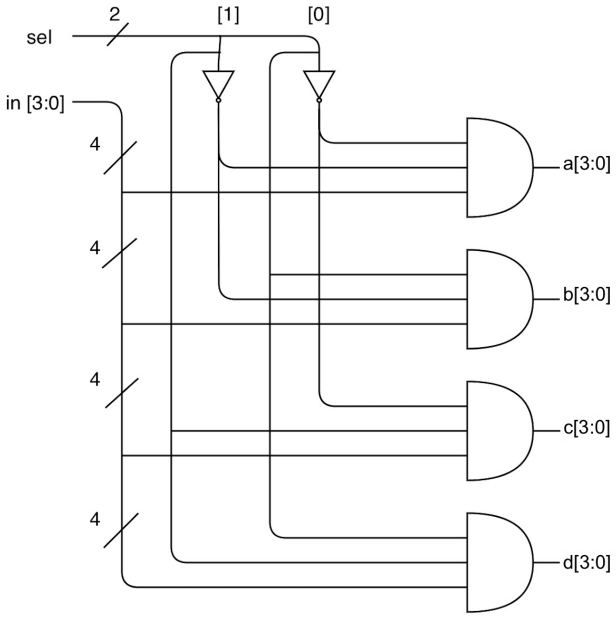
LAB 1 REPORT

DeMUX, Simple 2x2 Crossbar, 4x4 Crossbar, and Toggle Flip-flop
Field Programmable Gate Array (FPGA) and Xilinx Design Constraint (XDC) file

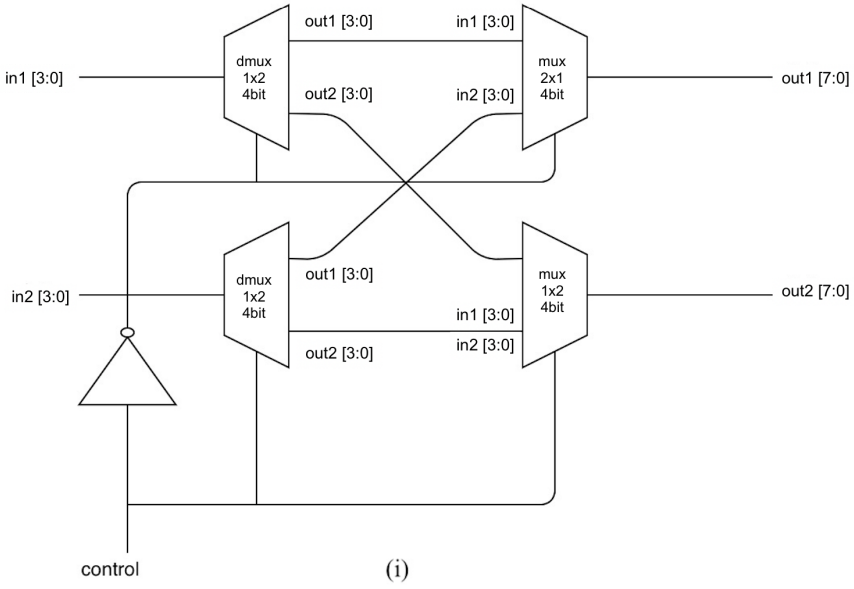
Team 37:

1. 徐美妮 Mary Madeline Nicole 109006205
2. 林之耀 Kevin Richardson Halim 109006277

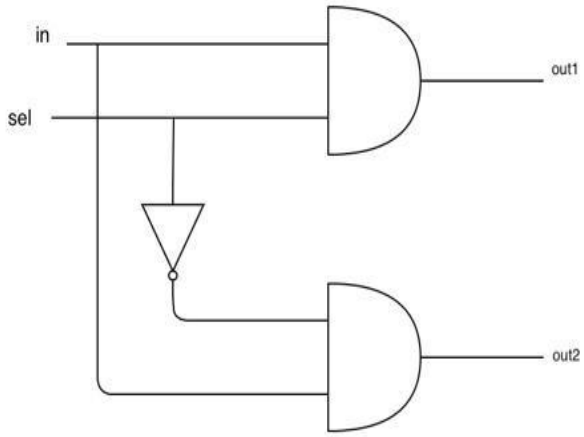
I. Gate-Level Circuits



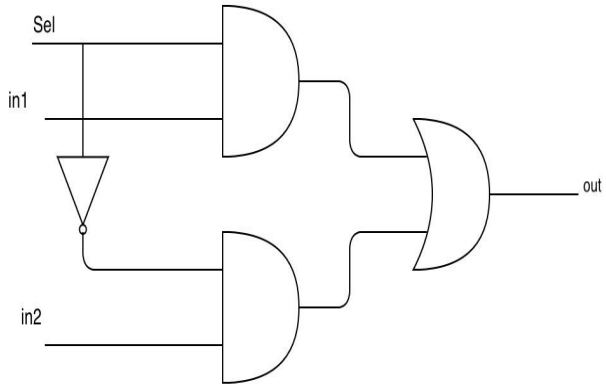
a) 4-bits 1-to-4 Demultiplexer (DMUX)



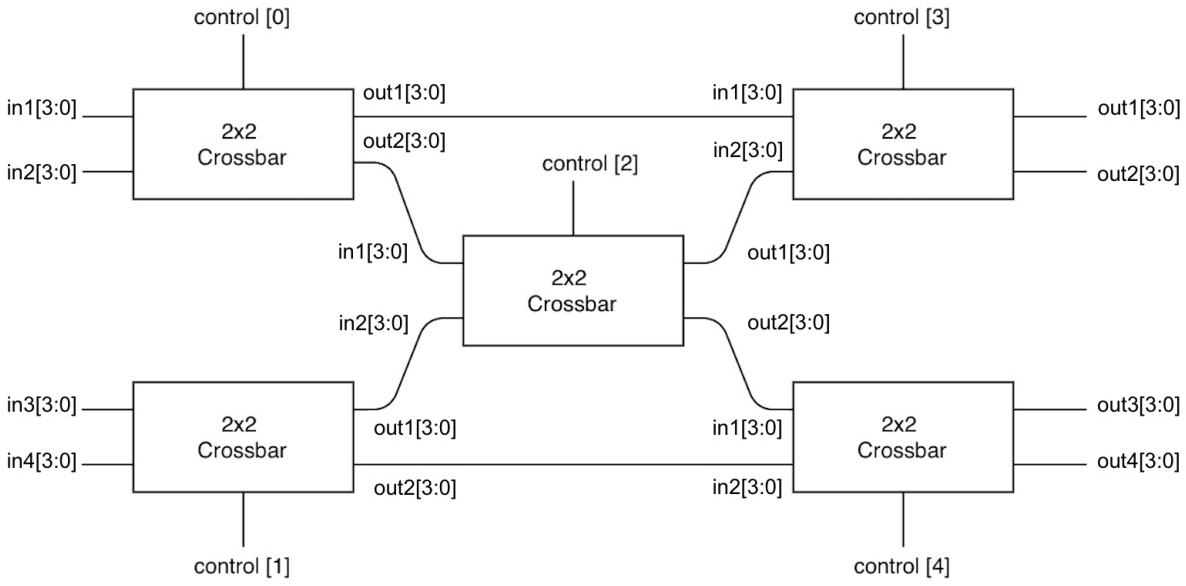
b) (i) 4-bits Simple Crossbar Switch with MUX/DMUX,



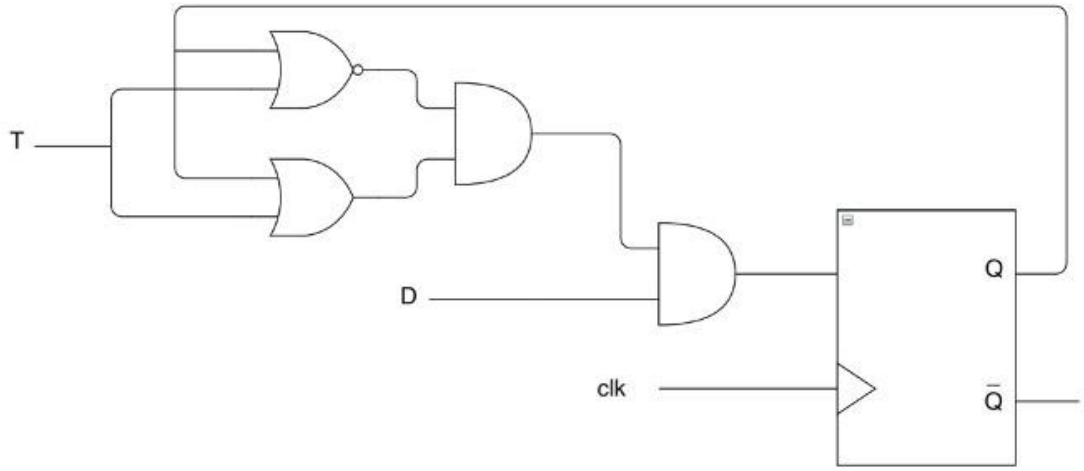
(ii) 1:2 DeMux



(iii) 2:1 Mux

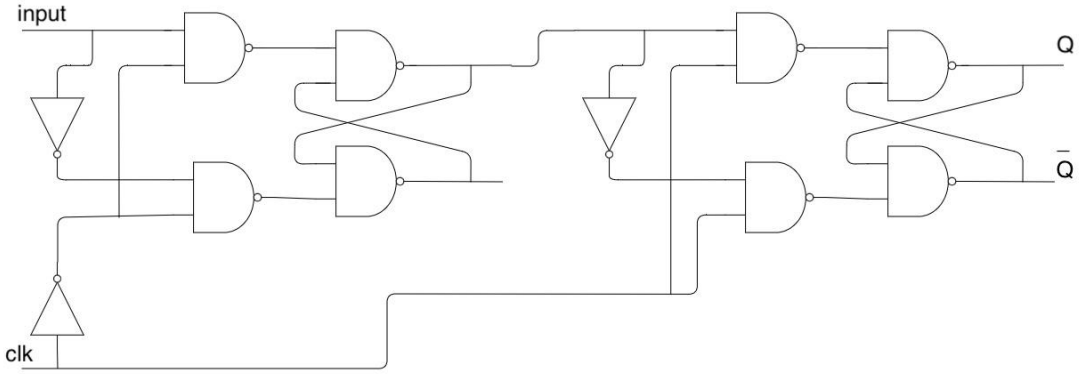


c) 4-bits 4x4 Crossbar with Simple Crossbar Switch



(i)

d) (i)1-bit Toggle Flip-Flop (TFF) with D Flip Flop



(ii)

(ii) D Flip Flop from D Latch

II. Design Process

First, understanding the assignment that is given. Implement the behavior using the gate-level description function. Start by drawing/sketching the design. We are using the breaking to smaller chunks method. To put it simply, we provide a smaller module and use it to build a bigger module.

Second, we wrote the module on verilog files. Implement each gate and each wire or/and register on the file to imitate the real gate behavior. After finishing the files, we check every spelling and every syntax.

Third, we build the testbench. Testbench is pretty useful to check whether or not the module works as we want. Wrote a testbench such as a test-case and we came up with some critical conditions. Critical conditions were made relying on the knowledge when the module would break.

Finally, we debug the code and reiterate each line to fix some bugs that we faced, such as wrong I/O, swapped variables, false naming, wrong behavior, typos, ect.

Modules:

1. (Gate-level) 4-bit 1-to-4 demultiplexer (DMUX)

Design demultiplexers by mimicking their behavior on each bit. Hence, using *AND gate* would be perfect to tackle this task. Each output will get input from *AND gate* filtering the input. If the number assigned does not match, the 0 values will be passed to the output. 00 for a, 01 for b, 10 for c, and 11 for d. Since, we were asked to only use the gate description function, We used *NOT* to manipulate the data for 00 to pass *AND gate*.

2.1.1 Truth Table of 1-to-4 DMux

Input	Sel1	Sel0	d	c	b	a
Data	0	0	0	0	0	Data
Data	0	1	0	0	Data	0
Data	1	0	0	Data	0	0
Data	1	1	Data	0	0	0

2. (Gate-level) 4-bit simple crossbar switch with MUX/DMUX

Design the crossbar by breaking it into smaller chunks of modules. Building MUX and DMUX, since we had the modules from the previous basic lab and from the DMUX before. We built the smaller modules of 2x1 MUX and 1x2 DMUX that were later used to construct the 2x2 Crossbar. Mimicking the behavior while modifying it to offer 2 selections on MUX/DMUX.

Combining two modules on one module and connecting the I/O on port level. Thus, the behavior would be as predicted if there's no syntax error or any wrong presentation.

3. (Gate-level) 4-bit 4x4crossbar with simple crossbar switch

Implement the previous modules from 2x2 Crossbar and combine it as one module. Wiring the data flow from each smaller module and connecting it as one network to work as a 4x4 crossbar switch.

Some configurations are not routable for this crossbar. For example, from the Gate level circuit in part I, when input 1 goes to output 3, it is not possible for input 2 to go to output 4, same as when input 3 goes to output 1, input 4 cannot go to output 2. (In other words, neighbor inputs cannot both go to the outputs that are not adjacent to them, or the outputs that require them to use another crossbar) This happens because the third crossbar in the middle is already used for their respective neighbor inputs (1 to 2, 3 to 4). Therefore there is no possible way for the respective neighbor input to go to the cross output that is not adjacent to it.

2.3.1 Configurations that are not routable

Input	Configuration 1	Configuration 2	Configuration 3	Configuration 4
Input 1	Output 3	Output 4	Output 3	Output 4
Input 2	Output 4	Output 3	Output 4	Output 3
Input 3	Output 1	Output 1	Output 2	Output 2
Input 4	Output 2	Output 2	Output 1	Output 1

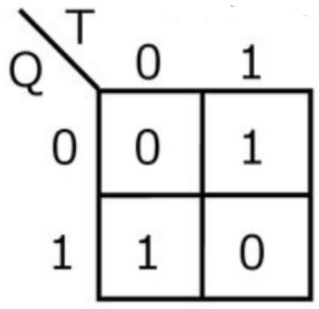
4. (Gate-level) 1-bit toggle flip flop (TFF)

Implement TFF by reusing the D-Flip Flop modules to manage the data flow, with an extra condition. Where if the Toggle is 1, the Output would switch. Below description of the truth table and *Kmap* of Toggle Flip Flop gives a clear explanation on how it works. Considering the requirements of not using *XOR gate* directly, the *XOR gate* was constructed by combining *NOR* with *OR* through an *AND gate*. Above, the Gate level circuit of the Toggle Flip Flop shows our implementation of the gates without using *XOR*, also a magnified version of how we implemented our D Flip Flop module which was created by using D master and slave latches.

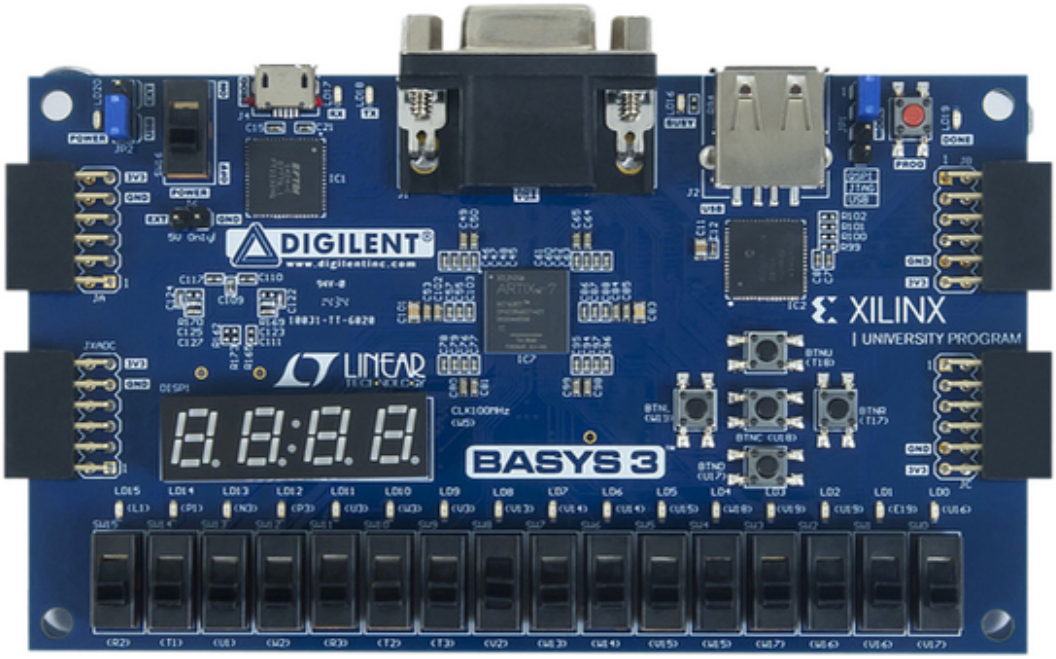
2.4.1 Truth table of TFF

T	***Qn***	***Qn+1***
0	0	0
1	1	1
0	0	1
1	1	0

2.4.2 Kmap of Toggle Flip Flop



5. (FPGA) 4-bit simple crossbar switch with MUX/DMUX



2.5.1 BASYS3 Artix-7 Field Programmable Gate Array (FPGA)

Program the FPGA by providing the module file and the constraint file as an XDC file. Configuring the I/O Ports, we found out that each port needs their own variable on the module and it wont take the same variable, such the last configure would be taken if variables shown multiple times.

```
set_property PACKAGE_PIN <hardware pin> [get_ports {<variable>}]
set_property IOSTANDARD LVCMOS33 [get_ports {<variable>}]
```

Connect the FPGA to the computer with the Vivado software and connect them both using the Hardware Manager provided by the Vivado itself. Click auto-connect and program the FPGA afterwards. After that, you would be able to use the FPGA board as you programmed.

III. List of Contributions

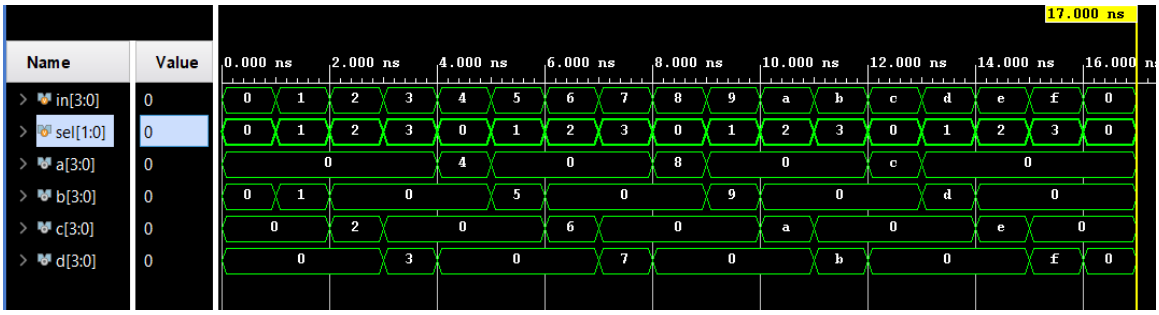
徐美妮 Mary Madeline Nicole 109006205's contributions, such as :

- Design and write module Demux_1x4_4bit,
- Design and write Demux_1x4_4bit testbench,
- Design and write module Crossbar_4x4_4bit,
- Design and write Crossbar_4x4_4bit testbench,
- Design and write module Toggle_Flip_Flop,
- Design and write Toggle_Flip_Flop testbench,
- Draw Gate level circuit,
- Program the FPGA board,
- Write the report.

林之耀 Kevin Richardson Halim 109006277's contributions, such as :

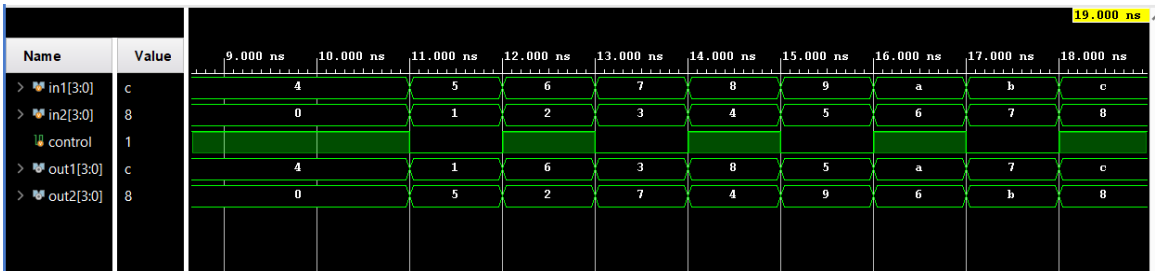
- Design and write for module Simple Crossbar Switch,
- Design and write Simple Crossbar Switch testbench,
- Design and write module Crossbar_4x4_4bit,
- Design and write Crossbar_4x4_4bit testbench,
- Design and write module Toggle_Flip_Flop,
- Design and write Toggle_Flip_Flop testbench,
- Simulate and synthesis of modules in Vivado,
- Design and write the XDC file,
- Program the FPGA board,
- Write the report.

IV. Design Test



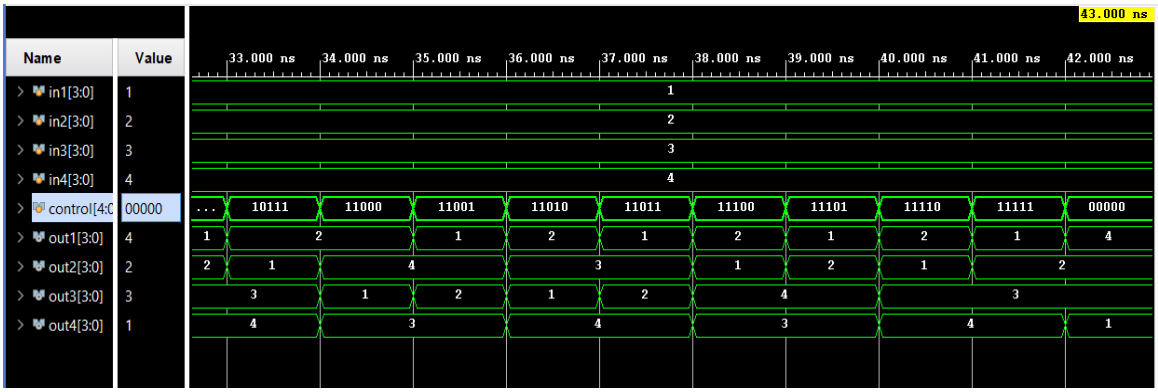
4.1 (Gate-level) 4-bit 1-to-4 demultiplexer (DMUX)

Write down some testbench code, to set some value and modify it to be a test case for observing the program behaviour in such conditions. As we can see, the value is the same between the "assigned" and the input. The value for the other output is 0. Thus, the design of this program is successful as wanted.



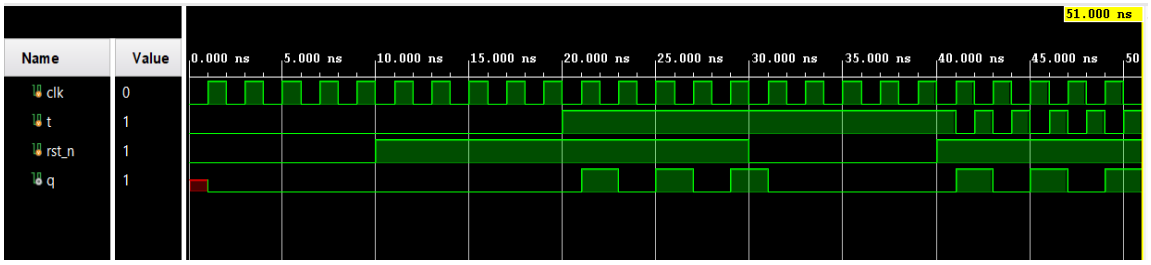
4.2 (Gate-level) 4-bit simple crossbar switch with MUX/DMUX

Setting up the variable input1 and 2, making sure both have different values and setting the value of the control to fluctuate between 1 and 0. Run a simulation and observe the behaviour. Since the value is switched whenever control got 0 assigned value and vice versa whenever control got 1 assigned value. Hence, the code works as it should.



4.3 (Gate-level) 4-bit 4x4 crossbar with simple crossbar switch

Set up 4 variables input1, input2, input3, input4, and the control. Initialize and give them values of 0. To make sure that the control would change between the range 0 to 3, add the bit by 1 on every delay. Run the simulation and observe the behaviour of each control. Values of the switching control will enable the transmission of the input to their respectives output accordingly. As every input has a connection to every output port. The simulation results showed that the program was successfully created.



4.4 (Gate-level) 1-bit toggle flip flop (TFF)

Set up all variables to 0 at first. Then invert the clock between 1 and 0 every delay. As the TFF is toggled when the set and reset inputs alternatively change by becoming the trigger. The flip flop has a control signal, a clock enable where every positive edge will reset. When tested with the testbenches, the outputs toggles back and forth from their previous states every time it was triggered. Since the output at Q only changes state every posedge and clock pulse, the output is equal to half the frequency of the clock, which is the toggling action. Therefore the program works successfully.

V. What we have learned from Lab 1

- ➔ Program the Basys3 FPGA board. We struggled to connect the FPGA board.
- ➔ Creating and configuring the XDC file settings. Set a specific configuration of I/O Port to link the verilog code with the FPGA board as an interface.
- ➔ The importance of splitting modules. Because of the restriction of only using basic gates, solving the problem is difficult when a lot of bits are used, when splitting the modules into multiple sub modules, it makes the main module design easier to do and neater.
- ➔ Maintain a good verilog code structure. Implementing smaller modules into bigger ones for efficiency and easier development.
- ➔ How to create our own testbenches, making sure it covers every case and debugging it.