RUTGERS, THE STATE UNIVERSITY OF NEW JERSEY

ECE493 SPECIAL TOPICS

Hardware/Software Design of Embedded Systems Laboratory

Fall 2013

Contents

1 Lab 1 - Introduction to FPGA's and VHDL **3**
1.1 Introduction . . . 3
1.2 The DE2-115 FPGA Development Board . . . 3
1.3 Pre-Lab . . . 4
1.4 Working with Quartus II . . . 4
1.4.1 Creating a New Project . . . 4
1.4.2 Creating an Empty File . . . 6
1.4.3 Writing VHDL Code . . . 7
1.4.4 Setting Pin Assignments . . . 7
1.4.5 Compiling Hardware . . . 8
1.4.6 Testing Hardware with Waveforms . . . 9
1.4.7 Uploading Hardware to Device . . . 9
1.5 Activities . . . 11
1.5.1 Implementing Logic . . . 11
1.5.2 7 Segment Display Decoder . . . 11
1.6 Lab Report . . . 12

2 Lab 2 - Latches, Flip-Flops, and Counters **13**
2.1 Introduction . . . 13
2.2 Pre-lab . . . 13
2.3 Lab Activities . . . 13
2.3.1 Latches . . . 13
2.3.2 Flip-Flop . . . 14
2.3.3 Counters . . . 14
2.4 Lab Report . . . 14

3 Lab 3 - Complex Addition Systems **16**
3.1 Introduction . . . 16
3.2 Pre-lab . . . 16
3.3 Lab Activities . . . 16
3.3.1 Half Adder . . . 16
3.3.2 Full Adder . . . 16
3.3.3 Full Adder Based ALU . . . 17
3.4 Lab Report . . . 18

4 Lab 4 - Finite State Machines **19**
4.1 Introduction . . . 19
4.2 Lab Activities . . . 19
4.2.1 FSM . . . 19
4.2.2 . . . 19

5 Lab 5 - A Simple Computer **20**
5.1 Introduction . . . 20
5.2 Lab Activities . . . 20
5.2.1 Design an ALU . . . 20
5.2.2 Design a RAM . . . 20
5.2.3 Design the Program Counter . . . 20
5.2.4 Create a VGA Driver . . . 20

6 Final Project **21**
6.1 Introduction 21
6.2 Requirements 21
6.3 Project Ideas 21
6.4 Deliverables 21

1 Lab 1 - Introduction to FPGA's and VHDL

1.1 Introduction

This lab will introduce you to the Altera DE2-115 FPGA Development Board. The DE2-115 contains all of the hardware necessary to prototype and create various hardware configurations on the Altera Cyclone IV FPGA chip that will be used throughout the course of this lab. By completing this lab, you will have an understanding of all the hardware contained on the FPGA development board, along with an understanding of how to connect peripherals to the development board. Lastly, this lab will go over the standard template for designing hardware in the VHDL programming language. All this will be accomplished by following the Quartus II introductory packet along with the following activities.

1.2 The DE2-115 FPGA Development Board

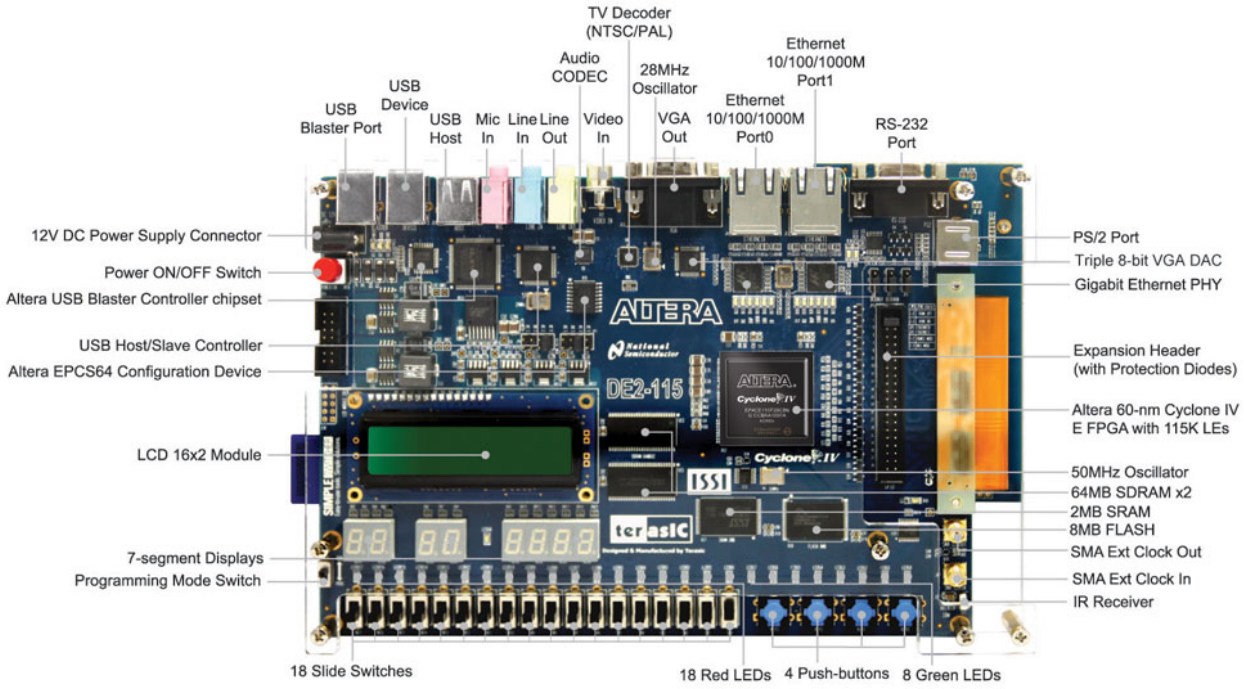


Figure 1: The DE2-115 FPGA Development Board

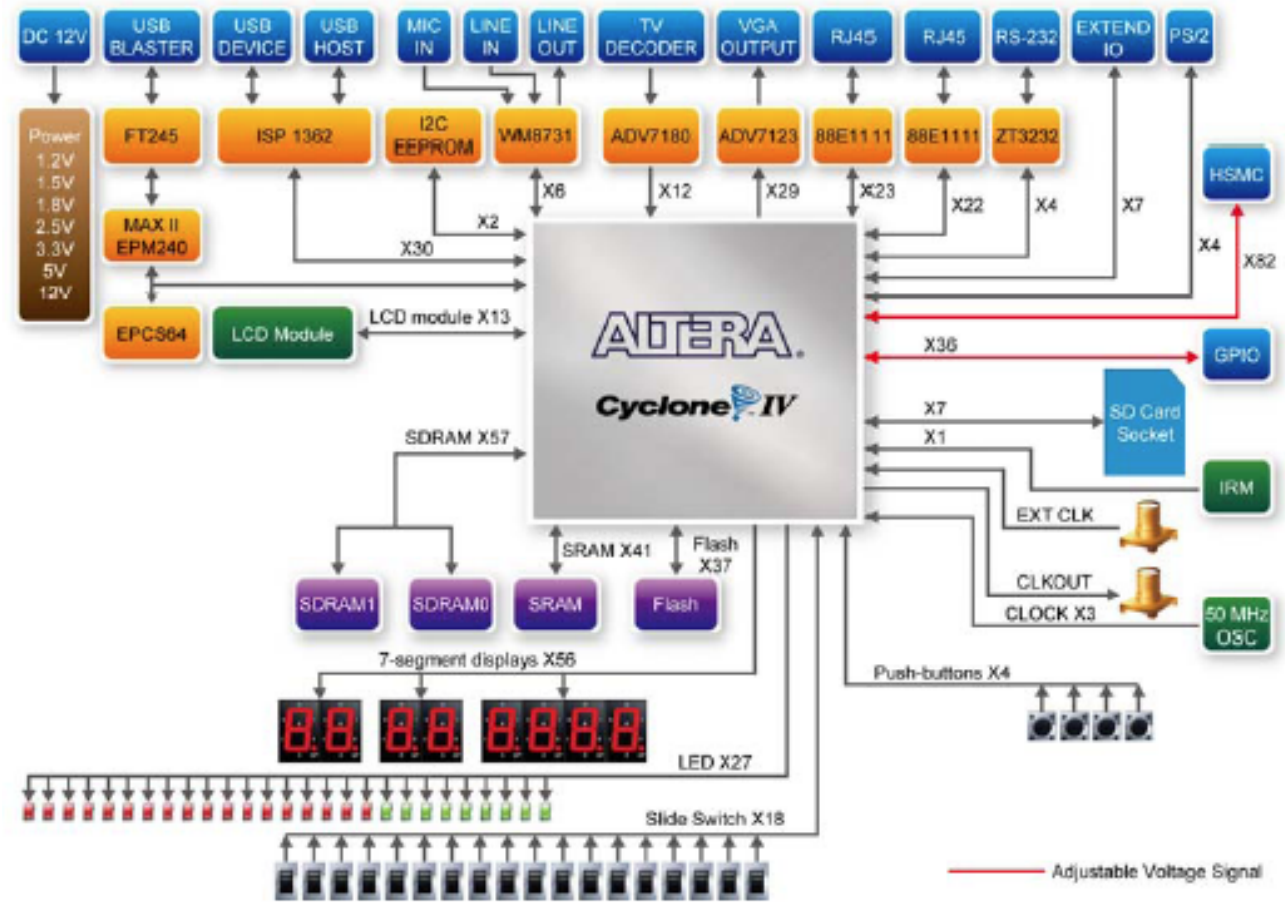


Figure 2: The DE2-115 Block Diagram

1.3 Pre-Lab

Before attempting this lab, you should complete the following:

- Download and install the *Quartus II 13.0 Web Edition Software* from the Altera website to your personal computer
- Download and install the *Altera University Installer Software* located at http://www.altera.com/education/univ/software/upds upds.html
- Download and read the following documents from the Sakai Resources pags; *DE2-115 User Manual*, *Quartus II Introduction*

1.4 Working with Quartus II

1.4.1 Creating a New Project

1. In order to begin working on your first FPGA project, you must open the program Altera Quartus II. To begin a new project goto **File → New Project Wizard** as shown in figure 3

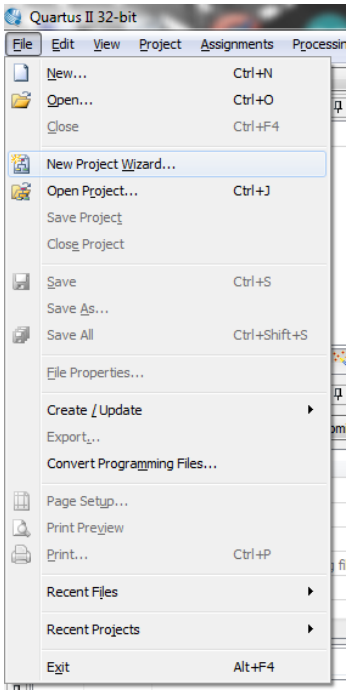


Figure 3: Create a new project menu

2. When the *New Project Wizard* window opens click **NEXT**
3. Create a folder in your Z-drive called *FPGA Lab*
4. Create a folder in FPGA Lab called *lab1*
5. Set the working directory to lab1
6. Name the project *lab1*
7. Click **NEXT** to proceed
8. Skip this step, click **NEXT** to proceed
9. Under Device Family select *Cyclone IV E*, set the package to *FBGA*, pin count to *780*, and speed grade to *7*. In the Available devices list, look for and select *EP4CE115F29C7* as shown in figure 4.

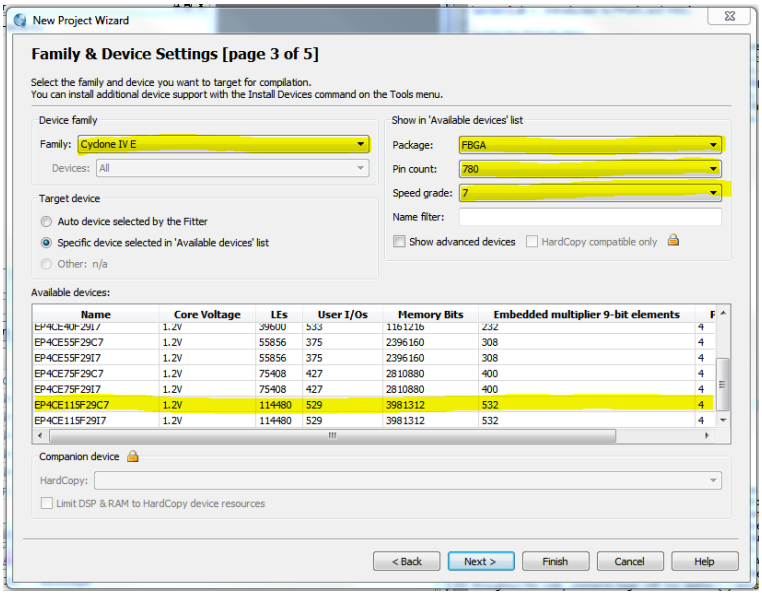


Figure 4: Device selection menu

10. Click **FINISH** to begin your new project

1.4.2 Creating an Empty File

1. Goto **File → New → VHDL File → OK** as shown in figure 5

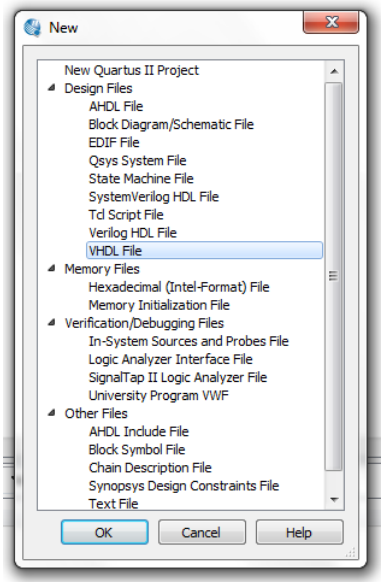


Figure 5: Create a new file menu

2. Save this new file by going to **File → Save As →** ***lab1.vhd*** **→ SAVE** as shown in figure 6

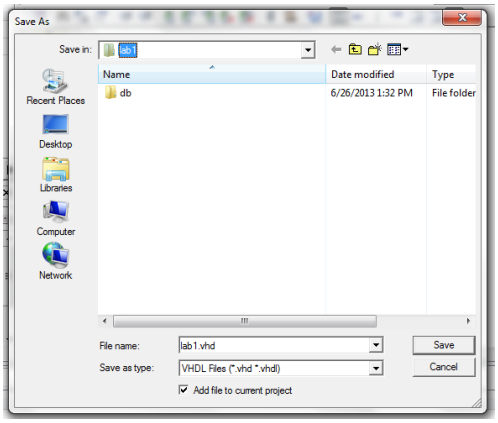


Figure 6: Save As dialog box

1.4.3 Writing VHDL Code

The following code block shows how to interact with the switches and LEDs on the DE2-115. Notice how the program begins with importing the ieee library which contains all of the basic logic primitives as established within the IEEE standard 1164. When working in industry it is common for large companies to create their own libraries as well. Every VHDL file should contain at least one entity (module) that is the same as the name of the file. An entity contains information about the structure of the module such as how many inputs/outputs (I/O) and what type of logic to expect at the I/O. Finally we define the entity in an architecture block, this section does the work on the hardware. As can be seen, this code is setting the red LEDs as defined in the array to the accompanying switches on the board. Take note on the use of comments throughout the code, comments begin with two dashes (–) and should always be used to describe what you are trying to accomplish, this way someone else who reads your code will understand it easily and your code will look more professional.

```
1  -- Import logic primitives
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4
5  -- Simple module that connects the SW switches to the LEDR lights
6  ENTITY lab1 IS
7  PORT ( SW: IN STD_LOGIC_VECTOR(17 DOWNTO 0); -- Initialize switches as an input
8       LEDR: OUT STD_LOGIC_VECTOR(17 DOWNTO 0)); -- Initialize red LEDs as an output
9  END lab1;
10
11 -- Define characteristics of the entity lab1
12 ARCHITECTURE Behavior OF lab1 IS
13 BEGIN
14       LEDR <= SW; -- Assign each switch to one red LED
15 END Behavior;
```

1.4.4 Setting Pin Assignments

The Cyclone IV chip on the DE2-115 contains 780 pins. The majoprity of these pins have copper traces to the hardware the control on the development board. As a result, Altera has provided a file that contains all of the pin assignments so that you can interface directly with the pins in your VHDL code. To set the pin assignments:

1. Goto **Assignments → Import Assignments → Select *D*E2-115.qsf** as shown in figure 7.This file can be dowloaded from the Altera website

2. Then click **ADVANCED → check Global Assignments → Ok** as shown in figure 8

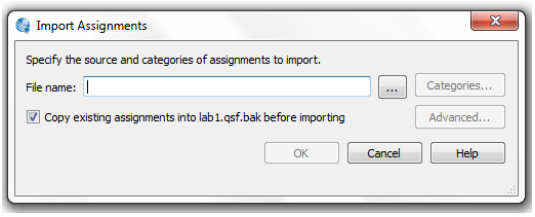


Figure 7: Import assignments dialog box

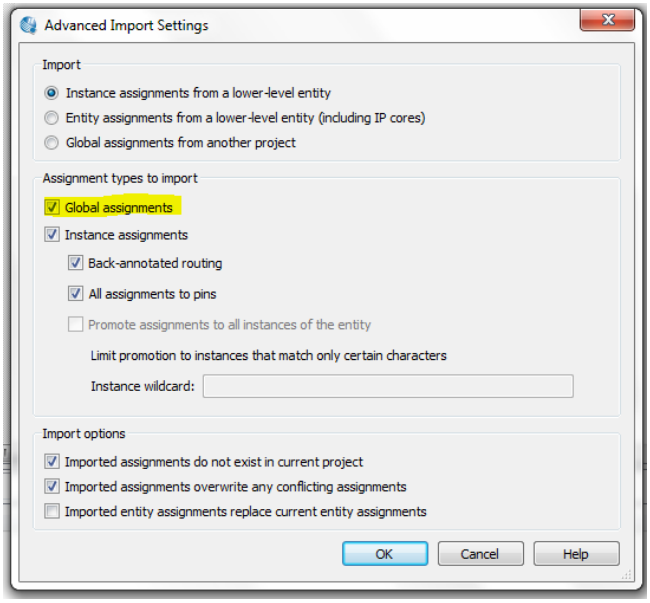


Figure 8: Import assignments advanced options menu

1.4.5 Compiling Hardware

1. To begin compiling your hardware, **Processing → Start Compilation** or you may press *Ctrl* + *L* on your keyboard

If the project compiles successfully, you may proceed to uploading the hardware. Otherwise if you have any errors you should debug your code. It is helpful to note that the first error should be solved first which will make it easier to solve the other errors. A succefull compilation will look like figure 9.

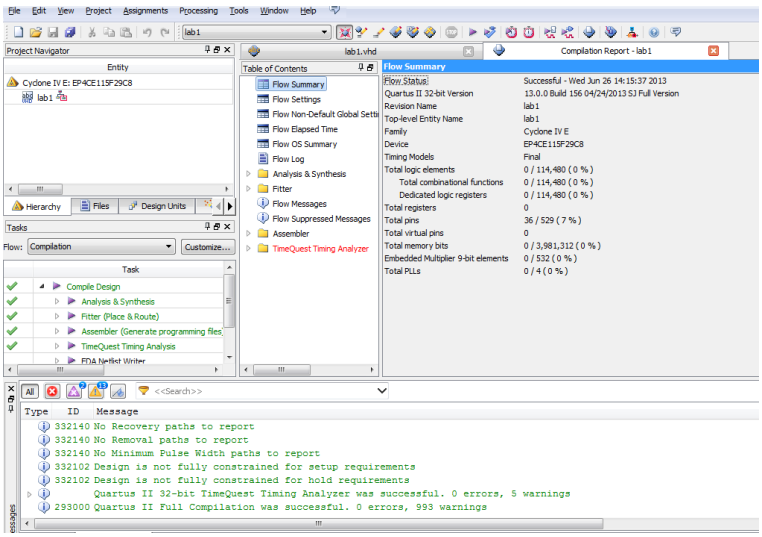


Figure 9: Successful completion of compilation

1.4.6 Testing Hardware with Waveforms

All hardware implementations should be tested for correctness before uploading to the FPGA device. To accomplish this, please follow the tutorial titled *Quartus_II_Simulation.pdf* listed under Sakai resources.

1.4.7 Uploading Hardware to Device

Once the hardware has compiled successfully, goto **Tools → Programmer** to open the hardware upload options. There are two typical modes for uploading hardware and it is important to understand when to use them. This can be seen in figure 10.

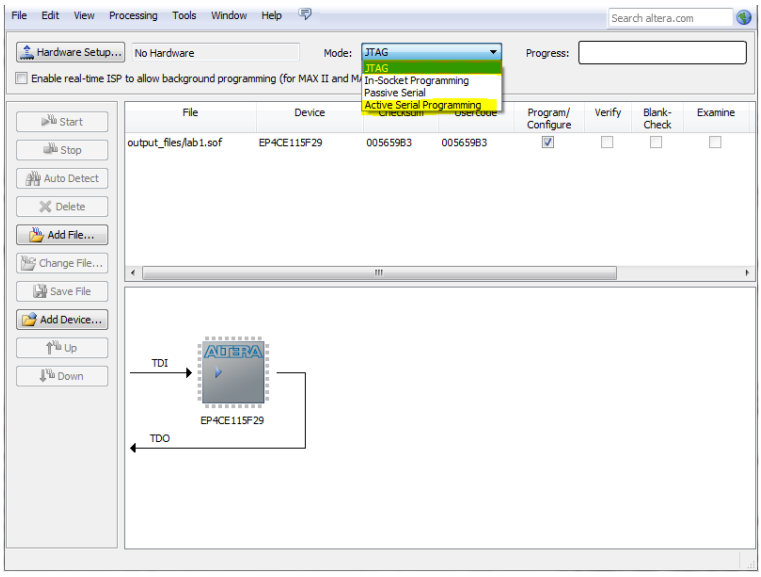


Figure 10: Modes for uploading hardware to the FPGA device

1. JTAG or Joint Test Action Group
    - This method loads the hardware directly to the FPGA chip

- The FPGA is unable to save its current state so if the power is turned off the programmed hardware will disappear
- To program the FPGA with this method all you need to do is connect the USB cable to the development board and ensure that under **Hardware Setup** that USB-Blaster is selected. Then you must goto **Add Files** and add your compiled *lab1.sof* file
- Press **START** to upload your hardware, in a few moments you should see your development board behaving as instructed by your code

2. Active Serial

- This method loads the hardware on to the on-board configuration device. What this means is that the hardware description is saved into memory and is loaded onto the FPGA chip whenever the board is powered on. This method is more desirable because it allows the FPGA to work without being connected to the computer and only to an external power source.
- Before beginning this method you should first check that under **Assignments → Device → Device and Pin Options** are configured in the same way as figure 11, if not you must set the configuration scheme to *Active Serial* and also set the configuration device to *EPCS64*. If this was not set you must recompile your hardware.

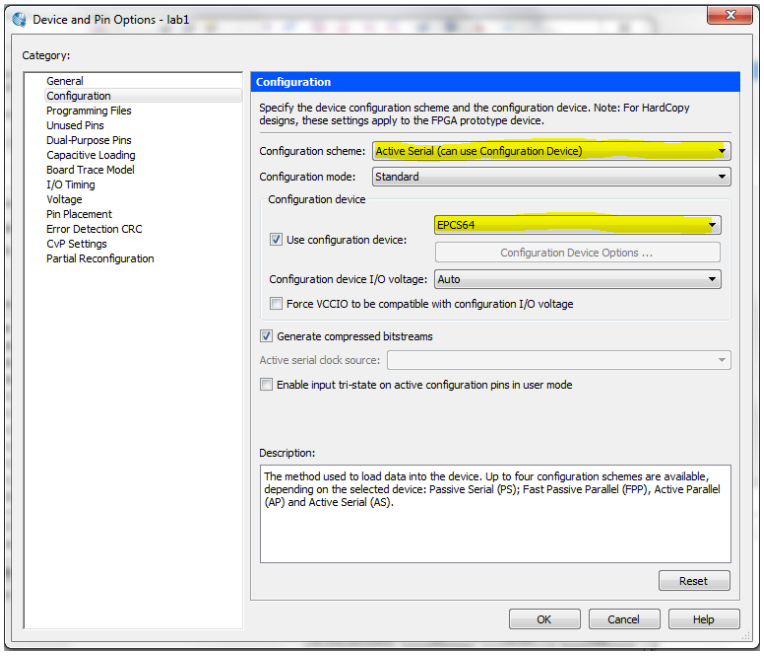


Figure 11: Active Serial Configuration Settings

- Next once again ensure that the hardware is set to *USB-Blaster* and that the mode is set to *Active Serial*
- Click on **Add Files**, and select *lab1.pof*
- Ensure that the development board is switched to *PROG*
- Click **START** to begin programming. This method takes slightly longer
- Switch the board back into the *RUN* position and verify that your logic is behaving properly

1.5 Activities

1.5.1 Implementing Logic

Implement the hardware from the circuit in Figure 12. The inputs should come from SW(1) and SW(2) and the output should be shown on any of the available LEDs. Use the implemented circuit to test and create a truth table with your results and place it within a comment in the program file.

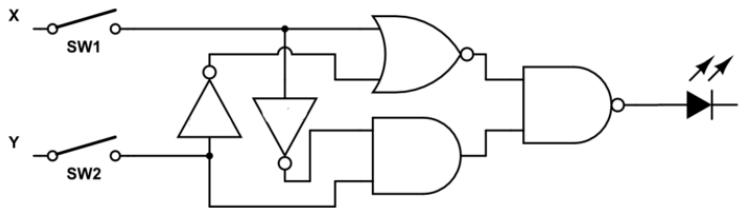
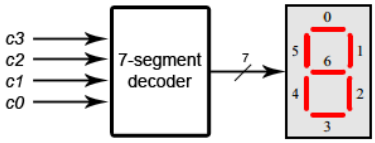


Figure 12: Circuit for activity 1

1.5.2 7 Segment Display Decoder

The 7-segment display is comprised of 7 LEDs that are arranged in such a way that allows for the creation of the numbers 0-9 and a select few characters with some clever use. Figure 13 shows the block diagram and output table. Your task is to create a 4 input, 7 output decoder that will display a number from 0-9 and the letters A-F. To accomplish this task, you should program the switches SW(0) - SW(3) to act as a 4-bit input to the decoder and output the result across all of the displays, HEX0 - HEX7.



c3c2c1c0	character	c3c2c1c0	character
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Figure 13: 7 segment display and decoder

Tips:

- You should create two entities, one labled part2 that contains the logic for the switch input and output to the displays and another labled bcd7seg that acts as a decoder for the display.
- The eight 7 segment displays can be accessed with the 7-bit signal vectors HEX0…HEX7. For example, to output to the first display (HEX0) you can either set each bit individually (HEX0(5) <= '0';) or set the whole vector with (HEX0 <= '00000000') which would display the number 8. Keep in mind that the LED segments use inverted logic.
- Figure 14 shows how to correctly display all the required characters.

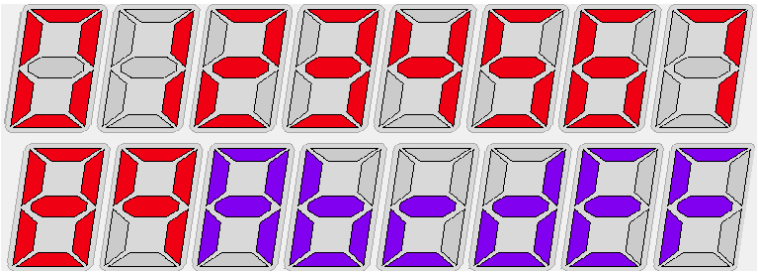


Figure 14: 7 segment displays showing all combinations for 0-9 and A-F

1.6 Lab Report

Your lab report should be upload to Sakai in a zip folder that includes; your commented VHDL code, your VHDL test bench, a pdf of your waveforms, and a text file answering any questions from the activities.

2 Lab 2 - Latches, Flip-Flops, and Counters

2.1 Introduction

Elementary latches and flip-flops have been used for years as a means to store temporary data either from the outputs of logic operations or by setting them to configure logic to behave in certain ways. This lab will go into the aspects of creating latches and flip-flops which will then be used to create a counter.

2.2 Pre-lab

Before you begin this lab you should complete the following and upload to Sakai:

- Write down the truth table for a D-latch, SR-latch and J-K flip-flop
- Design a block diagram for an 8-bit counter using J-K flip-flops

2.3 Lab Activities

2.3.1 Latches

The following VHDL code implements the logic for a D-latch based off of the schematic in figure 15.

```
-- A gated D latch
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dlatch IS
     PORT (     Clk, D     : IN STD_LOGIC;
                Q, Qbar    : OUT      STD_LOGIC);
END dlatch;

ARCHITECTURE rtl OF dlatch IS

     SIGNAL D1, D2, Qa, Qb : STD_LOGIC; -- Intermediate signals
     ATTRIBUTE keep: boolean; -- For waveform results
     ATTRIBUTE keep of D1, D2, Qa, Qb : signal is true;

BEGIN

     D1 <= NOT (D AND CLK);
     D2 <= NOT (D1 AND CLK);
     Qa <= NOT (D1 AND Qb);
     Qb <= NOT (D2 AND Qa);

     Q <= Qa;
     Qbar <=Qb;

END rtl;
```

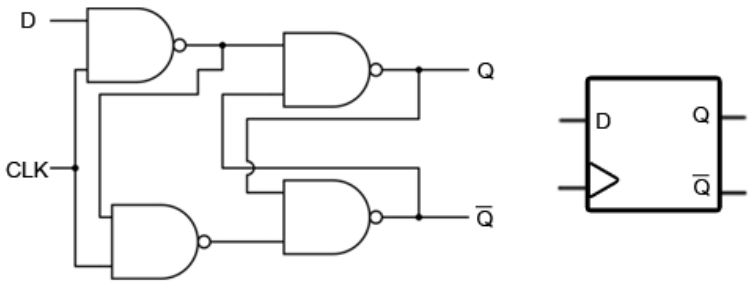


Figure 15: D-latch circuit and block diagram

Using this as a reference, design VHDL for an SR-latch with a clock input. Verify with waveforms that the circuit behaves the same as the truth table you created in the pre-lab.

2.3.2 Flip-Flop

Design VHDL code that implements the logic for a J-K flip-flop from figure 16. Verify with waveforms that the circuit behaves the same as the truth table you created in the pre-lab

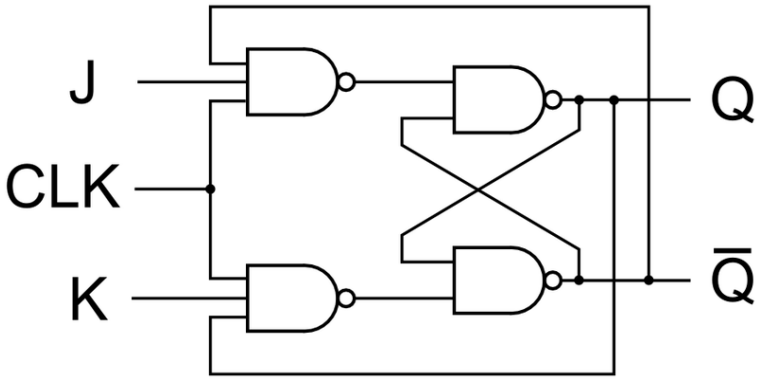


Figure 16: J-K fip-flop circuit

2.3.3 Counters

In the pre-lab, you created a block diagram for an 8-bit counter using J-K flip flops. Using the same VHDL code you created for implementing the J-K flip-flop, implement an 8-bit counter that increments when you press KEY0 on the DE2-115 development board. Link the binary output of the flip-flops to the red LEDs and then convert the binary value into hexadecimal to be shown on the 7-segment displays. *Hint: you should refer to your code for the 7-segment display driver designed in the previous lab*

2.4 Lab Report

After completing the activities in this lab you should create a zip folder with the following and then submit it to Sakai:

- Commented VHDL code
- VHDL test benches for all activities

- Waveforms for all activities
- A discussion on the results of compilation including longest path delay, the total number of logic elements used, and issues you encountered while performing the lab

3 Lab 3 - Complex Addition Systems

3.1 Introduction

From your pocket calculator to inside modern CPUs adders have long been used as more than just a simple way to sum numbers together. This lab will go into the logic structure of the adder as well as provide a method for converting the simple full adder in to an *arithmetic logic unit* (ALU) capable of handling 10 operations.

3.2 Pre-lab

Before coming to the lab, please complete the following and upload to Sakai:

- A truth table for a half adder and a full adder
- The logic equation for a 4-bit ripple carry adder

3.3 Lab Activities

3.3.1 Half Adder

Build the circuit in Figure 17, create a test bench and verify that the logic is correct.

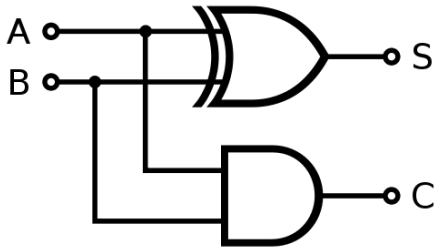


Figure 17: Circuit for a 1-bit half adder

3.3.2 Full Adder

The circuit in Figure 18 implements a full 1-bit adder. Implement this circuit in VHDL, create a test bench, and verify that the logic behaves as expected.

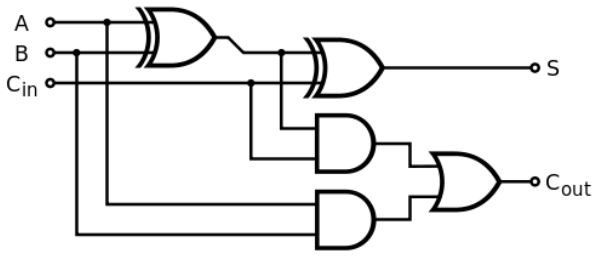


Figure 18: Circuit for a 1-bit full adder

Now that you have a working 1-bit full adder, implement a 4-bit ripple carry adder that sums the binary numbers "0110" and "0101." A block diagram for the 4-bit ripple carry adder is shown in Figure 19. Verify your results by creating a test bench and simulating the circuit.

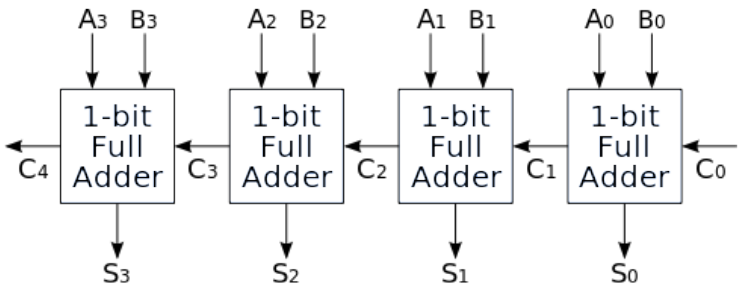


Figure 19: 4-bit ripple carry adder block diagram

3.3.3 Full Adder Based ALU

The block diagram in Figure 20 is an example of how the regular 1-bit full adder can be manipulated to implement additional functionality. For this activity, you must build VHDL code that implements a 4-bit complex adder ALU, the list of instructions can be found in Table 1.

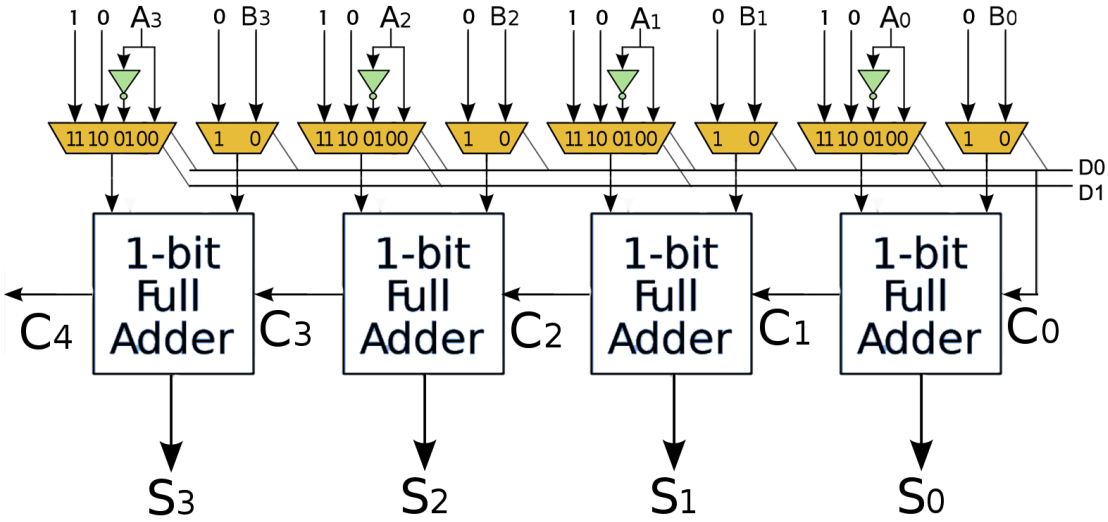


Figure 20: Block diagram for a 4-bit ripple carry adder alu with 10 operations

This activity will be performed on the DE2-115 FPGA development board, you should set **A** by connecting it to SW0 - SW3 while **B** should be set from SW4 - SW7. D_0 should be set with SW8 and D_1 with SW9. Display **A** in hexadecimal on HEX0, **B** on HEX2, and the result **S** on HEX4. If the result overflows display $\mathbf{C}_4$ on LEDG0. If the result is negative turn on LEDR0 and if it is zero turn on LEDR1.

$\mathbf{A}_i$	$\mathbf{B}_i$	$\mathbf{D}_0$ $\mathbf{D}_1$	**Result**
Set to 0	Set to 0	00	0
Set to 0	Set to 0	10	1
A	Set to 0	00	A
Set to 0	B	00	B
A	Set to 0	10	A + 1
Set to 0	B	10	B + 1
A	B	00	A + B
A	B	10	A − B
Set to invert	Set to 0	01	$\overline{\text{A}}$
Set to invert	Set to 0	11	−A

Table 1: List of opperations for the adder based ALU

When you finsih building the VHDL upload your code to the FPGA board. Test and verify all ten operations and take a picture of each result (make a table with that includes the values for **A, B, S, C_4, Neg, and Zero**.

3.4 Lab Report

After completing the activities in this lab you should create a zip folder with the following and then submit it to Sakai:

- Commented VHDL code
- VHDL test benches for the half adder and full adder activities
- Waveforms for the half adder and full adder activities
- Pictures of the results from the ALU activity
- A discussion on the results of compilation including longest path delay, the total number of logic elements used, and issues you encountered while performing the lab

4 Lab 4 - Finite State Machines

4.1 Introduction

4.2 Lab Activities

4.2.1 FSM

4.2.2

5 Lab 5 - A Simple Computer

5.1 Introduction

In this lab you will create a simple computer that can compute 7 instructions.

5.2 Lab Activities

5.2.1 Design an ALU

Add, Sub, Mult, Shift, XOR, AND, NAND

5.2.2 Design a RAM

512 bit memory

5.2.3 Design the Program Counter

simple counter

5.2.4 Create a VGA Driver

Display the output onto the screen

6 Final Project

6.1 Introduction

For the remainder of this course you will be required to work on and complete a project that

6.2 Requirements

6.3 Project Ideas

6.4 Deliverables