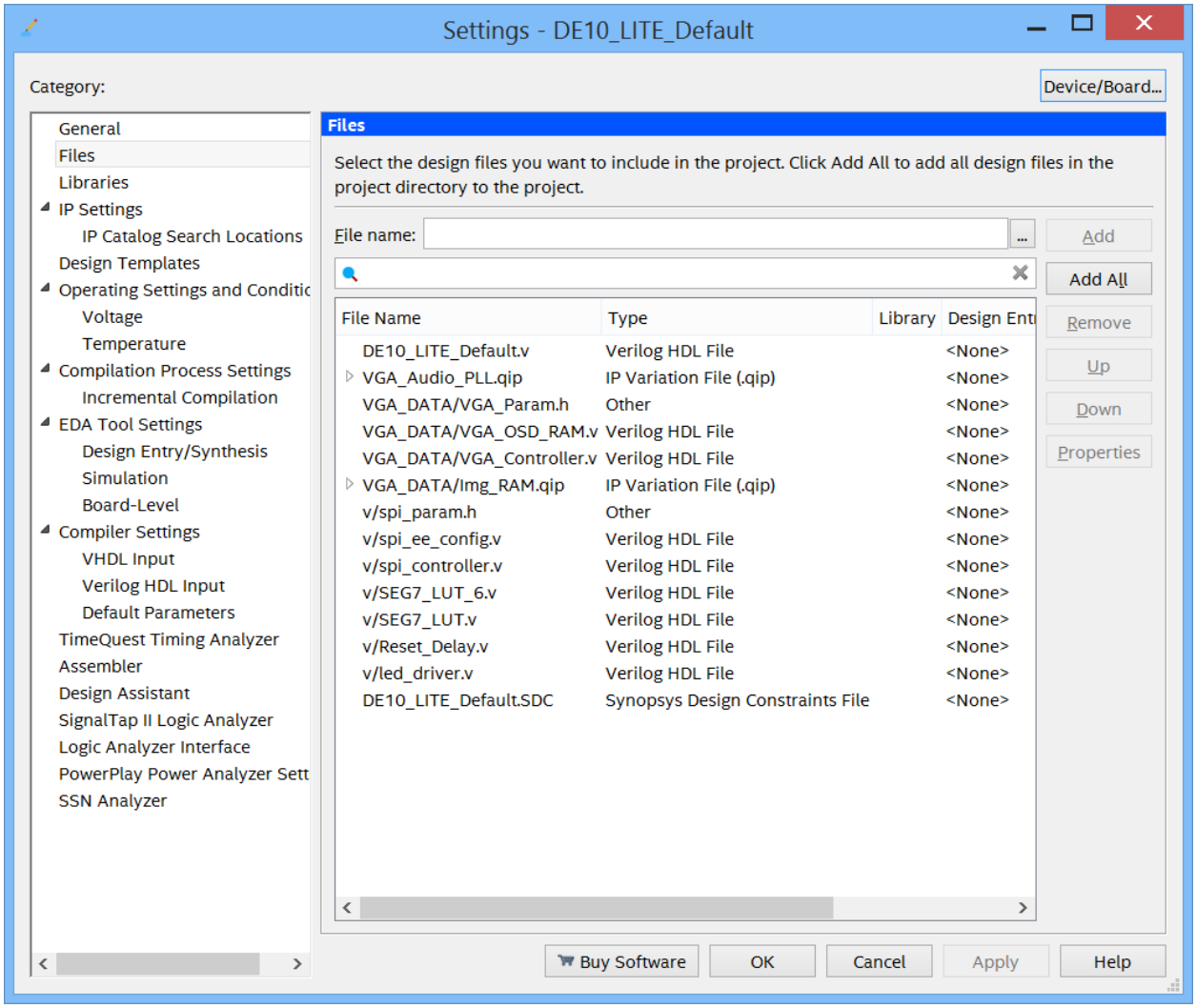
Ignore any unconstrained paths for now.

j) Using the instructions from the beginning of Chapter 3 of the user's manual, program the FPGA using the DE10_LITE_Small.sof programming file. If you see "No Hardware", you may have to update the USB Blaster Driver.
k) From the Compiler Report in Quartus, estimate the Fmax and % utilization of FPGA logic.
l) Drill down into the blocks in the RTL Viewer. Estimate the number of Flip-flops used.
m) Record your observations of the board behavior once the FPGA is programmed. Does it behave as you might expect?

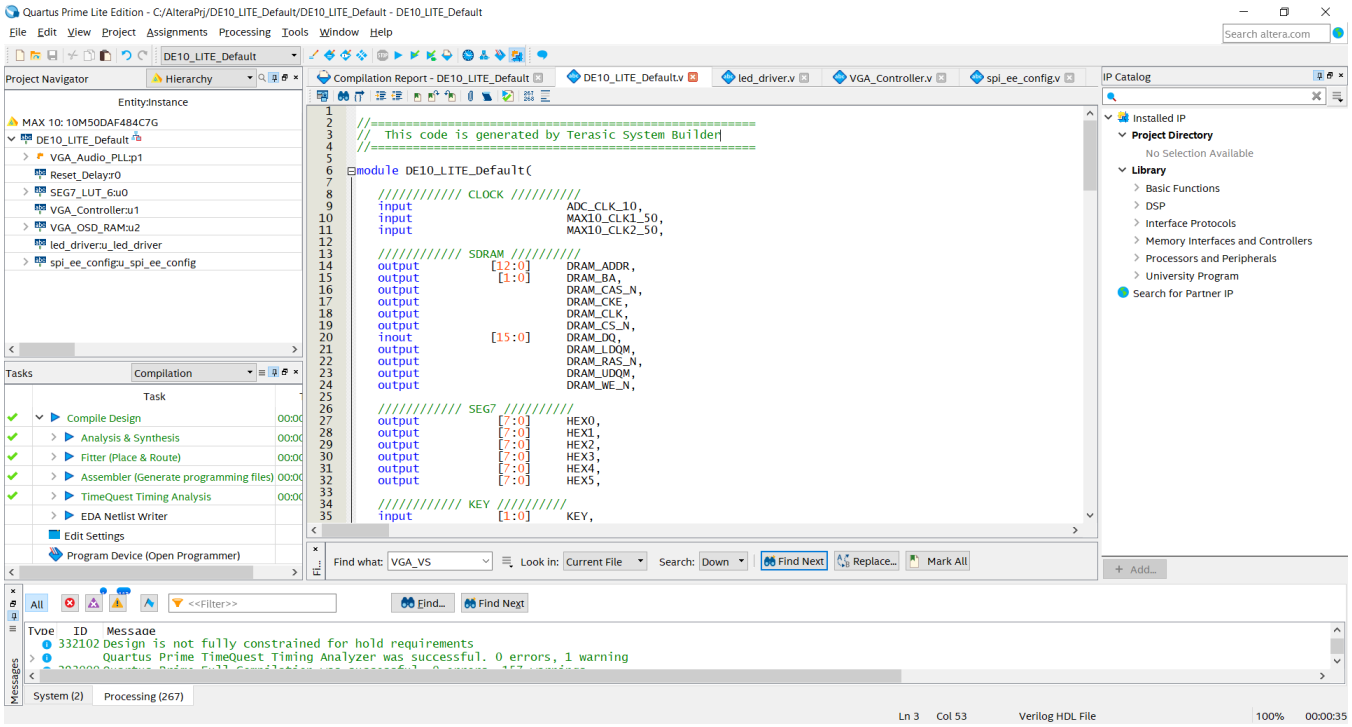
CONGRATULATIONS!!

You have just created your first working FPGA!

n) Repeat i) through l) for the default project you created in c) and d), the DE10_LITE_Default. Again use Projects-> Add/Remove files, but this time use this list:



After compilation, you should see something like this:



TIPS FOR SUCCESS

If you see a request to upgrade IP in the Project Navigator window, ignore it for now.
If you see error about an undefined entity, then you may need to add another file to the project, the error message should give you some idea which one.
If you see errors in the full compilation relative to memory initialization, consider using Assignments->Device->Device and Pin Options-> Configuration-> Configuration mode to change the configuration mode to include memory initialization.

CONGRATULATIONS!!

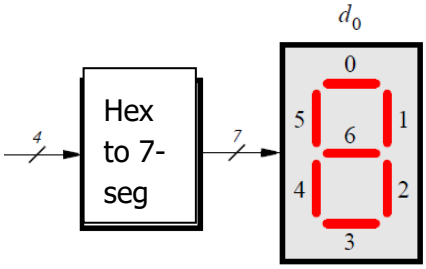
You have completed the setup and test of the DE10-Lite!

MODULE1: HIT THE HEIGHTS, LITE THE LIGHTS

NUMBERS AND DISPLAYS

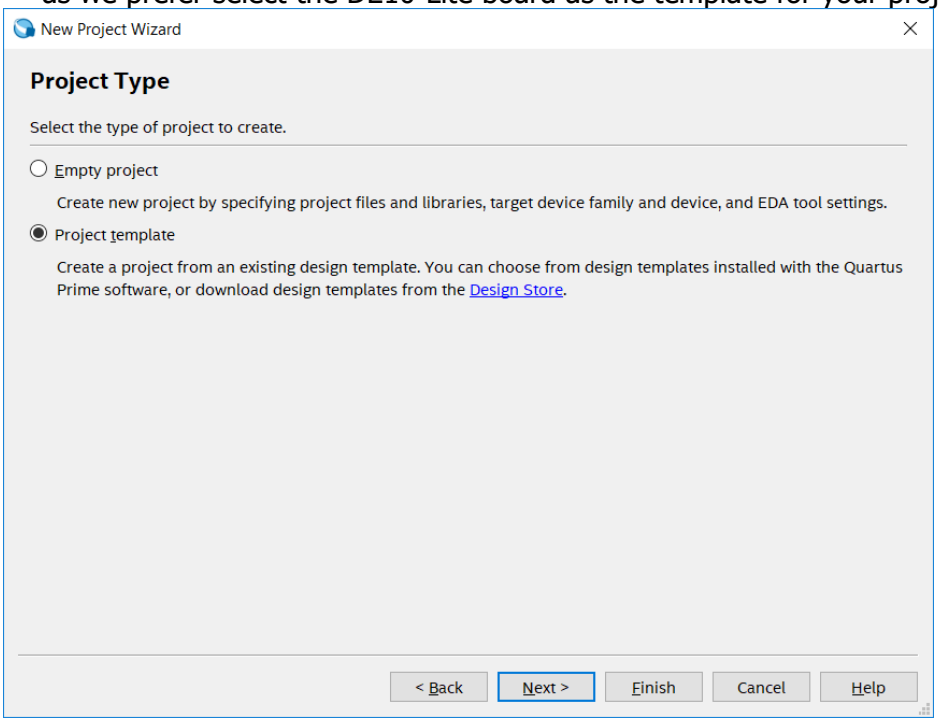
This is an exercise in designing combinational circuits that can perform binary-to-decimal number conversions and binary-coded-decimal (BCD) addition [1].

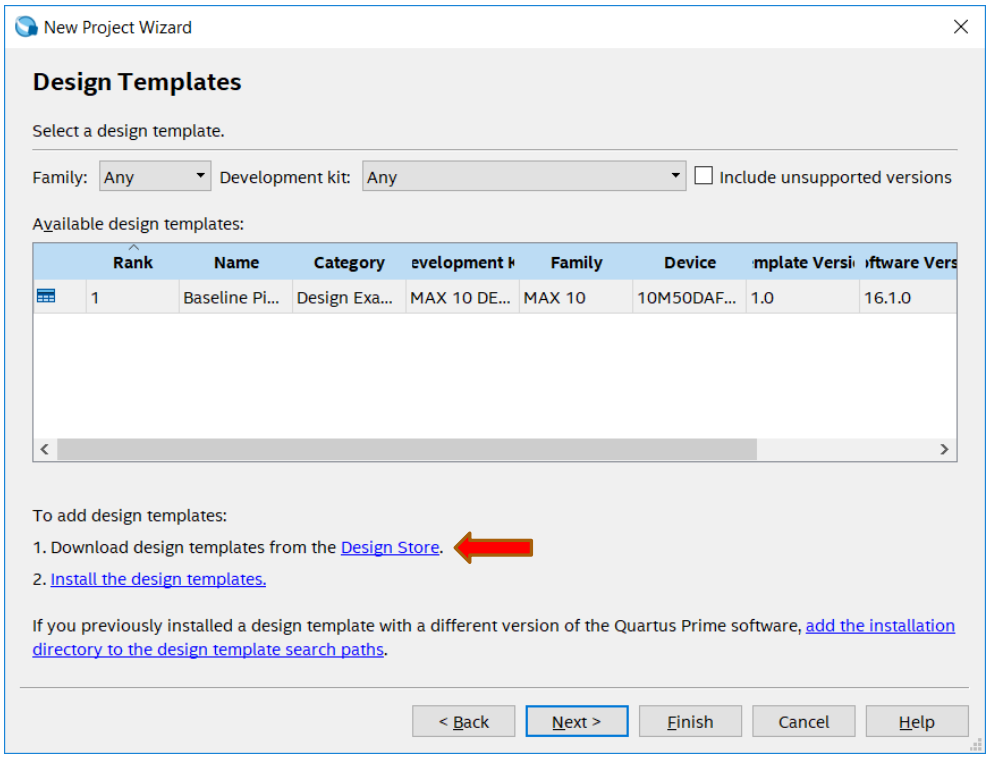
PART I – DISPLAY SWITCH VALUES



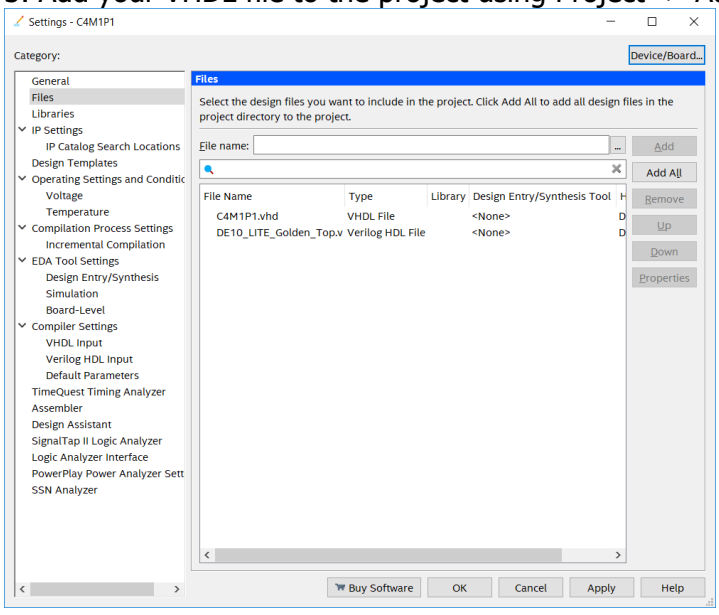
We wish to display on the 7-segment displays HEX1 and HEX0 the values set by the switches SW_{7-0}. Let the values denoted by SW_{7-4} and SW_{3-0} be displayed on HEX1 and HEX0, respectively. Your circuit should be able to display the digits from 0 to 9, and should treat the valuations 1010 to 1111 as don't-cares.

1. Create a new project named C4M1P1 which will be used to implement the desired circuit on your Intel FPGA DE-series board. You could copy your existing DE10_LITE_Default project using Project -> Copy Project and change the project name to get the correct board setup, or as we prefer select the DE10-Lite board as the template for your project as shown here:





2. Write a VHDL file that provides the necessary functionality. Name it C4M1P1.vhd. The intent of this exercise is to manually derive the logic functions needed for the 7-segment displays. Therefore, you should use only simple VHDL assignment statements in your code and specify each logic function as a Boolean expression. Include this file in your project and assign the pins on the FPGA to connect to the switches and 7-segment displays. Make sure to include or verify the necessary pin assignments using the pin planner.
3. Add your VHDL file to the project using Project -> Add/Remove Files



You will need to instantiate your VHDL code into the top-level Verilog template. To instantiate a VHDL module inside a Verilog design, make sure the two files are in the same directory and that they have been added to the project for compilation. Next, simply instantiate the lower level VHDL design by name in the Verilog file. In this case, insert a line of code in the top level Verilog file something like:

```
C4M1P1 u1 (.SW(SW), .HEX0(HEX0), .HEX1(HEX1));
```

This should wire together your VHDL design with the top-level. Save the modified top-level Verilog file, which should be either DE10_LITE_Golden_Top.v or DE10_LITE_Default.v. Be sure to set this file as the top-level entity.

4. Compile the project and download the compiled circuit into the FPGA chip using the .sof file found in the output files folder of the project.
5. Test the functionality of your design by toggling the switches and observing the displays. From the Compiler Report in Quartus, estimate the Fmax and % utilization of FPGA logic. Drill down into the blocks in the RTL Viewer. Estimate the number of Flip-flops used. Record your observations of the board behavior once the FPGA is programmed. Does it behave as you might expect?

CONGRATULATIONS!!

You have just completed your first FPGA Design!

PART II – DISPLAY BINARY CODED DECIMAL (BCD)

You are to design a circuit that converts a four-bit binary number V = v3v2v1v0 into its two-digit decimal equivalent D = d1d0. Table 1 shows the required output values. A partial design of this circuit is given in Figure 1. It includes a comparator that checks when the value of V is greater than 9, and uses the output of this comparator in the control of the 7-segment displays. You are to complete the design of this circuit.

v3v2v1v0	d1	d0
0000	0	0
0001	0	1
0010	0	2
…	…	…
1001	0	9
1010	1	0
1011	1	1
1100	1	2
1101	1	3
1110	1	4
1111	1	5

Table 1: Binary-to-decimal conversion values.

The output z for the comparator circuit can be specified using a single Boolean expression, with the four inputs $V_{3\text{-}0}$. Design this Boolean expression by making a truth table that shows the valuations of the inputs $V_{3\text{-}0}$ for which z has to be 1.

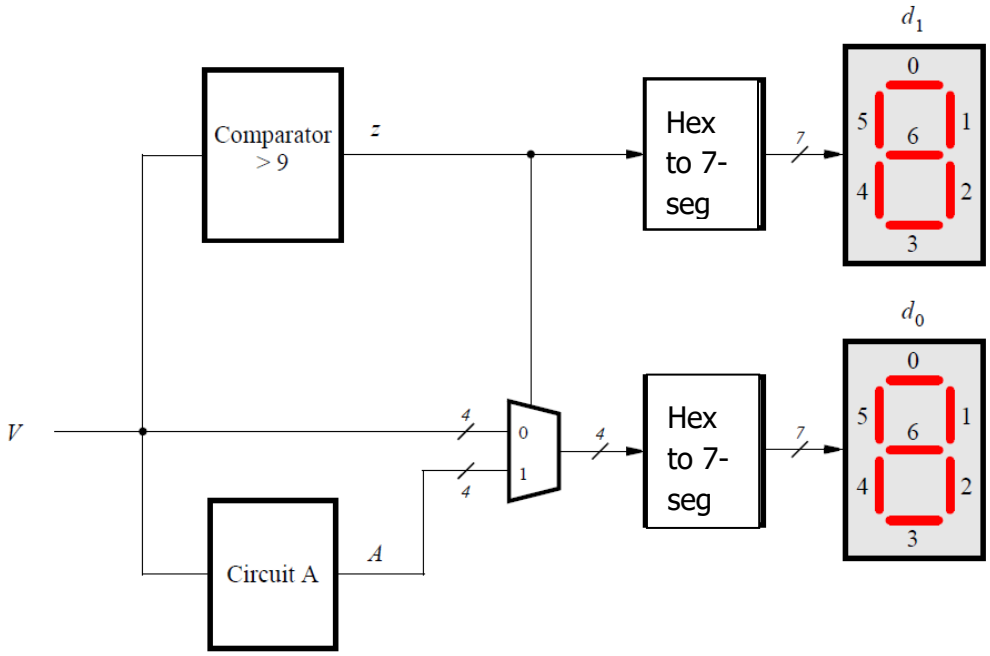


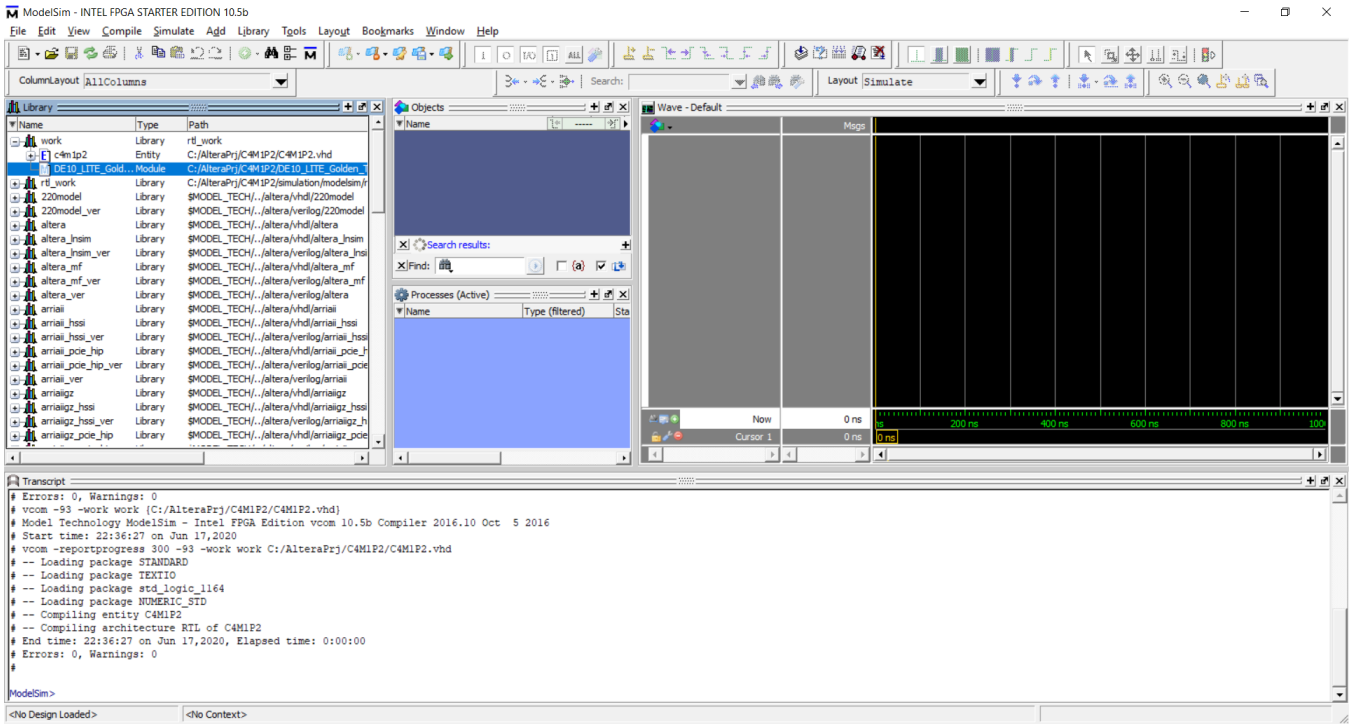
Figure 1: Partial design of the binary-to-decimal conversion circuit.

Notice that the circuit in Figure 1 includes a 4-bit wide 2-to-1 multiplexer. The purpose of this multiplexer is to drive digit d0 with the value of V when z = 0, and the value of A when z = 1. To design circuit A consider the following. For the input values V <= 9, the circuit A does not matter, because the multiplexer in Figure 1 just selects V in these cases. But for the input values V > 9, the multiplexer will select A. Thus, A has to provide output values that properly implement Table 1 when V > 9. You need to design circuit A so that the input V = 1010 gives an output A = 0000, the input V = 1011 gives the output A = 0001, …, and the input V = 1111 gives the output A = 0101. Design circuit A by making a truth table with the inputs V3-0 and the outputs A3-0.

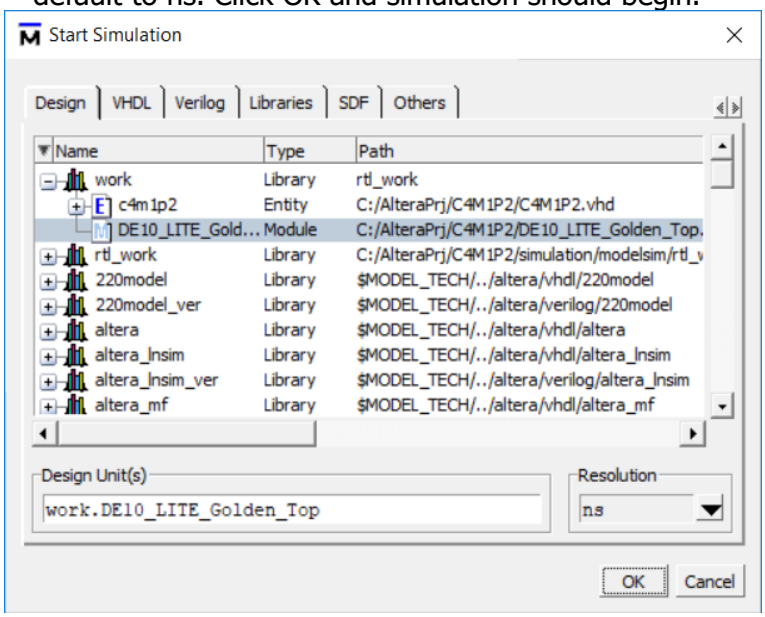
Perform the following steps:

1. Write VHDL code to implement your design. Name it C4M1P2.vhd. The code should have the 4-bit input SW3-0, which should be used to provide the binary number V, and the two 7-bit outputs HEX1 and HEX0, to show the values of decimal digits d1 and d0. The intent of this exercise is to use simple VHDL assignment statements to specify the required logic functions using Boolean expressions. Your VHDL code should not include any IF-ELSE, CASE, or similar statements.
2. Make a Quartus project for your VHDL entity by Project -> Copy Project with the name C4M1P2, or by using the project template as before. Include your VHDL file by adding it to the project. You will need to instantiate your VHDL code into the top-level Verilog template as in Part1.

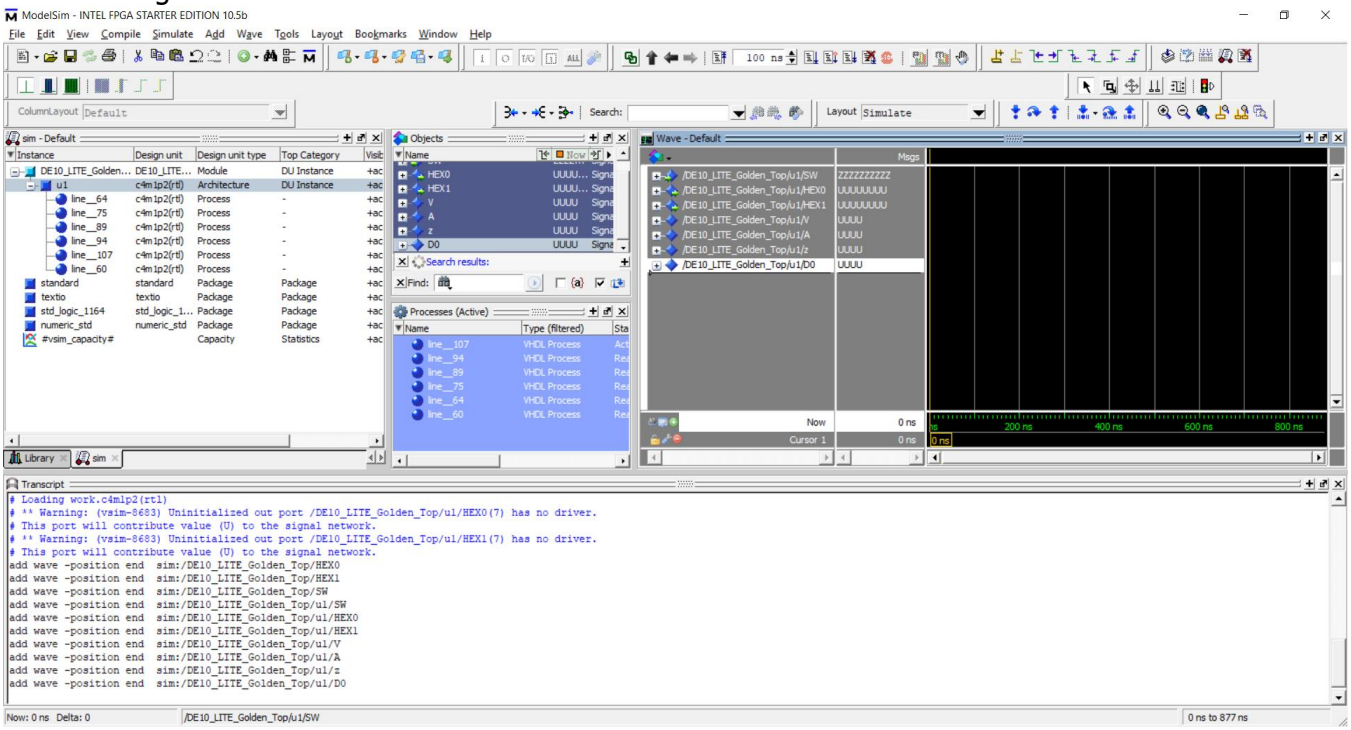
3. Compile the circuit and use functional simulation to verify the correct operation of your comparator, multiplexers, and circuit A.
   a. From Quartus, select Tools and then Run Simulation Tool, and then RTL Simulation. If installed correctly and chosen as the EDA tool in settings, then ModelSim-Altera Edition should start. Click on the Library Tab, and scroll down to work, and you should see something like this:



   b. From the top menus, select Simulate. In the Start Simulation Dialog box, scroll down to work and select DE10_LITE Top level as the design unit. Be sure to change the resolution from default to ns. Click OK and simulation should begin.



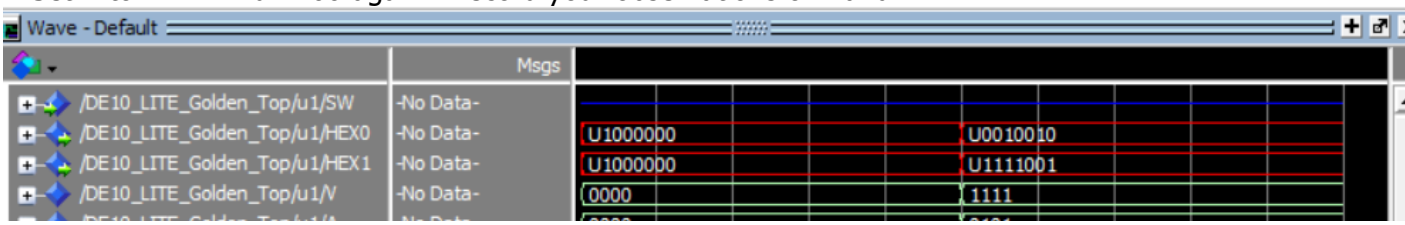
c. Select the Wave Tab. Click in the Instance window and select your instance for the VHDL circuit. Click into the Objects window, Select and then drag all the HEX0, HEX1, SW, V, z, A, and D0 signals to the wave window:



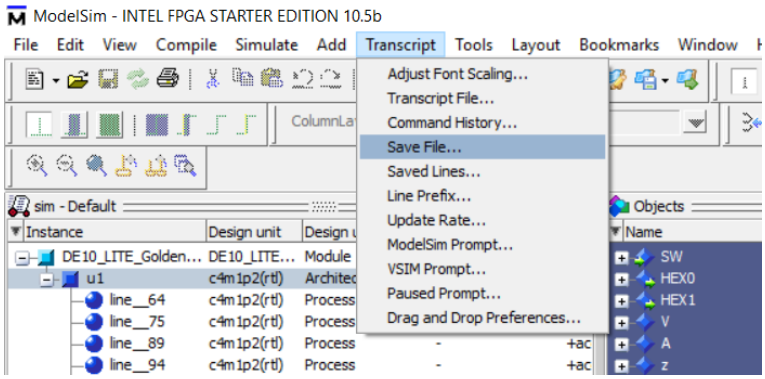
d. Change the run length time to 100 ns.

e. Set V to 0000 by right clicking and choosing Force. Click on the down arrow doc to run, or select from the top menu Simulate – Run – Run 100. Record your observations of z and A.

f. Set V to 1111. Run 100 again. Record your observations of z and A.



g. Be sure to save your ModelSim transcript file by using the top menu Transcript -> Save. Use the name C4M1P2.



4. Download the circuit into the FPGA board. Test the circuit by trying all possible values of V and observing the output displays. From the Compiler Report in Quartus, estimate the number of Logic Cells and % utilization of FPGA logic.

CONGRATULATIONS!!

You have created a Binary-Coded Decimal Display.

PART III – FOUR-BIT FULL ADDER

Figure 2 shows a circuit for a full adder, which has the inputs a, b, and c_i, and produces the outputs s and c_o. Figure 3 show a circuit symbol for the full adder with the truth table in Table 2, which produces the two-bit binary sum c_o s = a + b + c_i. Figure 4 shows how four instances of this full adder module can be used to design a circuit that adds two four-bit numbers. This type of circuit is usually called a ripple-carry adder, because of the way that the carry signals are passed from one full adder to the next. Write VHDL code that implements this circuit, as described below.

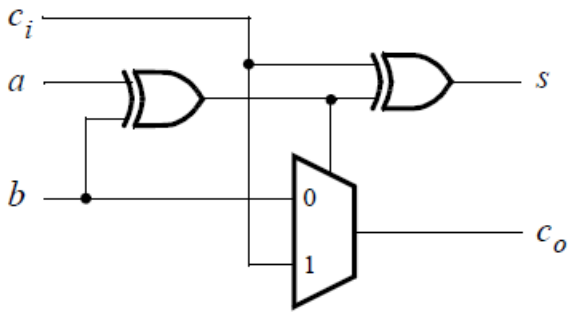


Figure 2: Full adder circuit

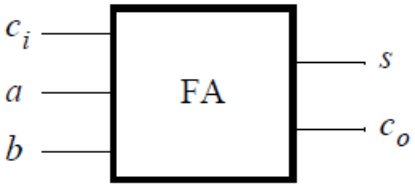


Figure 3: Full adder Symbol

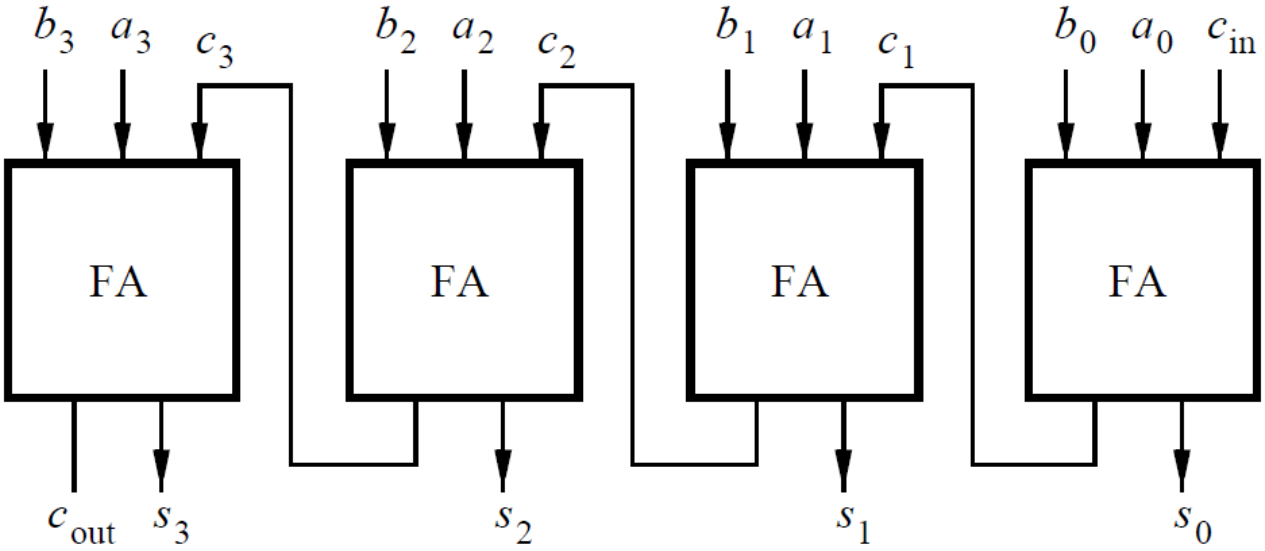


Figure 4: Four-bit Ripple Carry Adder Circuit

$b\ a\ c_i$	$c_o\ s$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

Table 2: Full adder truth table

1. Create a new Quartus project for the adder circuit, using the DE10-Lite as a template. Write a VHDL entity for the full adder subcircuit and write a top-level VHDL entity that instantiates four instances of this full adder.
2. Use switches SW_{7-4} and SW_{3-0} to represent the inputs A and B, respectively. Use SW_8 for the carry-in c_{in} of the adder. Connect the outputs of the adder, c_{out} and S, to the red lights LEDR(4) and LEDR(3:0) respectively.
3. Include the necessary pin assignments for your DE-series board, compile the circuit, and download it into the FPGA chip.
4. Test your circuit by trying different values for numbers A, B, and c_{in}. From the Compiler Report in Quartus, estimate the number of Logic Cells and % utilization of FPGA logic.

CONGRATULATIONS!!

You have created a 4-bit Adder.

PART IV – BCD ADDER

In Part II we discussed the conversion of binary numbers into decimal digits. For this part you are to design a circuit that has two decimal digits, X and Y, as inputs. Each decimal digit is represented as a 4-bit number. In technical literature this is referred to as the binary coded decimal (BCD) representation.

You are to design a circuit that adds the two BCD digits. The inputs to your circuit are the numbers X and Y, plus a carry-in, c_{in}. When these inputs are added, the result will be a 5-bit binary number. But this result is to be displayed on 7-segment displays as a two-digit BCD sum S_1S_0. For a sum equal to zero you would display $S_1S_0 = 00$, for a sum of one $S_1S_0 = 01$, for nine $S_1S_0 = 09$, for ten $S_1S_0 = 10$, and so on. Note that the inputs X and Y are assumed to be decimal digits, which means that the largest sum that needs to be handled by this circuit is $S_1S_0 = 9 + 9 + 1 = 19$.

Perform the steps given below.

1. Create a new Quartus project for your BCD adder. You should use the four-bit adder circuit from Part III to produce a four-bit sum and carry-out for the operation X + Y. A good way to work out the design of your circuit is to first make it handle only sums (X +Y) <= 15. With these values, your circuit from Part II can be used to convert the 4-bit sum into the two decimal digits S_1S_0. Then, once this is working, modify your design to handle values of 15 < (X + Y) <= 19. One way to do this is to still use your circuit from Part II, but to modify its outputs before attaching them to the 7-segment display to make the necessary adjustments when the sum from the adder exceeds 15. Write your VHDL code using simple assignment statements to specify the required logic functions–do not use other types of VHDL statements such as IF-ELSE or CASE statements for this part of the exercise. You will use these statements in the next Part.
2. Use switches SW_{7-4} and SW_{3-0} for the inputs X and Y, respectively, and use SW_8 for the carry-in. Connect the four-bit sum and carry-out produced by the operation X + Y to the red lights LEDR. Display the BCD values of X and Y on the 7-segment displays HEX5 and HEX3, and display the result S_1S_0 on HEX1 and HEX0.
3. Since your circuit handles only BCD digits, check for the cases when the input X or Y is greater than nine. If this occurs, indicate an error by turning on the red light $LEDR_9$.
4. Include the necessary pin assignments for your DE-series board using the board template as before, compile the circuit, and download it into the FPGA chip.
5. Test your circuit by trying different values for numbers X, Y, and c_{in}. From the Compiler Report in Quartus, estimate the number of Logic Cells and % utilization of FPGA logic.

CONGRATULATIONS!!

You have created a 4-bit BCD Adder.

PART V – BCD ADDER REVISITED

In Part IV you created VHDL code for a BCD adder. A different approach for describing the adder in VHDL code is to specify an algorithm like the one represented by the following pseudo-code:

```
1   T0 = A + B + c0
2   if (T0 > 9) then
3     Z0 = 10;
4     c1 = 1;
5   else
6     Z0 = 0;
7     c1 = 0;
```

```
8    end if
9    S0 = T0 - Z0
10   S1 = c1
```

It is reasonably straightforward to see what circuit could be used to implement this pseudo-code. Lines 1 and 9 represent adders, lines 2-8 correspond to multiplexers, and testing for the condition $T_0 > 9$ requires comparators.

You are to write VHDL code that corresponds to this pseudo-code. Note that you can perform addition operations in your VHDL code instead of the subtraction shown in line 9. The intent of this part of the exercise is to examine the effects of relying more on the VHDL compiler to design the circuit by using IF-ELSE statements along with the VHDL > and + operators. Perform the following steps:

1. Create a new Quartus project for your VHDL code. Use switches SW_{7-4} and SW_{3-0} for the inputs A and B, respectively, and use SW_8 for the carry-in. The value of A should be displayed on the 7-segment display HEX5, while B should be on HEX3. Display the BCD sum, S_1S_0, on HEX1 and HEX0.
2. Use the Quartus RTL Viewer tool to examine the circuit produced by compiling your VHDL code. Compare the circuit to the one you designed in Part IV.
3. Download your circuit onto your DE-series board and test it by trying different values for numbers A and B.
4. From the Compiler Report in Quartus, estimate the number of Logic Cells and % utilization of FPGA logic. How does this compare to the number of Logic Cells in Part 4?

CONGRATULATIONS!!

You have completed Module 1!

III. DELIVERABLES

Deliverables include:

1. Recorded Observations, Test Data, and Images

Include observations recorded in a lab notebook, test data taken, and any digital pictures of the proceedings. You will need these in order to answer the quiz questions. Be sure to include your lab notebook in your submission.

2. FPGA Project directory zipped for each Part of the Module

Submit the DE10_Lite_Small and DE10_Lite_Default projects you created. For Module 1, starting with the top level of each part, zip up the entire directory including subfolders, and submit the zip files. Be sure to include the transcripts of simulations in the zip file. Label these files C4M1Px.zip, where x is 1 through 5. This will allow us to replicate your work and help provide feedback on any errors you encounter. With each submission include a ReadMe.txt if appropriate to describe and list the locations of individual file deliverables.

IV. EVALUATION

Any grade awarded pursuant to this project will be based upon deliverables. The following elements will be the primary considerations in evaluating all submitted projects:

1. Reasonable logic utilization and Fmax results within expected bounds.
2. Compilation of hardware designs with no errors.
3. Completion of the project through generation of programming files.
4. Thorough recording of your observations in a lab notebook.

REFERENCES

[1] Intel Altera. (2016). *Laboratory Exercise 2*. [Online]. Available: https://software.intel.com/content/www/us/en/develop/topics/fpga-academic/teach/digital-logic.html