APPLIED HOMEWORK #1

Design Using FPGAs

1 Description and Homework Objectives

AHW1 is to be submitted by the report deadline listed on the course webpage. You need to **first view two** videos (**Structural Verilog & ECE352 CAD Tools**) in that order. Please note that, although this applied homework is significantly shorter than the remaining four, it will require some time to complete. You will demonstrate the work you complete for your report in the "lab" section for which you registered. This will take place in the week where "Applied HW#1 DEMO" is listed on the course webpage. The work for this report should be completed **individually**. At the demo, you can form groups of two (from the same lab section) for the demo work and remaining Applied Homeworks.

After this homework, you should be able to:
1. Launch ModelSim and compile Verilog design files
2. Debug Verilog compilation errors
3. Simulate a test bench in ModelSim
4. Debug Errors, recompile and re-simulate
5. Write structural Verilog for a simple unit

After having demoed (in "Lab") you should be able to:
6. Use the programmer to program the FPGA and verify operation of your design on the FPGA

2 Background: FPGAs and Quartus

A designer can choose from many different implementation technologies for a given application. The most basic choice is between using a microprocessor and a custom hardware circuit. In programming courses, your instructor essentially decided in advance that the implementation of your project was best done using a microprocessor running your software. In this course, we teach the fundamentals of digital design, and implement applications in *hardware*.

An engineer can implement hardware in several ways: full custom, semi-custom (standard cell), and field-programmable. In a full custom design, the engineer designs the entire circuit from the transistor level using no pre-designed elements. This method has the most potential performance but is the most expensive both in time to design and initial cost for one chip. In a semi-custom design, the designer uses a library of standard cells (e.g. logic gates, memory, etc) combined with automated tools to create the circuit. This method produces designs much more quickly than full custom, but is still expensive for initial cost (still need the factory for fabrication). The final method is to use field-programmable chips. In this method, the designer uses a pre-made chip able to implement many different circuits, such as a field-programmable gate array (FPGA). The designer creates their circuit and uses automated tools to map the circuit to the FPGA. This method is the quickest and has the smallest initial cost; however, it also has the worst potential for performance. In this class, because we want to implement many different small circuits in a short period of time, we will use FPGAs.

2.1 ECE352 CAD Tools

2.1.1 Verilog & ModelSim

We will implement our designs in a **H**ardware **D**escription **L**anguage (HDL) called Verilog which is commonly used in industry. We will use a simple subset of Verilog as to not confuse you too much. Testing of your design will be done by placing your design (DUT = Device Under Test) in a test bench. The test bench is also written in Verilog, however, we will provide the test bench to you. The Verilog simulator we use is called ModelSim, and is installed on all CAE machines. You can also install a student edition of ModelSim on your own machine if you have Windows.

2.1.2 Quartus & Altera Cyclone IV FPGA

During the demo of the applied homework you will "map" your design to the FPGA via a tool called Quartus. Quartus is written by Altera (manufacturer of the FPGA in use). We will simply use Quartus to compile our Verilog designs and program the FPGA board. Then you will test out your design on the board.

Note: Even if you use ModelSim on your own machine, be sure that you get your CAE account several days before your demo and put all your files on your CAE network file space. You must have a CAE account and your design files at your demo session to get credit!

Design Entry

For this course, we will provide Verilog shells of the designs we wish you to implement. You will complete the "guts" of the Verilog to implement the design. Design entry can be done in any text editor. You can use the editor built into ModelSim or an external editor of your choice. Notepad++ is not a bad choice. We will use the latest version of Verilog referred to as System Verilog. All of our design files will end with a **.sv** extension.

Functional Simulation

Once your design is created, you have to simulate it to ensure it is functionally correct. A self-checking Verilog test bench will be provided. You will launch ModelSim and simulate the test bench. Invariably your design will have bugs and you will have to view waveforms and figure out why your design is not working properly. You will fix the bugs and iterate until the self-checking test bench says your DUT is good. In the course ECE551 (352 is a prerequisite for it) you will use Verilog in more depth and write all blocks and test benches from scratch.

Synthesis (Implementation)

Once you have a functionally correct design, you can then perform implementation, which produces a file that can be loaded into an Altera FPGA to prototype the design (ie, actually implement the hardware circuit). The new file describes our circuit in terms of the FPGA's internal logic elements. Implementation consists of five main steps:

Map: The tool optimizes our circuit (eliminating unnecessary logic), and then breaks the circuit into pieces that can fit into a logic element (LE)

Place: The pieces from the mapping step are assigned to specific LEs and other FPGA resources. Placement tries to simplify the next step (routing) by assigning pieces with many interconnections to LEs that are near each other.

Route: All interconnections between the pieces of the circuit and the inputs/outputs are now assigned to specific routing resources of the FPGA. In other words, we connect the pieces that were placed in the previous step.

Timing: After mapping and place and route, locations of the look-up tables, flip-flops, input, outputs and all of the interconnection paths are known. Based on this (particularly the number of transistors along each interconnection path), delay estimates for various paths through the circuit and delay statistics for the FPGA can be calculated. These values are placed in a file which is used to perform timing analysis.

Configuration: The final implementation step is the generation of the configuration file—the sets of 1s and 0s to load into the FPGA configuration SRAM that implements the circuit. The file we use is an SOF (*SRAM object file*). When this file is loaded into the FPGA's configuration SRAM, the look-up tables are filled, the multiplexers are set, and the routing transistors turned ON or OFF to create the interconnections.

All these steps produce report files that record what happened during that step, related statistics, and whatever warning or error messages were output (if any). A report viewer is provided to look at the reports. Another file produced by place and route is the Pin file that indicates which pin is assigned to each of the FPGA inputs and outputs. This file is useful for making board-level connections to the FPGA chip and knowing which pins were used for what purpose..

Prototyping

Prototyping is the final verification step. With actual hardware available, it is possible to apply large numbers of inputs rapidly, far faster than can be done in simulation. Also, delay can be accurately measured, and the clock frequency can be varied for more detailed tests. For simple combinational and sequential circuits, we can apply enough combinations or sequences to fully demonstrate the function is correct. But for more complex circuits, this is impossible. Further, due to the limited time we have available, testing of such complex circuits must be selective.

At any one of the major steps listed, you may discover a problem with the design. To correct the problem, you will usually have to return to the design entry step, modify the design, and repeat most of the above steps. When this occurs at the prototyping stage, the number of steps to be repeated is high, so it is important to find problems as early as possible using very thorough simulation at the functional level to minimize the length of the entire design cycle.

2.2 Prototyping Hardware

We will use an Altera Cyclone IV FPGA board for our hardware demonstrations. The boards are powered by an external power supply (and two on-board voltage regulators), and include many resources attached to the FPGA for use in a wide variety of projects. In this homework, we will use only a limited subset of these resources. Notably, we will use small slide switches, LEDs, seven-segment LED displays, and pushbuttons on the board that operate through an IC that eliminates contact bounce problems.

A .qsf (Quartus Settings File) will be provided to ensure that the inputs and outputs of our circuit described in Verilog are correctly mapped to FPGA pins that connect to the intended devices on the board. It is important to verify that you have named your signals correctly to prevent damage to the FPGA or the external devices. Please be **very careful** in using these boards, follow instructions carefully, and verify everything **before** powering up the board.

Altera Cyclone IV Series FPGAs

The information in this box is not needed to complete this homework. It is provided in case you are interested.

Look-up tables (LUTs) implement combinational logic functions in most current FPGA architectures. A LUT is a very small static random-access memory (SRAM) that holds the truth table for the function it implements. The inputs of the LUT act as an address for the SRAM, and the output is the value held at that address (that line of the truth table). A complete logic circuit is formed by programming the contents of the LUTs as needed, then setting other SRAM bits that control how the LUTs are connected to one another through tristates and multiplexers. These values are loaded during FPGA configuration (generally each time the FPGA is powered on), which can take 100s of milliseconds, depending on the size of the FPGA.

The LUTs in the Cyclone FPGA are 4-LUTs: they have four inputs. Any logic function with 4 inputs and 1 output can be implemented using a single 4-LUT. The fundamental structure of the Cyclone FPGA is known as a logic element (LE), shown below in Figure 1, and each LE contains 4-input LUT and a flip-flop (a storage element we'll discuss later).

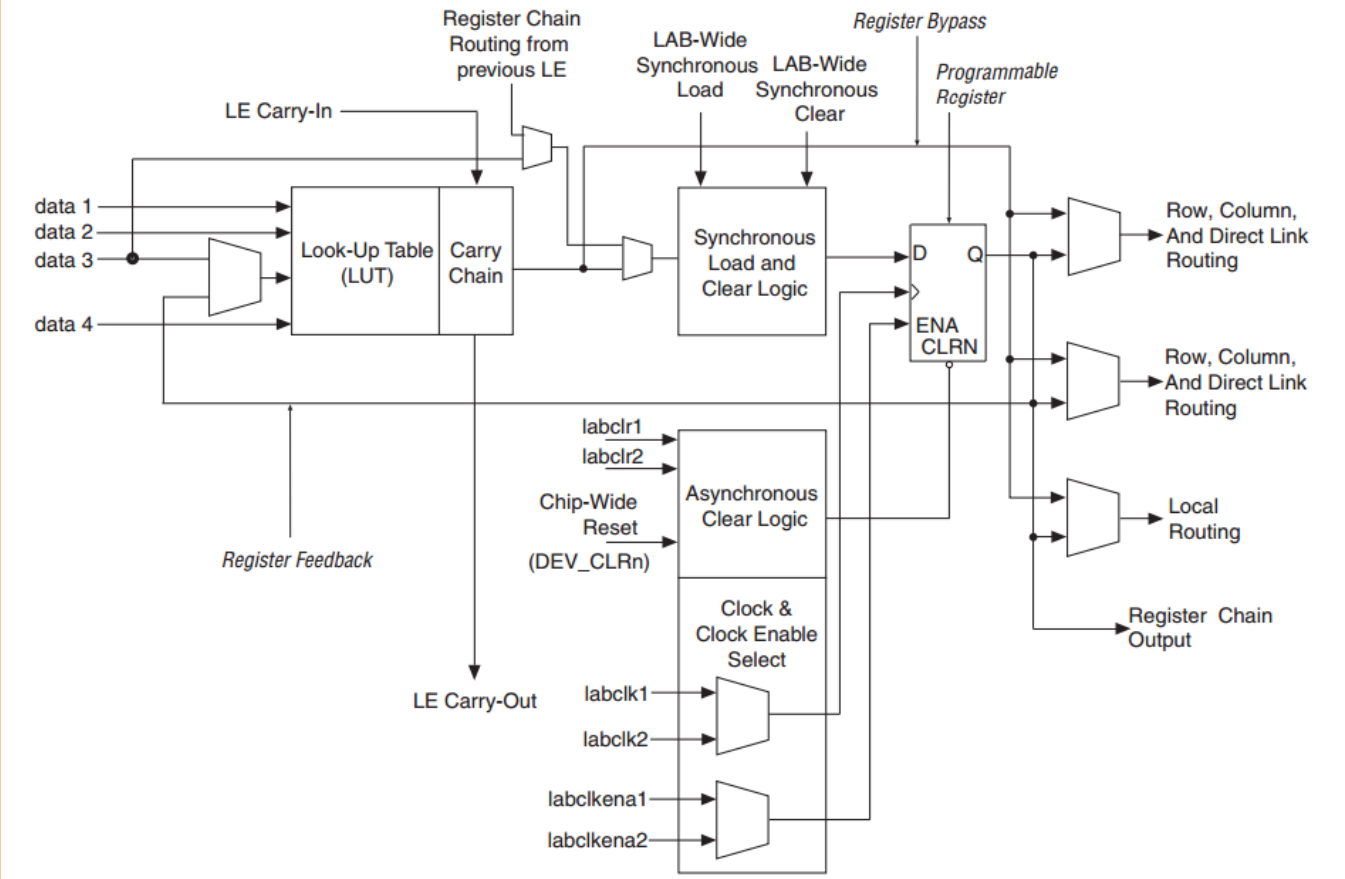


Figure 1: Altera Cyclone IV Logic Element (LE) [1]

The process of deciding which LUT in the FPGA should implement which logic gate, and which wires should implement which signals between gates is very complex. These choices affect whether or not the circuit fits in the FPGA, and if it does, how fast it can run. Fortunately, this is done automatically for us by the Quartus development tool. In a realistic commercial design, you would probably have to "tweak" certain tool parameters to meet your particular design requirements and produce an "optimal" design.

[Remember "optimal" will mean different things for different designs and situations (minimal delay, minimal area, minimal power, minimal skew, etc). This is why companies hire highly-paid hardware designers!]

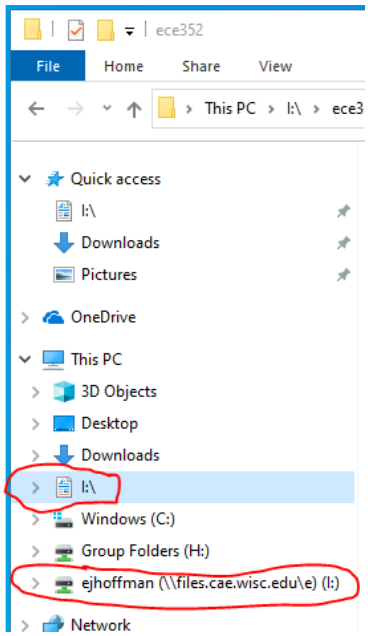
3 Applied Homework Tasks

This must be done ***individually***. Do not just watch someone else do the tutorial. You need to do the tutorial yourself to help you understand how to use the tool, which you will need to use for written and applied homework throughout the semester. All work submitted for AHW1 **must be your own work**, based on working through the tutorial and this document.

Be sure you have watched the two videos (Structural Verilog & ECE352 CAD Tools**) before attempting this AHW.**

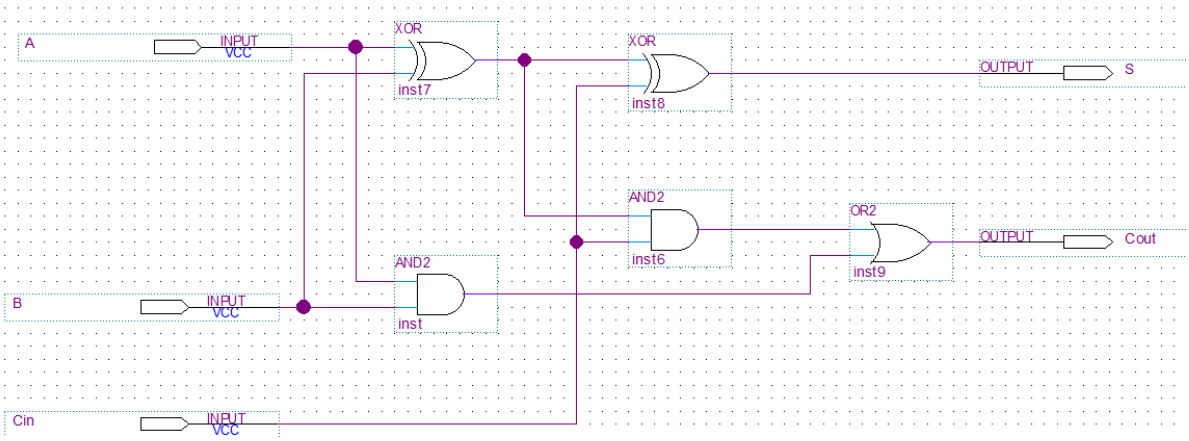
In this assignment, you will design and test an FPGA-based "full adder" – a circuit that adds two bits and produces a sum bit and a carry bit, and use it to create a larger adder. Later during the demo, we will configure this circuit onto the FPGA board and use LEDs to display the input and output values.

3.1 Full Adder Design/Simulation/Debug/Submission

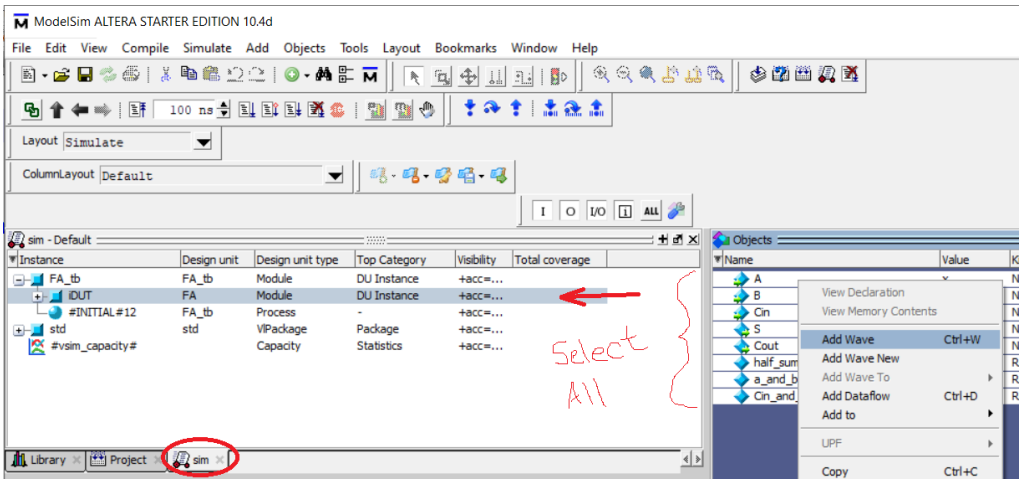


1) Create an **ECE352** folder under your **I:** drive on you CAE account. Then create an **AHW1** sub-folder under that. Your **I:** drive on CAE is "your" space and will follow you no matter which computer you use to logon. Your **I:** drive might show up in two places like in the image to the side, but are the same area. Your **I:** drive is also mirrored to linux, so is available if you ever login there.
2) Download the **AHW1.zip** file. Extract the contents of the zip archive to your newly created **I:\ECE352\AHW1** directory
3) You now need to create a new ModelSim project that includes the downloaded verilog files. ***See the 2nd very short video under "ECE352 CAD Tools Videos" on how to create a ModelSim project.***
4) The first circuit you will be editing is **FA.sv**. You can either open in using the text editor built into ModelSim, or the text editor of your choice. I personally like Notepad++.
5) **FA.sv** contains a shell of a full adder circuit. A full adder will add to operand bits (**A** & **B**) and a possible carry in from a lower order bit position (**Cin**) and produces a sum (**S**) for that bit position and a carry out (**Cout**) that can service a higher order bit position.
6) Complete the code for **FA.sv** by filling in any needed intermediate signals and writing the structural Verilog to implement the functionality (**do NOT** use assign statements. You will need to use the built-in gate types)

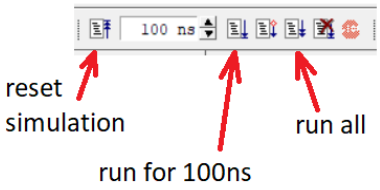
   The picture below shows how a full adder is constructed



7) When you have finished save the file and compile it within ModelSim. Does it compile (green check mark) or does it have errors (red X). If it has errors double click on the red text in the "Transcript" window and fix the errors that are mentioned.
8) If you have not already, also compile the provided test bench (**FA_tb.sv**).
9) Now simulate the test bench
   a. Start the simulator by either Simulate=>Start Simulation and expanding the **work** library and choosing **FA_tb**. Or…my preferred way by going to the "Library" tab, expanding the **work** library selecting **FA_tb** and right clicking to launch.
   b. Once the simulation tab is up select the DUT (iDUT) then click and shift click to select all the signals of your DUT. Finally right click and select "Add Wave" (see figure below).



c. Now run the entire simulation using the simulation control in either the waveform window or the main ModelSim window.

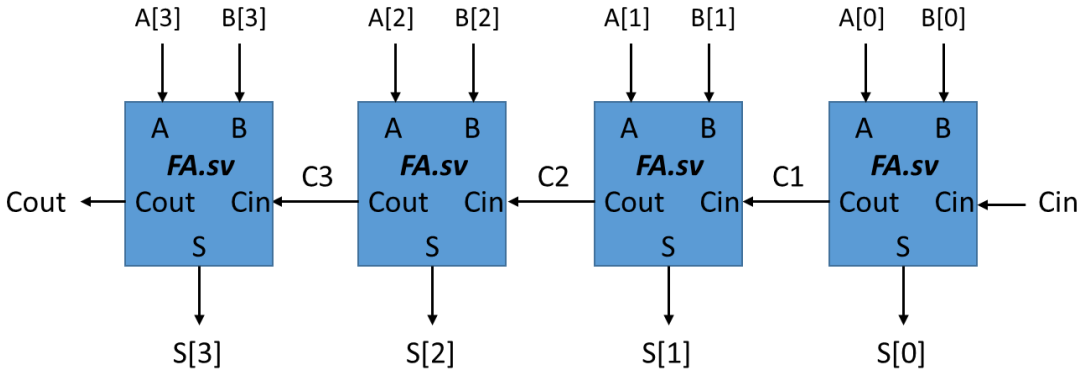


d. Did your design run error free? If so you got lucky this time. If not…start debugging. Don't like debugging? … pick a different career path. When debugging remember you do not have to stop simulation in ModelSim after editing your verilog. You can edit and save the verilog, recompile, reset the simulation (see graphic above) and then re-run the simulation.

10) Once your simulation of FA.sv runs with no errors take a screen image of the waveforms. Call this **FA_sim.jpg**. Submit **FA_sim.jpg** and **FA.sv** to the dropbox for AHW1.

3.2 Ripple Carry Adder Design/Simulation/Debug/Submission

Next you will create a 4-bit ripple carry adder (RCA) by instantiating 4 copies of your newly created FA module and threading the carry chain. This ripple carry adder accepts two 4-bit vectors (**A[3:0]** &**B[3:0]**) and a carry in (**Cin**). It produces a 4-bit sum (**S[3:0]**) and a carry out (see image below).



1. A shell called **RCA4.sv** is available. Edit it to flush out the ripple carry adder implementation. Implement it as shown with 4 discrete instantiations of **FA**, and 3 intermediate nets (C1,C2,C3). Your instantiations of FA should use connect by name, not connect by reference order. Meaning…connections like FA iFA0(.A(A[0]), .B(B[0]), …

2. Compile RCA4.sv in ModelSim (fix any compilation errors)
3. Compile the provided test bench **RCA4_tb.sv**
4. Simulate **RCA4_tb.sv**. Does your DUT perform correct? If not start debugging.
5. Once your RCA4.sv design is passing the test bench perform a screen capture of its waveforms. Call it **RCA4_sim.jpg**. Submit **RCA4_sim.jpg** and **RCA4.sv** to the dropbox for AHW1.

3.3 Vectored Instantiation…A smarter way

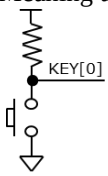
1. Now we will ask you to make a 16-bit adder using your FA cell. It would be pretty tedious to instantiate 16 separate copies of the FA cell. There is a smarter way.
2. Prof. Lipasti presented a smarter way of doing multiple instantiations of a block. A "vectored" way of doing it.
3. A file **RCA16.sv** is provided. You will implement a 16-bit adder in this file using vectored instantiation of your **FA** cell. So you will only need one line to infer all 16 instantiations. You will also have to have a separate line to drive Cout.
4. Compile **RCA16.sv** in ModelSim (fix any compilation errors)
5. Compile the provided test bench **RCA16_tb.sv**
6. Simulate **RCA16_tb.sv**. Does your DUT perform correct? If not start debugging.
7. Once your RCA16.sv design is passing the test bench perform a screen capture of the transcript window showing it passes.. Call it **RCA16_yahoo.jpg**. Submit **RCA16_yahoo.jpg** and **RCA16.sv** to the dropbox for AHW1.
8.

3.4 AHW1 (top level).

There are 18 switches on the DE2 FPGA board (**SW[17:0]**). There are also 18 red LEDs (**LEDR[17:0]**). We will use switches [17:14] to form **A[3:0]** into our **RCA4**. We will use switches [3:0] to form **B[3:0]**.

For convenience of display the state of **SW[17:14]** will also read out on **LEDR[17:14]** and the state of **SW[3:0]** will read out on **LEDR[3:0]**.

The DE2 FPGA board also has four push buttons (**KEY[3:0]).** We will use **KEY[0]** to form **Cin** to our **RCA**4. We will not have an LED to represent Cin. **NOTE**: The push buttons are a push button to ground. Meaning they are active low. We want it to behave as though Cin==1 when we are pushing the button.



There are 8 green LEDs on the DE2 FPGA board (**LEDG[7:0]**). We will use bits [3:0] to represent the sum from our **RCA4**. We will use **LEDG[7]** to represent the carry out of **RCA4**.

1. There is a shell of the top level (**AHW1.sv**). Flush this out to instantiate RCA4 and make the specified connections to the top level ports. You do not need to simulate this, or even submit it for this AHW, but you will have to have it done to pass the demo portion of this homework, so get it done. It is pretty simple.

NOTE: The remaining steps are a demonstration of how you will program the FPGA during lab. You will not be able to complete these steps without the FPGA, but please follow along so that you know how to program the FPGA when you attend your lab session.

11) In your lab demo, you will need to compile and load your project onto the DE2-115. The image below shows you how to load a project onto the DE2-115. Make sure that the USB-Blaster is listed in the Hardware Setup prior to programming. The video will show you how to select the USB-Blaster if it is not listed.

12) Verify that the .sof file is pointing to the current directory. There shouldn't be any characters preceding AHW1.sof.

13) If the USB-Blaster is listed, select the Start button. As the FPGA programs, you will see the progress indicator go from 0% to 100%.

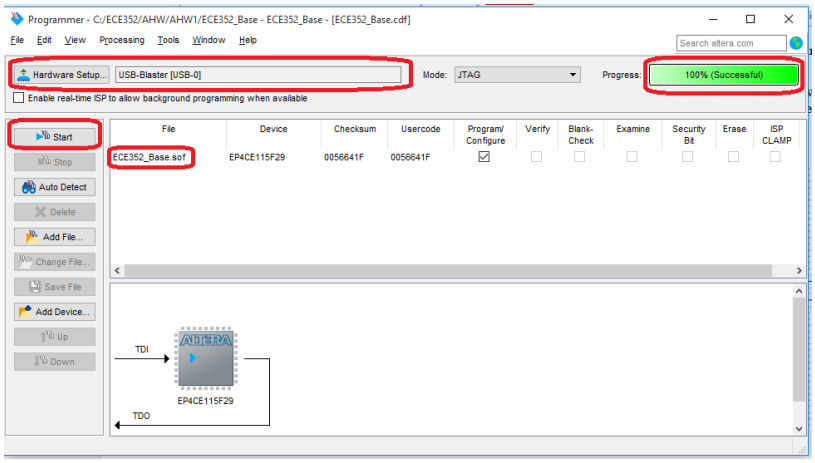


Figure 2

Programming the FPGA

Assemble the files listed at the end of this document, and upload them to the AHW1 dropbox. You must submit all files by the Applied Homework deadline listed on the class webpage. Later, in your demo session (lab), you will further verify and demonstrate your circuits by loading them onto an FPGA board so that you can use the switches to control the circuit inputs, and the LEDs to see the outputs of the circuit.

4 Applied Homework Submission

You will upload a number of files to the AHW1 dropbox be sure that your files are named correctly.

Files to be submitted in advance of the demo:

File Name:	**Description:**
FA.sv	Structural Verilog of full adder
FA_sim.jpg	Waveforms showing functionally correct simulation of full adder
RCA4.sv	Structural Verilog of 4-bit ripple carry adder
RCA4_sim.jpg	Waveforms showing functionally correct simulation of 4-bit adder
RCA16.sv	Structural Verilog using vectored instantiation.

File Name:	Description:
RCA16_yahoo.sv	Capture of transcript window showing simulation passed

This submission is due by the Applied Homework deadline listed on the class webpage.

Note that this deadline is in the week BEFORE the related demo session!

5 Applied Homework Demonstration

The following step in the tutorial will be completed in lab.

1. **After** you have successfully compiled your 4-bit full adder, you will load the resulting image onto the DE2-115 development platform. Use switches SW[17:14] & SW[3:0] and KEY[0] to supply inputs to your 4-bit full adder. Observe the resulting 4-bit number on LEDG[3..0], and carry out on LEDG[7] Demonstrate to you TA that your 4-bit full adder functions correctly.
2. After your TA has verified your 4-bit full adder, you will be given a 3-input Boolean equation to implement in structural verilog

   The following equation is an example of the type of equation that will be given to you: $F = AB + A\overline{C} + \overline{A}C$