

# ECE-399: Selected Topics in ECE: Design with FPGAs

## Report 1: Familiarization with FPGAs and Zedboard

Prof. Shlayan  
Nithilam Subbaian

May 15, 2020

### 1 Objective of Report 1:

For this report, the first goal is to familiarize myself with FPGAs in general, learning about their history and relevant background information and the second goal is to learn to use Zedboard with basic environment setup and programming. To complete these goals I will go through lecture presentations 1-4 in ECE311 Hardware Design from Moodle, and complete the tasks outline in the lectures.

### 2 Materials

1. Zedboard [4]
2. 4GB SD card (purchased independently of Zedboard Kit)
3. 12V Power supply
4. Micro USB cable
5. USB Adapter: Male Micro-B to Female Standard-A
6. Xilinx Vivado Design Suite Version 2018.3 [13]

### 3 FPGA History and Relevant Background

The information in this section is my notes taken from lecture [9] and [11] among other resources.

#### 3.1 ASIC vs FPGA

An Integrated Circuit (IC) is a set of electronic circuits on a chip of semiconductor material. IC chips allow for circuits that are significantly smaller, faster, and cheaper than circuits made with discrete electronic components.

Traditionally, an Application-Specific Integrated Circuit (ASIC) is an integrated circuit chip made for a specific use case, and uses hardware description language (HDL) such as verilog or VHDL. Examples of an ASIC chip include a chip to be used in satellites or a chip to be used in a Bitcoin Miner [6]. The sizes of ASICs have dramatically grown over the years, from 5,000 logic gates, to over 100 million, with possibly including microprocessors, memory blocks, and more [5].

A Field-Programmable Gate Array (FPGA) is an integrated circuit that is made to allow an user to design or configure it's function using HDL. FPGAs are the modern equivalent for making a breadboard or prototype. FPGAs consist of programmable logic blocks and programmable interconnects, allowing the same FPGA to be used for different applications.

The non-recurring engineering (NRE) cost of an ASIC can be really high and therefore, for smaller designs or lower production volumes, FPGAs are more cost effective. For very high production volumes, the NRE costs can be amortized over all the products, and ASICs may be more cost effective.

## 3.2 Further look into FPGAs

Programmable logic devices (PLDs) are growing rapidly in industry amassing around \$10 billion dollars a year by 2020. PLDs are used in autonomous vehicles, the internet of things, machine vision and learning, facial recognition, smart medical diagnostics, secure data centers and cloud computing.

FPGAs are a type of PLD and therefore do not have a fixed function when it is manufactured, allowing the user to reconfigure the circuit for almost any digital function. FPGAs have as many as 20 million gates. FPGAs have an easy entry- barrier, having low costs for FPGA development compared to other alternatives. FPGAs are the preferred method for prototyping, in fact many ASICs are prototyped using FPGAs. FPGAs are also used by process manufacturers to verify proper function of System on Chips (SoCs). An SOC is an integrated circuit that includes all components of a computer or electronic system, specifically, at least a CPU, memory, input/output ports and secondary storage [3].

## 3.3 About the Zed board

Zed board description: “ZedBoard is a low-cost development board for the Xilinx Zynq™-7000 SoC” [1]. The Zedboard is equipped with the functionality and interfaces to allow for rapid prototyping and proof of concept development. Xilinx advertises that the “Zynq-7000 SoCs tightly coupled ARM® processing system and 7 series programmable logic,” can be used to create applications such as “video processing, motor control, software acceleration, Linux/Android/RTOS development, embedded ARM processing, and general Zynq-7000 SoC prototyping” [14].

## 4 Vivado installation

The information in this section is my notes taken from lecture [9] and [11] among other resources.

The Vivado Design Suite is made by Xilinx to work with HDL designs. It replaces the previous ISE that was used for 15 years, by offering more abilities to work with SoC development and high level synthesis.

Following the instructions from [11], I verified communication with the Zedboard, completed the installation of Vivado 2018.3 HL Webpack and programming the FPGA through JTAG. JTAG is a hardware interface that allows a computer to communicate directly to chips on a board and is used for debugging, programming and testing on most embedded devices [8].

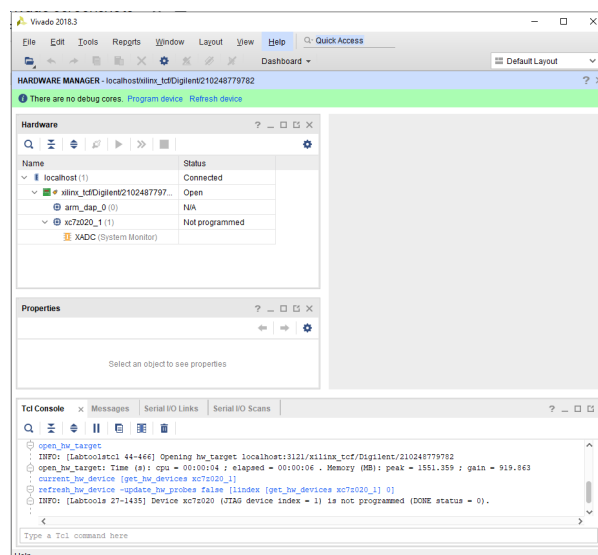


Figure 1: Programming a FPGA through JTAG: Connection is Successful

In this process, the Xilinx software development kit (SDK) was also downloaded. The Xilinx SDK is an integrated development environment that works with Vivado. It is used for creating C/C++ embedded software applications and built on the Eclipse open-source framework.

## 5 Design Flow and Example

The information in this section is my notes taken from lecture [10] among other resources.

There are various design flows such as RTL & IP, HLS - High Level Synthesis, Embedded Processor Design, and Partial Reconfiguration.

The design flow from the lectures notes starts with a problem statement containing product specifications and assignments, which leads to a Behavioral Description (RTL, State transition diagrams), which leads to HDL Description (Verilog, VHDL), and then finally, the Hardware Implementation (FPGA, ASICs).

In using the Vivado Design Suite, with the System Level Design Flow, there are a series of phases, leading to programming the FPGA. In the first phase, in System Design Entry are HDL Source/IP and Testbenches. Then in the next phase, Implementation, are Simulation, Assign Logical and Physical Constraints, Synthesis (High to low-level gate Netlist), Implementation (map netlist on FPGA) and Timing & Design Analysis. In the final phase, Hardware Bring-Up & Validation, the Bitsream is generated (describes re-configurations of the FPGA) and the FPGA is programmed.

As specified in Lecture 3, a new project was started, and a combinational block design was created using the utility vector logic IP and external ports as shown in figure 2. From this, HDL code was created by clicking "Create Design Wrapper". A constraint file was created (.xdc) to make input/output assignments and configure voltage levels. The constraint file can be seen in figure 3. Lastly, the bit stream was generated, the device was programmed, and tested on the Zedboard.

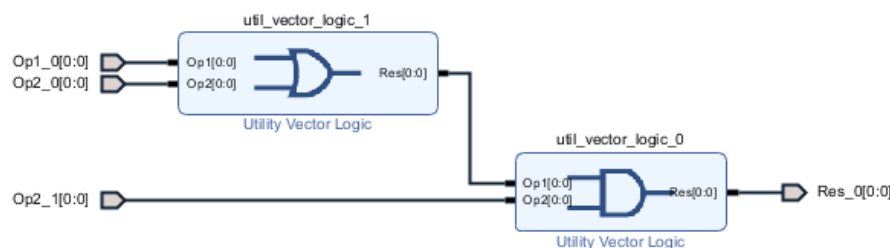


Figure 2: a nice plot

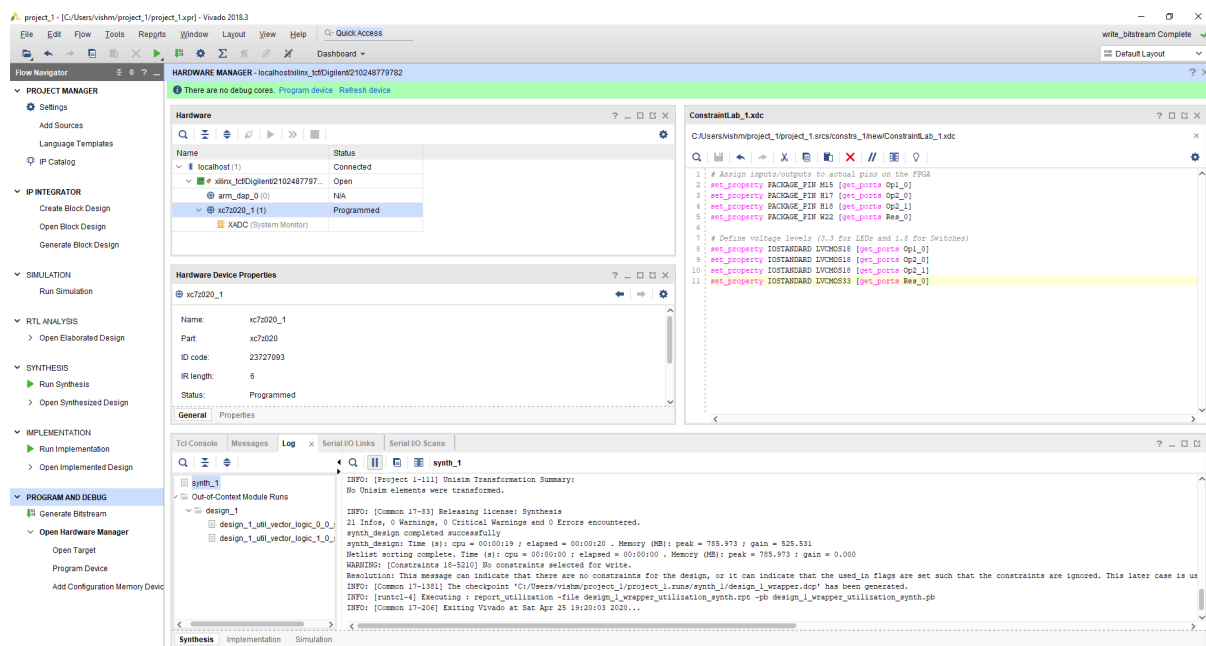


Figure 3: a nice plot

## 6 Introduction to Digital Logical Design: Combinational Logic

The information in this section is my notes taken from lecture [12] among other resources.

### 6.1 Introduction to Digital Design

There are levels of abstraction for an electronic computing system. Going from the highest level, to the lowest level, the levels are: Application Software: Programs, Operating Systems: Device Drivers, Architecture: Instructions/Registers, Micro-architecture: Datapaths/Controllers, Logic: Adders/Memories, Digital Circuits: AND Gates/NOT Gates, Analog Circuits: Amplifiers/Filters, Devices: Transistors/-Diodes, and Physics: Electrons.

In the middle levels are Digital Circuits and Logic Gates. Digital Circuits restrict voltages to discrete ranges to build more complex structures, such as, adders, multipliers memories, and more. Logic Gates are digital systems that perform operations on these binary system. NOT, BUF, AND, OR, XOR, NAN, NOR, and XNOR, are different logic gates, whose relationship between the inputs and the outputs can be described with a truth table or Boolean equation.

Beneath the abstraction, are logic levels and noise margins. Logic Levels are the mapping of a continuous variable onto a discrete binary variable and the noise margin is the amount of noise that could be added to a worst-case output such that the signal can still be interpreted as a valid input. From the output of the driver to the input of the receiver, it is important that the voltage of a logic level from the output transfers to the input within the noise margin from the output to the input to ensure that the logic level is not read incorrectly by the receiver. To calculate the Noise margin:  $NM_L = V_{IL} - V_{OL}$  and  $NM_H = V_{OH} - V_{IH}$  where I is for input, O is for output, L is for low, and H is for high.

In defining the logic levels, choose logic levels at the unity gain points to maximize the noise margins. This is where, the slope of the transfer characteristic  $cV(Y)/dV(A) = -1$ . To avoid inputs falling into the forbidden zone, digital logic gates are designed to conform to the static discipline, which means that given logically valid inputs, every circuit element will produce logically valid outputs. There is a trade off associated with the static discipline as simplicity and robustness come at the expense of freedom of using arbitrary analog circuit elements. All gates that communicate must have compatible logic levels. Gates are grouped into four major logic families, TTL, CMOS, LVCMOS, LVTTL, that all have different acceptable ranges of voltages for  $V_{DD}$ ,  $V_{IL}$ ,  $V_{OL}$ , and  $V_{OH}$ .

### 6.2 Combinational Logic Design

A digital circuit is a network that processes discrete valued variables with one or more input terminals, one or more output terminals, a functional specification describing the relationship between inputs and outputs, and a timing specification describing the delay between inputs changing and outputs responding. For this class, an "element" is a circuit with inputs, outputs, and a specification ( functional and timing). A "node" is a wire, whose voltage conveys a discrete-valued variable and are classified as input, output, or internal.

Combinational circuits are memoryless, outputs depend only on the current values of the inputs, ex. logic gates. They have a function specification and timing specification. On the other hand, sequential circuits have memory, outputs depend on both current and previous values of the inputs or input sequence, ex. registers.

A circuit is combinational if every element is itself combinational, every node of the circuit is either designated as an input to the circuit or connects to exactly one output terminal of a circuit element, and the circuit contains no cyclic paths: every path through the circuit visits each circuit node at most once.

In order to make the function specification, a truth table or Boolean algebra and Karnaugh maps can be turned into a Boolean equation, which is implemented with logic gates, and lastly, analyzed for speed. General Schematic guidelines specify that a dot where wires cross indicates a connection between the wires, and wires crossing without a dot make no connection.

In a Programmable Logic Array (PLA), Sum-of-Products Form, the inverters, AND gates, and OR gates are arrayed in a systematic fashion. Depending on the implementation, it may be cheaper to use the fewest gates or to use certain types of gates in preference to others. For example, NANDs and NORs are preferred over ANDs and ORs in CMOS implementations.

Multilevel combinational circuits may use less hardware than their two-level counterparts. An example being a 3-input XOR. This 3 input XOR can also be built out of a cascade of two-input XORs. There are trade offs associated with selecting the best multilevel implementation of a specific logic function such as fewest gates, speed, design time, cost, and power consumption.

### 6.3 More Terminology

Bubble Pushing - helpful way to redraw circuits so that the bubbles cancel out and the function can be more easily determined.

Illegal Value X - occurs when a node is being driven to both 0 and 1 at the same time (a contention). This is indicated by a symbol "X", and voltage on a node like this is often in the forbidden zone. This can cause large amounts of power to flow between the fighting gates, resulting in the circuit getting hot and possibly damaged.

Floating Value Z - is when a node is being driven neither HIGH or LOW. The node might be 0, might be 1, or a voltage in between depending on the history of the system. One way this could be produced is if one forgets to connect a voltage to a circuit input.

### 6.4 Combinational Building Blocks

**Multiplexers** choose an output from among several possible inputs based on the value of a select (control) signal. Two ways to build multiplexers are by using two-level logic or tristate buffers, among other ways. A wider multiplexer is a N:1 multiplexer with  $\log_2 N$  select lines.

Multiplexers can be used as lookup tables to perform logic functions. For example, a 4:1 multiplexer implementation can be designed to mimic the function of two-input AND function. The two inputs, A and B, serve as select lines. Then, in the MUX, the inputs (00, 01, 10, 11) are connected to 1 or 0 based on the AND truth table. Essentially, any  $2^N$  input multiplexer can be programmed to perform any N-input logic function by applying 0's and 1's to the appropriate data inputs. Additionally, using only a  $2^{N-1}$  input multiplexer we can perform any N-input logic function cutting the multiplexer's size in half. For example traditionally, in an AND gate the truth table reads: input 00 outputs 0, input 01 outputs 0, input 10 outputs 0 and input 11 outputs 1, where the first number in the input corresponds to input A and second number in the input corresponds to input B. This can be reduced to a 2-1 multiplexer in which A is a select line, and if A is 0, the output is 0, and if A is 1, the output is the value of B.

Bellow in figure 4 is an example of a reduction from an 8:1 multiplexer to a 4:1 multiplexer from lecture [12] that I believe well demonstrates the concepts.

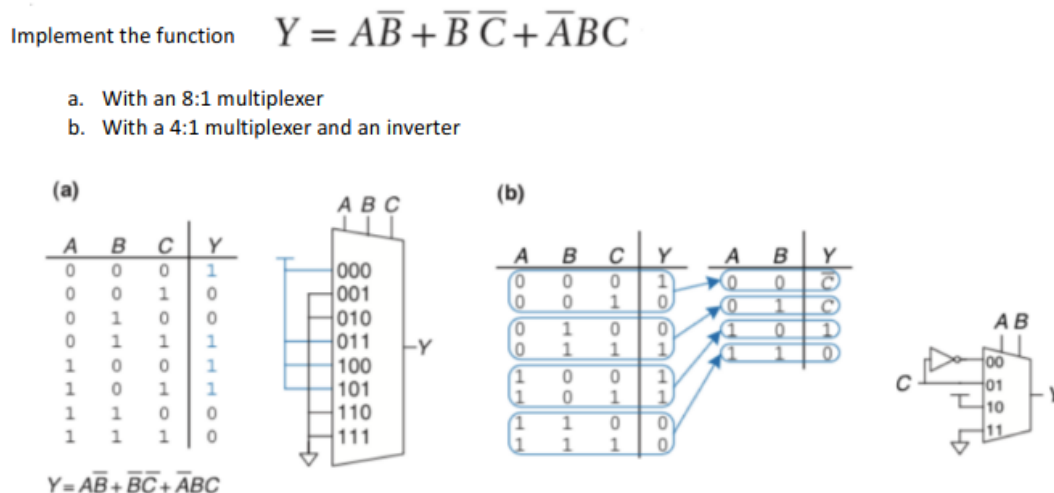


Figure 4: a nice plot

A **decoder** has N inputs and  $2^N$  outputs. It asserts exactly one of its outputs depending on the input combination.

For example, here is an implementation of 2:4 decoder with AND, OR, and NOT gates from [12] in figure 5. An  $N:2^N$  decoder can be constructed from  $2^N$  N-input AND gates that accept the various combinations of true or complementary inputs. Each output in a decoder represents a single minterm. For example Y3 represents the minterm A1A0.

Minterm and Maxterm refer to how truth tables can be converted to Boolean equations [2].

To elaborate on minterms, a minterm is a Boolean expression resulting in 1 for the output of a single cell in a Karnaugh map, or truth table. As defined by resource [2], "Each minterm has value 1 for exactly

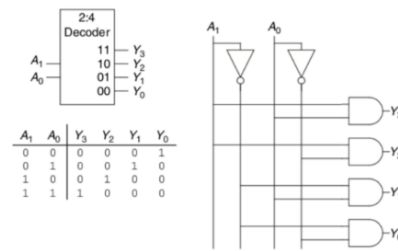


Figure 5: a nice plot

one combination of values of variables. A function can be written as a sum of minterms, which is referred to as a minterm expansion or a standard sum of products.”

A maxterm is a Boolean expression resulting in a 0 for the output of a single cell expression, and 1s for all other cells in the Karnaugh map, or truth table. As defined by resource [2], “Each maxterm has a value of 0 for exactly one combination of values of variables. A function can be written as a product of maxterms, which is referred to as a maxterm expansion or a standard product of sums.”

Decoders can be combined with OR gates to build logic functions. An example is seen in figure 6.

### Example- XNOR

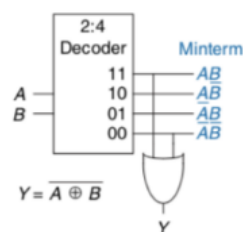


Figure 6: XNOR using a decoder

When using decoders to build logic, it is easiest to express functions as a truth table or in canonical sum-of-products form. In general, a  $N$ -input function with  $M$  1's in the truth table can be built with an  $N: 2^N$  decoder and an  $M$ -input OR gate attached to all of the minterms containing 1's in the truth table. This is used to build Read Only Memories (ROMs).

The ROM stores binary data which can be read from an input address. For every address there is corresponding data, as visualized in figure 7 from resource [7].

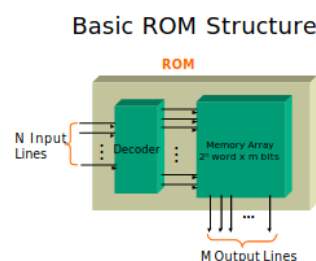
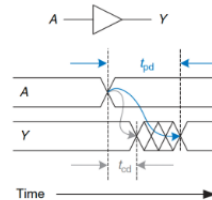


Figure 7: XNOR using a decoder

## 7 Timing

A delay is when an output takes time to change in response to an input change. Shown in figure 7 is a transient response of a buffer. Measured from the 50 percent point of the input signal, A, to 50 percent of the output signal Y. Note that a buffer, a digital buffer, is used to isolate the input from the output, if the input is 1 the output is 1, if the input is 0 the output is 0. A voltage buffer has a high input impedance, therefore the circuit will draw little current, not affecting the power source.



### 7.1 Propagation and Contamination Delay

Combinational logic is characterized by Propagation delay ( $t_{pd}$ ) and Contamination delay  $t_{cd}$ . Propagation delay is the maximum time from when an input changes until the output or outputs reach their final value. Contamination delay is the minimum time from when an input changes until any output starts to change its value.

Propagation and contamination delays are determined by the path a signal takes from input to output as seen in figure 10. The propagation delay of a combinational circuit is the sum of the propagation delays through each element on the critical path. The contamination delay is the sum of the contamination delays through each element on the short path.

### 7.2 Glitches

Single input transition can cause multiple output transitions.

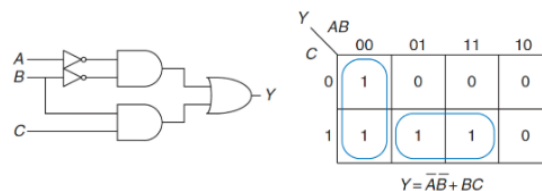


Figure 8: When  $A=0$ ,  $C=1$ , and  $B$  transitions from 1 to 0

### 7.3 Introduction to Verilog HDL

There are three kinds of modeling styles: Gate-level, Dataflow, and Behavioral. Combinatorial circuits are made with all three types whereas sequential circuits are made from Behavioral.

For this section, the goal are: to create scalar and wide combinatorial circuits using gate-level, dataflow, and behavioral modeling; Write models to read switches and output on LEDs; Simulate and understand the design output; Create hierarchical designs; Synthesize, implement and generate bitstreams; Download bitstreams into the board and verify functionality.

### 7.4 Gate-Level Modeling

Built-in primitive gates modeling are supported:

Multiple-input (one output)

and — nand — or — nor — xor — xnor [instance name] (out, in1, ..., inN)

Multiple-output (one input)

buf — not [instance name] (out1, out2, ..., outN, input);

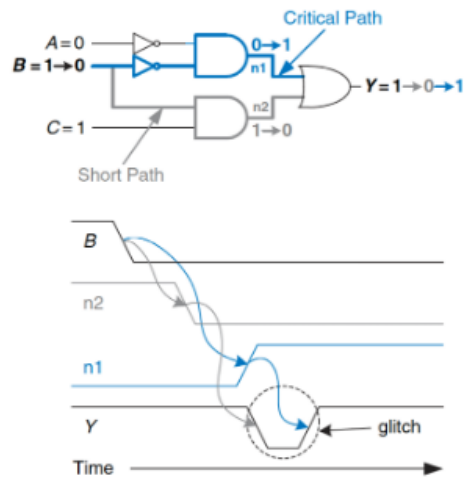


Figure 9: When B transitions from 1 to 0, n2 falls before n1 can rise. Until n1 rises, the two inputs to the OR gate are 0, and the output Y drops to 0. When n1 eventually rises, Y returns to 1.

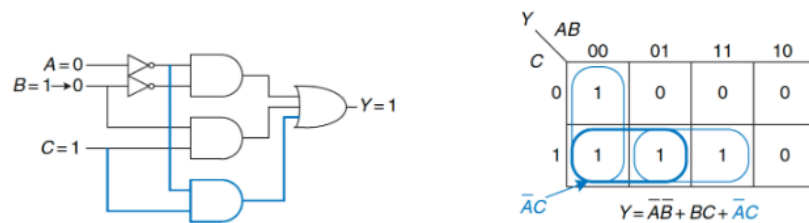


Figure 10: The Circuit Designed without a Glitch

Tristate (one input, one control signal, one output)

bufif0 — bufif1 — notif0 — notif1 [instance name] (outputA, inputB, controlC);

Pull gates (single output, no input)

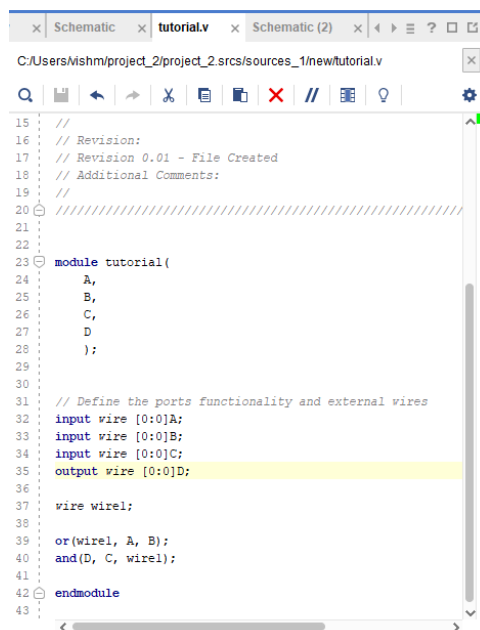
pullup — pulldown [instance name] (output A);

// [] is optional and — is selection



## 8 Vivado and Zedboard Work

To start of, a module was written based on the circuit created earlier in figure 3, as shown by the file called tutorial.v in figure 11. This verilog source file describes the combination function of the circuit in figure 3.



```

15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21
22
23 module tutorial(
24     A,
25     B,
26     C,
27     D
28 );
29
30
31 // Define the ports functionality and external wires
32 input wire [0:0]A;
33 input wire [0:0]B;
34 input wire [0:0]C;
35 output wire [0:0]D;
36
37 wire wire1;
38
39 or(wire1, A, B);
40 and(D, C, wire1);
41
42 endmodule
43
  
```

Figure 11: Module based on circuit created in figure 2

Shown in figure 12 is the RTL analysis on the source file. The model (design) is elaborated and a logic view of the design is displayed.

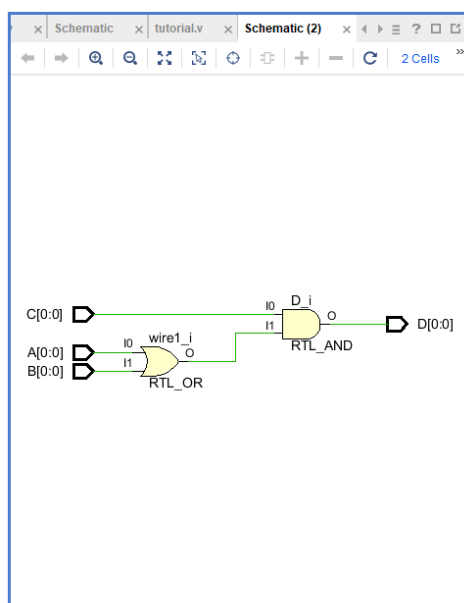


Figure 12: RTL analysis on the source file in figure 11

Shown in figure 13, is the package view after clicking on the I/O planning layout. In this view we can see the design ports in the I/O Ports tab with multiple I/O standards. The pin site number (along with the pin type and I/O bank it belongs to) is at the bottom of the Vivado screen and changes as the mouse is moved across the board.

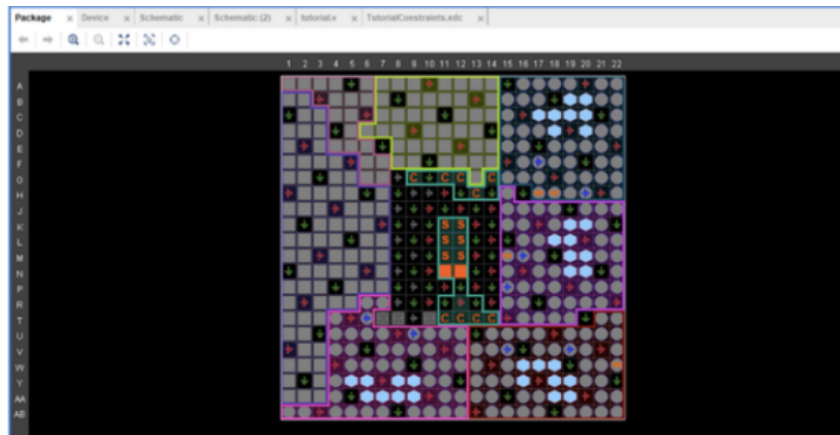


Figure 13: I/O planning Package View

At this point, the ports can be assigned to pins. There are multiple ways to do this such as using the "Package Pin" column, selecting an entry in the I/O ports tab or entering the pin constraints and I/O standards using tcl commands. Here is a sample TCL command:

```
1 set_property -dict { PACKAGE_PIN w22 IOSTANDARD LVCMOS33 } [get_portsD];
```

Next, the design is simulated using a testbench. The testbench used is seen in the second file in the Appendix. Shown in figure 14 is the testbench simulation.

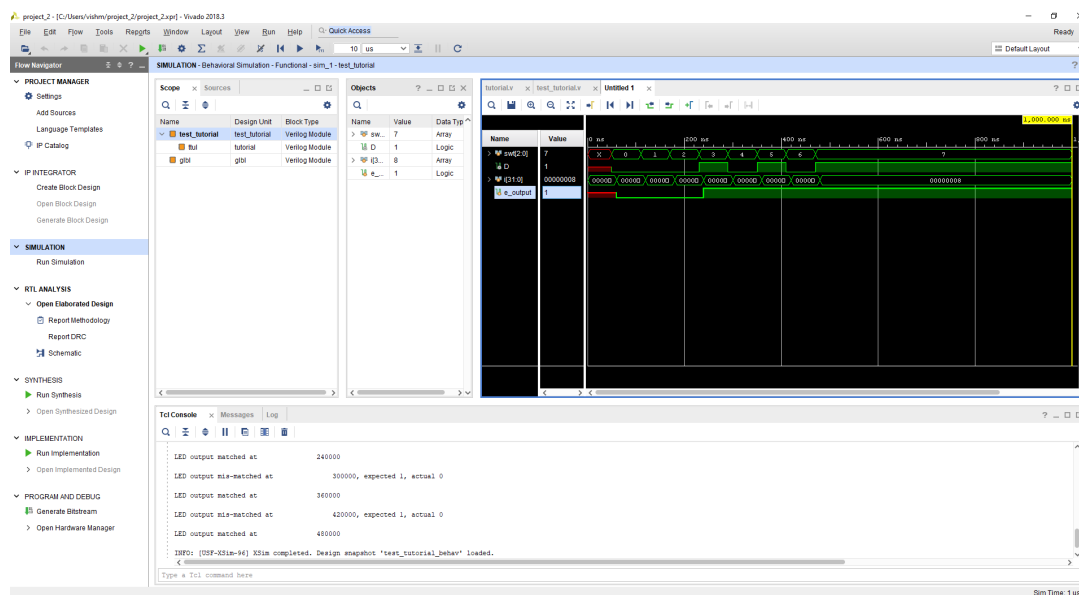


Figure 14: Simulation output from testbench

Then, the design was synthesized by clicking on "Run Synthesis". In the projects summary tab are the estimated LUTs and IOs that are used as shown in figure 15. The IBUF and OBUF are automatically added to the design as the input and output are buffered. The logical gates are implemented in LUTs as shown in figure 16. Shown in figure 17, is the schematic view.

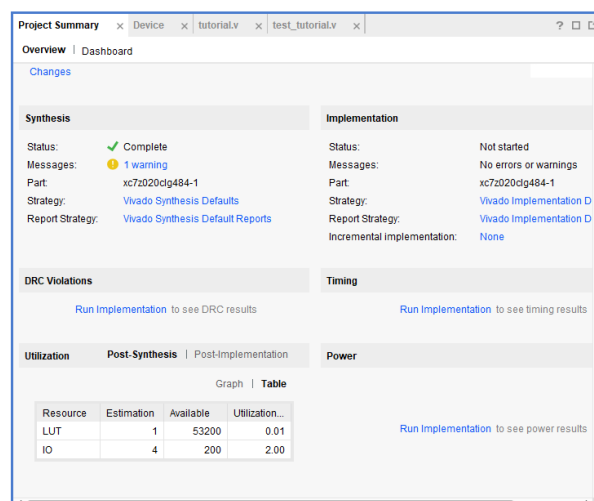


Figure 15: Project Summary Tab

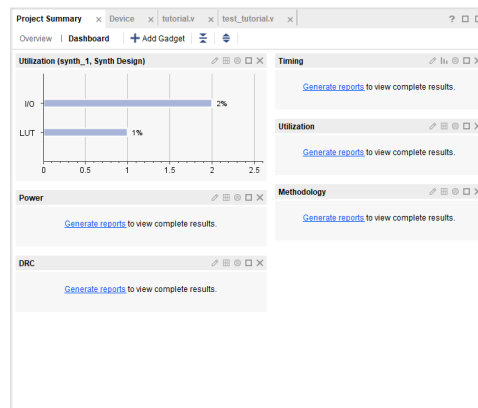


Figure 16: Dashboard view os LUTs and I/O

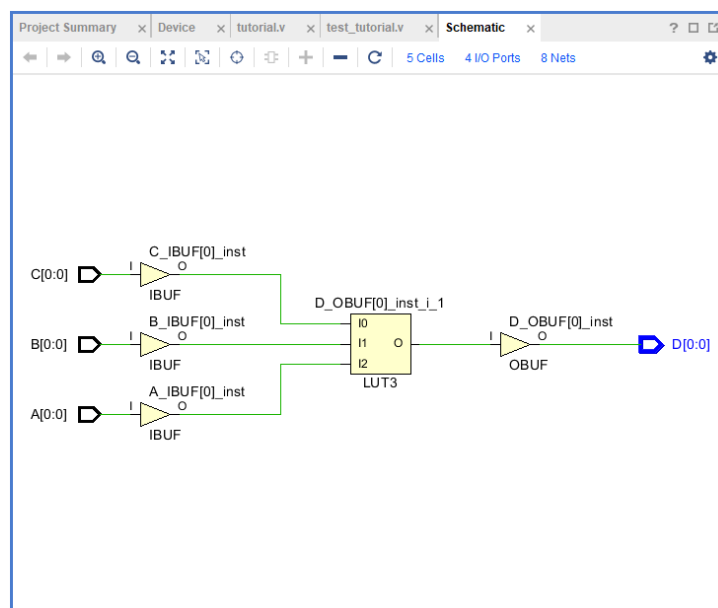


Figure 17: Schematic View

Next, the design is implemented by clicking on "Run Implementation". Shown in figure 18, is the Netlist pane with A\_IBUF(1) selected, and the net displayed in the X1Y1 clock region in the Device view tab. Shown in figure 19, is the utilization. As the design is combinatorial no registers are used.

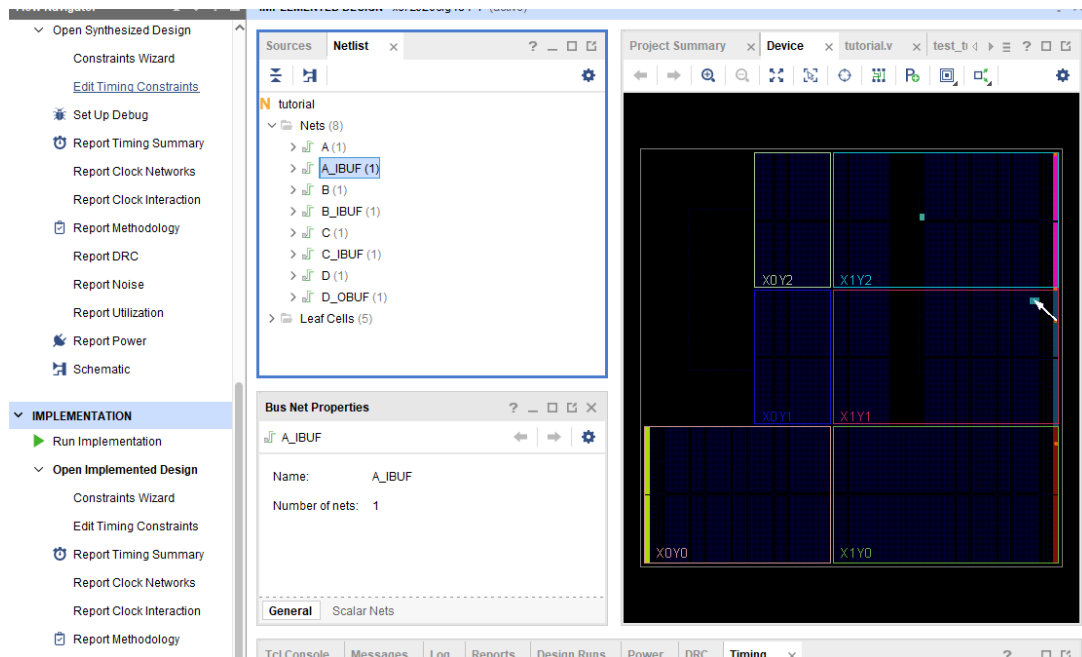


Figure 18: Net List Pane visualized

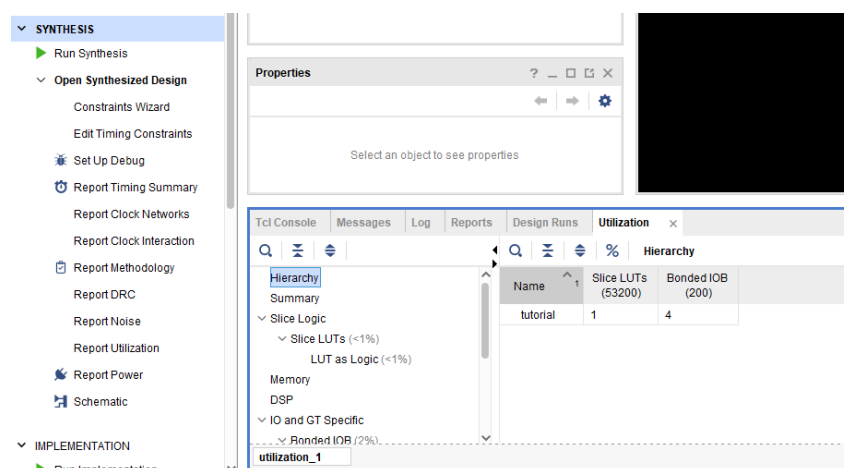


Figure 19: Schematic View 2

Lastly, the simulation was run, and “Run Post-Implementation Timing Simulation” was clicked on. As seen in figure 20, a marker was added where the switch input is set to 0000000b. Another marker was added where the output changes. The time between the two match the delay set in the testbench of 60ns.

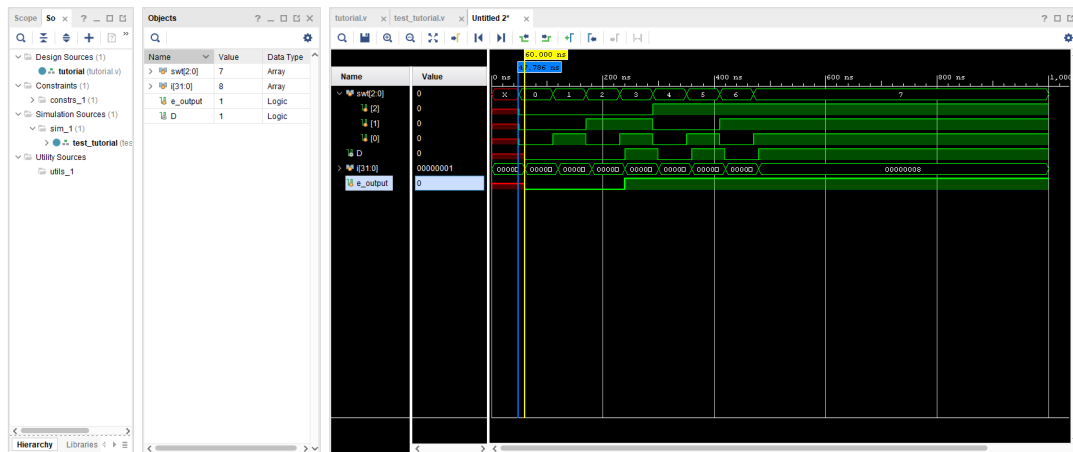


Figure 20: Schematic View 3

At the very end, the bitstream was generated and the function was verified as shown in figure 21.

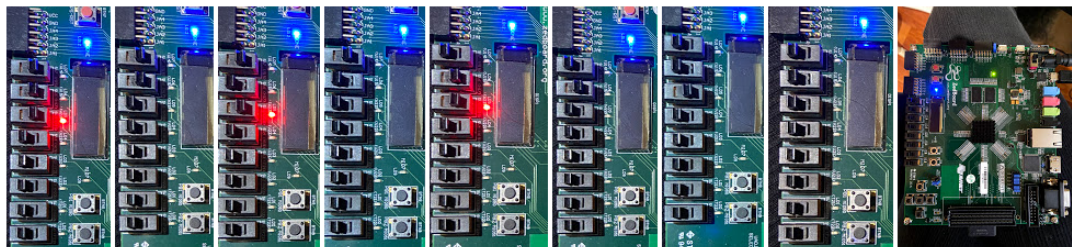


Figure 21: Verification of Bit stream functionality

## 9 Discussion of Difficulties

Due to the COVID-19, all students were no longer able to use the ECE labs, and as a results I took all materials to work on at home. It was too late when I realised that the SD card was missing, and then I got really worried I would not be able to work with the Zedboard. I soon realised I did not need the SD card and was able to complete the assignments!

## References

- [1] Avnet. *Zedboard English Brochure*. URL: <http://www.zedboard.org/sites/default/files/Avnet%5C%20ZedBoard%5C%20Brochure%5C%20English%5C%20Version.pdf>.
- [2] CASLab. *Lecture 4 Minterm and Maxterm*. URL: [https://caslab.ee.ncku.edu.tw/dokuwiki/\\_media/course:logic\\_system:lecture\\_4\\_minterm\\_and\\_maxterm.pdf](https://caslab.ee.ncku.edu.tw/dokuwiki/_media/course:logic_system:lecture_4_minterm_and_maxterm.pdf).
- [3] Numato Lab. *FPGA Vs ASIC: Differences Between Them And Which One To Use?* URL: <https://numato.com/blog/differences-between-fpga-and-asics/>.
- [4] Avnet Electronics Marketing. *Zynq™ Evaluation and Development Hardware User's Guide*. URL: [https://reference.digilentinc.com/\\_media/zedboard:zedboard\\_ug.pdf](https://reference.digilentinc.com/_media/zedboard:zedboard_ug.pdf).
- [5] Jaden Mclean and Carmen Hurley. *Logic design. The Science of Microfabrication*. Scientific e-Resources, 2019.
- [6] mepits. *Application Specific Integrated Circuit*. URL: <https://www.mepits.com/tutorial/169/vlsi/application-specific-integrated-circuit>.
- [7] Cal Poly Pomona. *Decoder ROM*. URL: <https://www.cpp.edu/~zaliyazici/sp2002/ece204/ece204-14.pdf>.
- [8] SENRIO. *JTAG Explained (finally!): Why IoT, Software Security Engineers, and Manufacturers Should Care*. URL: <https://blog.senr.io/blog/jtag-explained>.
- [9] Naveen Shlayan. *Course Introduction*. URL: [https://moodle.cooper.edu/moodle/pluginfile.php/81505/mod\\_resource/content/1/Lecture%5C%201.pdf](https://moodle.cooper.edu/moodle/pluginfile.php/81505/mod_resource/content/1/Lecture%5C%201.pdf).
- [10] Naveen Shlayan. *Design Flow Example*. URL: [https://moodle.cooper.edu/moodle/pluginfile.php/81506/mod\\_resource/content/1/Lecture%5C%203.pdf](https://moodle.cooper.edu/moodle/pluginfile.php/81506/mod_resource/content/1/Lecture%5C%203.pdf).
- [11] Naveen Shlayan. *Getting Familiar with ZedBoard and Vivado Installation*. URL: [https://moodle.cooper.edu/moodle/pluginfile.php/81504/mod\\_resource/content/1/Week%5C%202-Lecture2.pdf](https://moodle.cooper.edu/moodle/pluginfile.php/81504/mod_resource/content/1/Week%5C%202-Lecture2.pdf).
- [12] Naveen Shlayan. *Introduction to Digital Logic Design: Combinational Logic*. URL: [https://moodle.cooper.edu/moodle/pluginfile.php/81904/mod\\_resource/content/1/Week3-Lecture4.pdf](https://moodle.cooper.edu/moodle/pluginfile.php/81904/mod_resource/content/1/Week3-Lecture4.pdf).
- [13] xilinx. *Vivado Design Suite - HLa Editions*. URL: <https://www.xilinx.com/products/design-tools/vivado.html>.
- [14] xilinx. *ZedBoard Zynq-7000 ARM/FPGA SoC Development Board*. URL: <https://www.xilinx.com/products/boards-and-kits/1-elhabt.html>.

## 10 Appendix

### 10.1 tutorial.v

```
1 `timescale 1ns / 1ps
2 ///////////////////////////////////////////////////////////////////
3 // Company:
4 // Engineer:
5 //
6 // Design Name:
7 // Module Name: tutorial
8 // Project Name:
9 // Target Devices:
10 // Tool Versions:
11 // Description:
12 //
13 // Dependencies:
14 //
15 // Revision:
16 // Revision 0.01 - File Created
17 // Additional Comments:
18 //
19 ///////////////////////////////////////////////////////////////////
20
21 module tutorial(
22     A,
23     B,
24     C,
25     D
26 );
27
28 // Define the ports functionality and external wires
29 input wire [0:0]A;
30 input wire [0:0]B;
31 input wire [0:0]C;
32 output wire [0:0]D;
33 wire wire1;
34 or(wire1, A, B);
35 and(D, C, wire1);
36 endmodule
```



## 10.2 test\_tutorial.v

```

1 //Define simulation step size and resolution
2 `timescale 1ns / 1ps
3 ///////////////////////////////////////////////////////////////////
4 // Company:
5 // Engineer:
6 //
7 // Design Name:
8 // Module Name: test_tutorial
9 // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21 //Define testbench module
22 module test_tutorial(
23     );
24
25     //inputs (use regs)
26     reg [2:0] swt;
27
28     //Outputs use wires
29     wire D;
30
31     integer i;
32     reg e_output;
33
34     //Instantiate the Device/Module Under Test (DUT)
35     tutorial ttul(.A(swt[2]),.B(swt[1]), .C(swt[0]),.D(D));
36
37     //Define the same module functionality for the expected value computation
38     function expected_led;
39         input [2:0] switch;
40         begin
41             expected_led = switch[2] | switch[1] & switch[0];
42         end
43     endfunction
44 //Define the stimuli generation and compare with the expected output
45 //Initial blocks occurs only once at time zero
46 initial
47 begin
48     for (i=0; i <8; i = i+1)
49     begin
50         #50 swt=i;
51         #10 e_output = expected_led(swt);
52         // the $display task will print the message in the simulator console window when
         the simulation is run
53         if (D == e_output)
54             $display("LED output matched at %t\n", $time);
55         else
56             $display("LED output mis-matched at %t, expected %b, actual %b\n", $time,
e_output, D);
57         end
58     end
59
60 endmodule

```