ECE 368: Digital Design

Lab 1: Introduction to Vivado

I certify that this work is original and not a product of anyone's work but my own.

Daniel J. Lopes: _____

Submitted: January 28th, 2022

Due by: February 2nd, 2022

Graded by: _____ Date: _____

Contents

List of Figures

List of Tables

List of Files

Abstract

Becoming familiar with software is crucial to the development of a students ability to understanding the system they will be working on. This lab would be the first in which students were exposed to the Vivado program and the NEXYS A7 FPGA board inside of this course and would act as a precursor to the next labs for the remainder of the semester. During the course of the laboratory, students will learn how to use Vivado and will apply their knowledge of VHDL commands to implement a one bit full adder. Students will utilize three buttons and two LED's to demonstrate the inputs and outputs of their full adder to validate correct operation.

1 Introduction

This laboratory consisted of three parts in which, the Vivado program and NEXYS A7 FPGA board was used for the first time by the students. For this lab, students were tasked with the development of VHDL code that would implement a one bit full adder using three switches and two LED's. The three switches will correspond to the three inputs of a one bit full adder, those being the two bits to add (namely "A" and "B") and the carry in bit labeled "cin". The LED's will correspond to the two outputs which are the output bit "S" as well as the carry out bit "C".

2 Methods and Procedures

2.1 Unit/System Under Test

To fill the requirements given, Vivado was the only program needed for this lab. In terms of hardware, the Nexys A7 FPGA Board was required in order to compile the code. The steps followed in this lab were detailed in the Lab 1 handout which can be found in the ECE 368 mycourses page as well as in the References section of this report. Summarizing, a one bit full adder was to be implemented using VHDL code, an FPGA, three dip switches, and two LED's. A one bit full adder is a combinational logic circuit where two binary bits, along with a carry in bit, are summed to produce an output as well as a carry out bit. Below in Figure 1, a truth table of the one bit full adder can be seen.

2.2 Test Procedure

Before generating the bit stream and programming the FPGA, a testbench should be used to simulate the VHDL code developed. For the purposes of this laboratory, the tb_my_full_adder test

bench is provided and found in the ECE 368 mycourses page. For reference to this test bench, see the References section of this report. By incorporating this test bench into the project and running a simulation, logic waveforms can be produced and tested against the one bit full adder truth table found in Figure 1. This will ensue confidence that the code will be successfully implemented with little to no errors.

To test the functionality of the one bit full adder implementation onto the FPGA, the truth table found in Figure 1 must be validated. In other words, using the three switches assigned to the inputs "A", "B", and "C_in" and the two LED's as outputs, "S" and "C_out", one can set the switches either high or low corresponding to one row of the table and then analyze the output LED's for the correct lighting sequence. In this sequence, an LED being lit would indicate a logical "1". This process can be repeated for all rows of Figure 1 to confirm the full adders operation.

Inputs			Outputs	
A	B	C_in	S	C_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 1: One Bit Full Adder Truth Table

3 Laboratory Experimental Results

The results of this lab came in the form of completed VHDL code where both the logic waveforms as well as the physical dip switch and LED test procedure matched exactly with the theoretical outputs shown in Figure 1. Shown below is the logic waveform captured from the Vivado simulation. Accompanying, Figure 3 shows one row of the truth table being tested using the programmed FPGA board. For a demo video see the embedded YouTube link below. All VHDL code developed will be attached in Section 9: Appendices.

`https://youtu.be/dxinwmaSvUY`

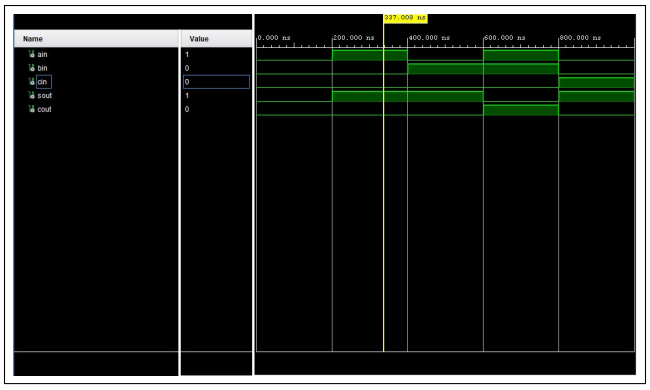

Figure 2: Logic Waveforms

From Figure 2, the test bench simulated four interesting input scenarios for the developed VHDL code. The first of the test cases was when the inputs A, B, and C_in were “1”, “0”, and “0” respectively. For this instance, the objects S and C_out took on the values “1” and “0” which corresponds exactly with the 5th row of Figure 1. Following this procedure for the next 3 test cases will as well yield identical results to that shown in the truth table.

As can be seen in Figure 3, when all switches are closed indicating a 1 for all inputs, both LED’s are lit indicating that the sum and carry out bits are a logical high. Referring back to the truth table in Figure 1, the last row indicates that when all input bits are high, the sum and carry out bit are also high meaning that Figure 3 correctly demonstrates the functions of a one bit full adder. Though this is one state, all other states are demonstrated and validated in the above YouTube video.

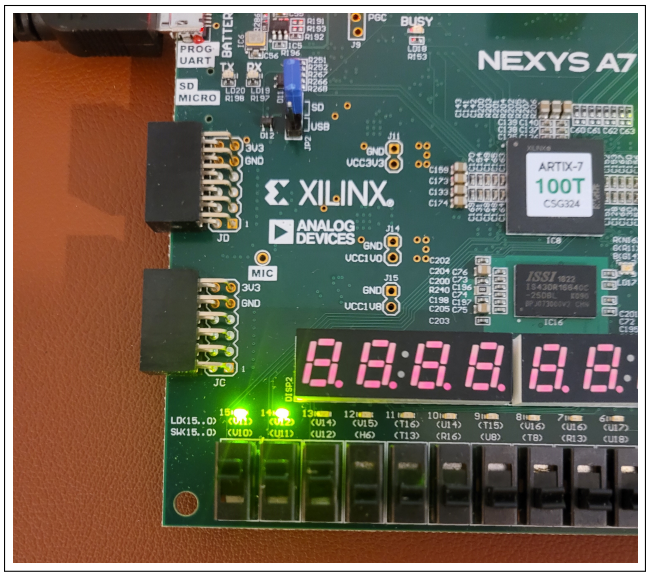

Figure 3: One Bit Full Adder ’111’ Sum

4 Discussion

Perhaps the most important take away from this laboratory experiment is the experience gained within Vivado as this is the first of many assignments that will be done within this course. It is here that we are able to apply the concepts of the lectures into more grounded and concrete examples which will inevitably help in the understanding of the language. Aside from this, in terms of the actual code, the assignment was fairly simple as it only called for implementing a one bit full adder onto an FPGA. The way that the VHDL code was structured required the use of three coding blocks, those being "LIBRARIES", "ENTITIES", and "ARCHITECTURE". For this laboratory, only the STD_LOGIC_1164 package located within the IEEE library was utilized which allowed for the use of the STD_LOGIC data type.Within the "ENTITIES" block, seen in Appendices Figure 5, only 5 signals, 3 inputs and 2 outputs, were declared all as an STD_LOGIC data type. Finally, the "ARCHITECTURE" block followed a dataflow structure rather than a behavioral or structural format since all that is within the block was transferred from the example code given in the Lab handout which can be found in the ECE 368 mycourses page as well as in the References section of this report.

To set the switches and LED's to certain inputs and outputs of the full adder design, the NEXYS A7 constraints file was found on GitHub and imported into the project (see References). Within the file, certain lines corresponding to the switches and LED's used were uncommented and tied to specific inputs. Below in the Appendix is the constraints file used within this laboratory.

Applications behind this laboratory stand out greatly from systems that require robust FPGA implementations such as underwater Bathymetry systems to Aviation flight electronics. System like this can include devices as big as Submarines to something as small as a network line card.

5 Conclusion

In summary, through the completion of this lab, students were exposed to the world of Vivado and were able to refine their understanding of the concepts taught in the lectures of this course. The ability to design logic circuits and implement them into the VHDL hardware language was tested as students had to use their understanding on the VHDL language to accomplish tasks such as including libraries and packages, declaring objects and object types, writing architecture designs, using test benches and simulation, including and modifying constraint files, and programming the FPGA. While the contents of the lab were nothing extreme, the groundwork it laid out was pivotal as more complex applications of the NEXYS A7 is explored throughout the remainder of this

course.

6 Recommendations

When doing these labs, it is often important to stick to the lab handouts and fully understand the task at hand. Use any template code to your advantage and try not to stay too off course from the lab assignment. Also be very careful to not accidentally write VHDL code in a verilog file as this will ensure failure.

7 Laboratory reflection

When attempting to complete this lab, the first attempt failed and cost a small amount of time to be lost. This was simply due to accidentally creating a verilog file and writing VHDL code within it. Also, in the future one should take care to review the waveforms generated after the simulation for accuracy before preceding as this will help ensure that the design functions as intended and bugs in the code are not present.

8 References

GitHub Containing NEXYS A7 Constraints File:

`https://github.com/Digilent/Nexys-A7-100T-Keyboard/blob/master/src/constraints/Nexys-A7-100T-Master.xdc`

ECE368 Digital Design
Lab 1 Introduction to Vivado for Simulation and Synthesis

Due on: 2/2/2022 @ 9am in myCourses

Objective: This is a one-week lab designed to be familiar with Vivado simulation and synthesis tool.

Requirements:

- **Part 1:** Write a VHDL code for 1-bit full adder with ENTITY's name `my_full_adder`; `A, B, Cin` as inputs; `C, S` as outputs, all in type of `STD_LOGIC`. You may reference the example given in lecture slides, shown in Figure 1.

```
ENTITY full_adder IS
PORT (a, b, cin: IN BIT;
      s, cout: OUT BIT);
END full_adder;
---------------------------------------
ARCHITECTURE dataflow OF full_adder IS
BEGIN
   s <= a XOR b XOR cin;
   cout <= (a AND b) OR (a AND cin) OR
           (b AND cin);
END dataflow;
```

Figure 1. An incomplete code example of a 1-bit full adder

- **Part 2:** Use the given testbench, `tb_my_full_adder.vhd` to simulate your 1-bit full adder. Check the logic of your design.

- **Part 3:** Program your FPGA, use switches as the `A, B, Cin` inputs, and LEDs to indicate the `C, S` outputs. Demonstrate to the TA.

Report:

Follow the report template to write the report using Latex. Name your file: ECE368_Lab1 _FirstName_LastName.pdf. Attach your .vhd files as appendix of your report. Upload the pdf file to myCourses.

Grading:

- VHDL Code: 20%
- Simulation: 20%
- Demo on FPGA: 30%
- Report: 30%

Figure 4: Lab Handout

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY tb_my_full_adder IS
5  END tb_my_full_adder;
6
7  ARCHITECTURE my_arch OF tb_my_full_adder IS
8
9  COMPONENT my_full_adder
10   PORT(a, b, cin: IN STD_LOGIC; S, C: OUT STD_LOGIC);
11 END COMPONENT;
12
13 signal ain: std_logic := '0';
14 signal bin: std_logic := '0';
15 signal cin: std_logic := '0';
16 signal sout: std_logic;
17 signal cout: std_logic;
18
19 BEGIN
20   U1: my_full_adder port map (a => ain, b => bin, cin => cin, s => sout, c => cout);
21
22   process
23     begin
24       ain <= '0';
25       wait for 200 ns;
26       ain <= '1';
27       wait for 200 ns;
28   end process;
29
30   process
31     begin
32       bin <= '0';
33       wait for 400 ns;
34       bin <= '1';
35       wait for 400 ns;
36   end process;
37
38   process
39     begin
40       cin <= '0';
41       wait for 800 ns;
42       cin <= '1';
43       wait for 800 ns;
44   end process;
45
46 END my_arch;
```

Figure 5: One Bit Full Adder Test Bench Code

9 Appendices

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  ----------------------------------------------
4  ENTITY my_full_adder IS
5  PORT (a, b,cin: IN STD_LOGIC;
6          S, C: OUT STD_LOGIC);
7  END my_full_adder;
8  ----------------------------------------------
9  ARCHITECTURE dataflow OF my_full_adder IS
10 BEGIN
11     S <= a XOR b XOR cin;
12     C <= (a AND b) OR (a AND cin) OR
13          (B and cin);
14 END dataflow;
```

Figure 6: VHDL Code For One Bit Full Adder

```
## This file is a general .xdc for the Nexys A7
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];


##Switches

#set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
#set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
#set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
#set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
#set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
#set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { SW[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { SW[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { SW[8] }]; #IO_L24N_T3_34 Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { cin }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { b }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { a }]; #IO_L21P_T3_DQS_14 Sch=sw[15]

```

Figure 7: Constraints File Switches Declarations

```
## LEDs

#set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LED[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
#set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { LED[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
#set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { LED[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
#set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { LED[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
#set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { LED[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { LED[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
#set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { LED[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { LED[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { LED[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { LED[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
#set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { LED[13] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { C }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { S }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

#set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports { LED16_B }]; #IO_L5P_T0_D06_14 Sch=led16_b
#set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports { LED16_G }]; #IO_L10P_T1_D14_14 Sch=led16_g
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { LED16_R }]; #IO_L11P_T1_SRCC_14 Sch=led16_r
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { LED17_B }]; #IO_L15N_T2_DQS_ADV_B_15 Sch=led17_b
#set_property -dict { PACKAGE_PIN R11   IOSTANDARD LVCMOS33 } [get_ports { LED17_G }]; #IO_0_14 Sch=led17_g
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { LED17_R }]; #IO_L11N_T1_SRCC_14 Sch=led17_r

```

Figure 8: Constraints File LED Declarations