ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	5
EXPERIMENT DATE	:	14.04.2021
LAB SESSION	:	FRIDAY - 10.30
GROUP NO	:	G1

GROUP MEMBERS:

150170908	:	MUHAMMAD RIZA FAIRUZZABADI
070170364	:	AİŞE HÜMEYRA BOZ

SPRING 2021

Contents

1 INTRODUCTION

In this project, we were required to complete two distinct tasks, preliminary and experiment. First in preliminary part, we recalled clocked synchronous sequential circuits. Then, we tried to analyze the circuit in part1. In the experiment part, we implemented all modules. Then we tested them with the given values.

2 MATERIALS AND METHODS

2.1 MATERIALS

Tools Used

- Vivado Design Suite - Xilinx
- Latex (overleaf.com)
- Logisim

Firstly we recalled the aforementioned topics and noted the important characteristics we have to keep in mind in the implementation of the modules. We then implement and program the logic circuits alongside their NOT,AND,OR gates modules and specific modules that constructed for this experiment in Vivado Design Suite, lastly we used overleaf.com to prepare the report document in LaTex.

2.2 PRELIMINARY

In the preliminary part, we recalled and revised what we learnt in Digital Circuits course. There are some features we have to keep in mind in the implementation of the clocked synchronous sequential circuits : In here, we tried to analyze the circuit mostly j-k and t flip flops' structure. We recalled what Moore and Mealy models mean. In part1, the example belongs to the Mealy model.

The expression for the F function that drives the flip-flops:

$$J0 = (A.B)' + Q0$$
$$K0 = (A.B)' + Q1'$$
$$T = (A + B).Q1'$$

Expressions for the next state:

$$\begin{aligned}
Q0+ &= J0.(Q0)' + (K0)'.Q0 \\
Q0+ &= ((A.B)' + Q0).(Q0)' + ((A.B)' + Q1')'.Q0 \\
Q1+ &= ((A + B).Q1')'.Q1 + ((A + B).Q1').Q1'
\end{aligned}$$

The expression of the output function G.

$$Z = ((A.B)'.Q0') + Q1'$$

We couldn't complete the state/output table and state transition table. Also, we chose not to implement part2 of the experiment since it was a bonus part, so didn't complete the preliminary part regarding that.

2.3 PART 1

In the first part, we were asked to design an implement a specific circuit design that is given in the pdf file of the experiment. It contains of two flip flops of JK and T, with logic gates in between. For that reason at first we implemented the corresponding JK and T Flip Flops. Since we have implemented D Latch and D Flip Flop in the previous experiment, we decided to implement both flip flops using those. T Flip Flop is rather simple, in the module we just instantiated/used D Flip Flop once, with input T being the result of operation XOR between input wire and Q. Since we don't have Q initially, error occured. To fix this we then added one RST / Reset input, which then we use to set the Q in the D Latch module to 0 by operating the original Q with RST's negate. The design is as shown in Figure 1. Then to implement JK Flip Flop, we used the T Flip Flop we just implemented, since JK Flip Flop is basically a T Flip Flop with 2 inputs, and vice versa. To implement this idea we use one OR and two AND. J and K inputs are going into two distinct ANDs, with K being negated before-wards. Then both of the outputs of the two are going inside an OR gate which output being the T input for T Flip Flop. The design is as shown in Figure 2. After designing and testing both of these flip flops, we then implemented the other parts of the given design, resulting in a elaborated design schematic of part 1 which can be seen in Figure 3. We did this part before the preliminary part, therefore we implemented the design in verilog with the exact number of logic gates and with the same wires. The result of the simulation of the implemented design is also as shown in the Result part of the report.

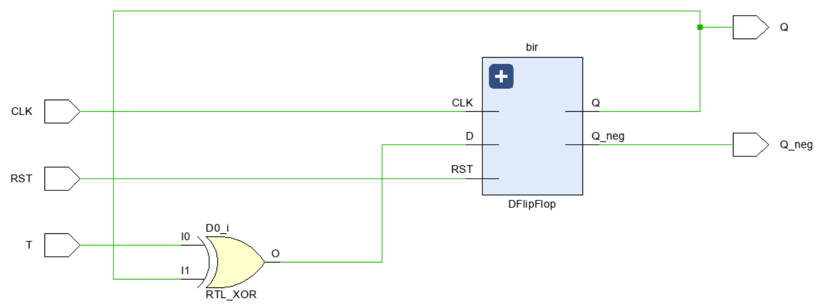

Figure 1: Elaborated Design Schematic of T Flip Flop

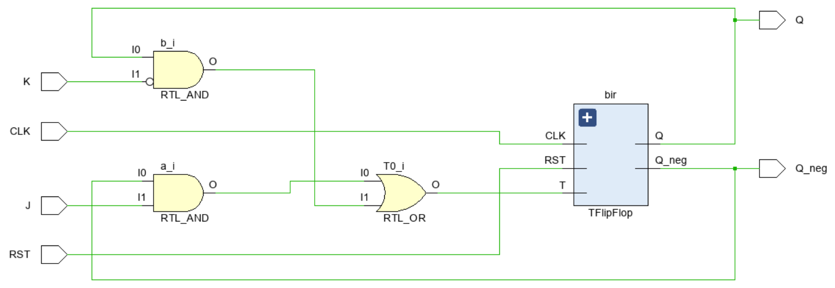

Figure 2: Elaborated Design Schematic of JK Flip Flop

2.4 PART 3

Since the second part is bonus, we directly moved to the third part. In the third part we were asked to design and implement a counter that have 16-bit input data, load, clock, direction, 3-bit value to be operated and clear inputs. Load is to determine whether the counter loads a new input or continues doing the increment/decrementing operations. Direction is to determine which operation to be done between 1 / increment or 0 / decrement. Clear is to clear the counter register. To implement this at first we created a reg data type 16 bit counter to store the result of calculations before assigning to output. We use always block, with posedge clk so the module operates in positive edge mode according to clock. We use if statements to do the required assortments, like to do the increment/decrement operations when load is 1, or to load the input data value to counter reg when load is 0. Also when clear is 0, counter reg's value are assigned with 16-bit zero. Lastly we assigned the latest counter reg value to the output wire O, concluding our implementation of this counter. The elaborated design schematic of the implementation is as shown in the figure 4. The simulation result is also as seen in the Result part of the report. We give a variety of inputs to test the correctness of each function of the counter.

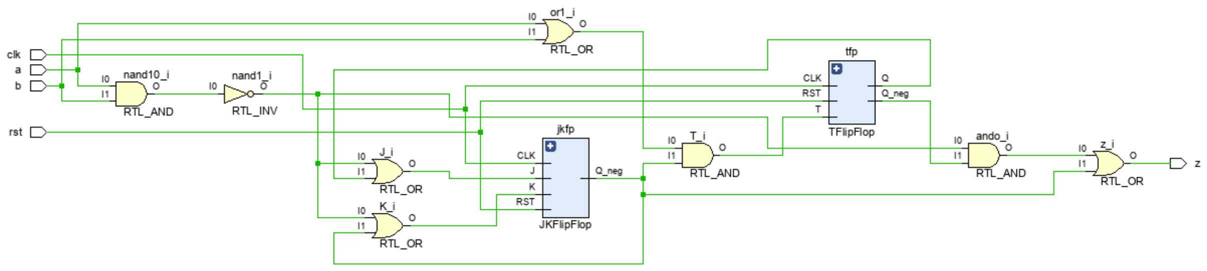

Figure 3: Elaborated Design Schematic of Part 1

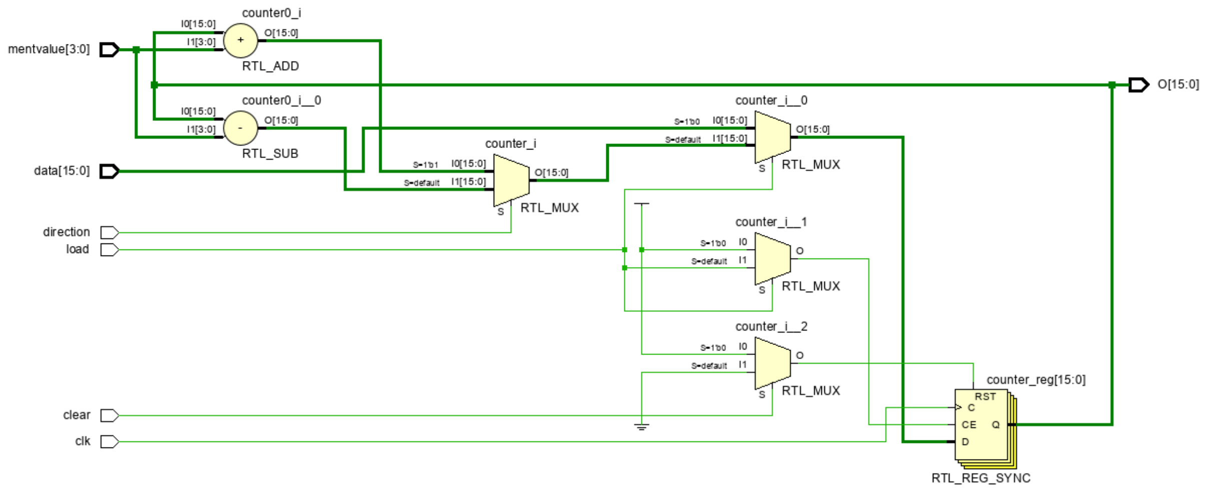

Figure 4: Elaborated Design Schematic of Part 3 - Counter

2.5 PART 4

In the fourth part, we were asked to design many modules using part 3's module, the counter. For all parts of the part4, we showed circularity. For 0to40 example, we had two inputs, clock and initiative and output was a 16-bit number. We had reg inputs namely ctr, clear, log. We used always block, with posedge clk so the module operates in positive edge mode according to clock. If the output was equal to 36, we asked clear to be 0 when clear becomes 0, the last if block becomes active saying load to be 0 and ctr to be 0. If initiate is 0, load becomes 0 and ctr becomes 0, then when it grow to 1, load becomes 1 and ctr becomes 0 thus, the implementation starts. Since the increment value 2 and addition sign was defined in counter module, implementation occurs when initiator increases from 0 to 1 in simulation code.

The same applies to 350to371 module. The start point is now 350 and increment value is 3. Addition occurs in each time clock becomes positive edge.

For 93to5 module, decrement value was 4 and it goes back to 93 after 5 value. The other difference was the output's equal to 13, in order to turn it back to the initial value.

For 22525to22535 module, increment value was 1. Again we asked for the output value to be 22533, in order to return to its initial value. For all modules, we benefited from counter module constructed in part3. Simulation outputs are present in the Result part of the report.

3 RESULTS

We completed the first three tasks we were asked in the experiment. Unfortunately we couldn't do the second part due to lack of time caused by the high amount of other works we have for other lectures this week. Since we draw and design the expressions, it was not hard to validate their correctness through simulations. We added required tables, images, circuits to the specific places materials methods part of the experiment. Lastly, we supplemented our images namely elaborated design, simulations to the report for each part.

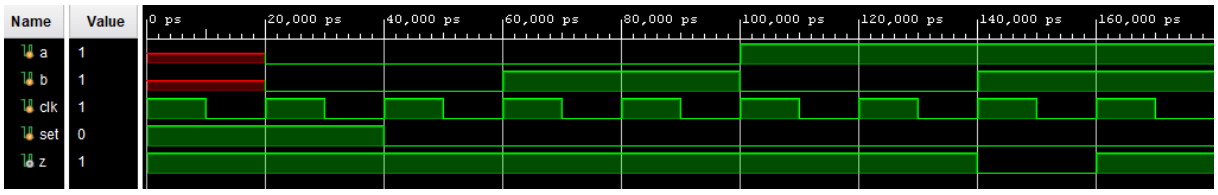

Figure 5: Simulation Output of Part 1

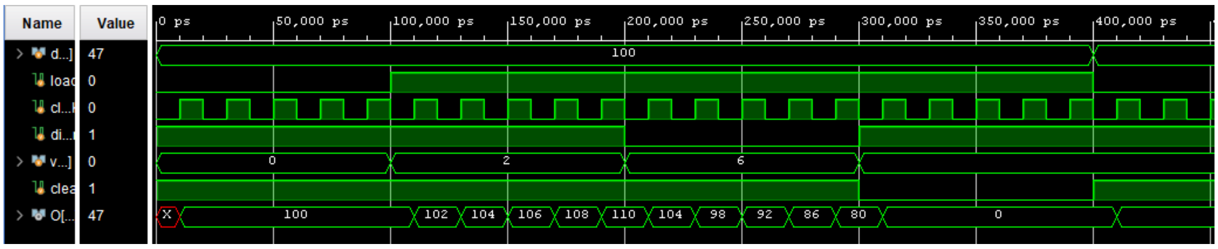

Figure 6: Simulation Result - Part 3

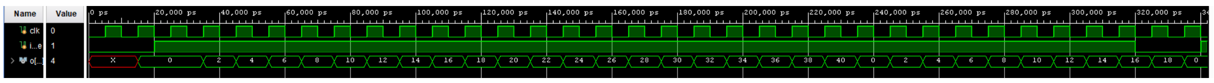

Figure 7: Simulation Output of Part 4-a

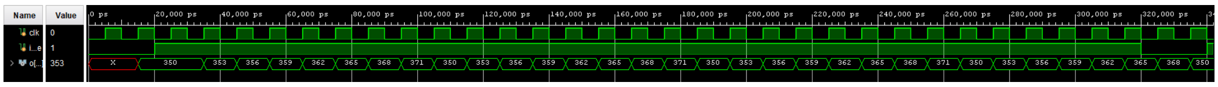

Figure 8: Simulation Output of Part 4-b

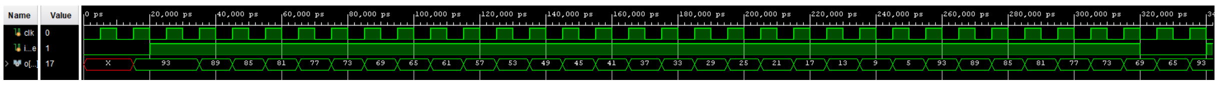

Figure 9: Simulation Output of Part 4-c

4 DISCUSSION

In preliminary part, we recalled the concept of t flip-flop and j-k flip flops and clocked synchronous circuits. We implemented t flip-flop and j-k flip flops with reset input. First we set the module of t flip-flop using d flip-flops and then we implemented j-k flip-flop with t flip-flops. For the clocked synchronous circuit, we assigned what we saw. In the third part, we implemented a 16-bit counter with load, mentvalue, direction, clear inputs using always block. We set the outputs according to the inputs. For part 4, we implemented separate modules as it was asked that way. Finally, we could be able to design all modules correct as the output was both incrementing/decrementing and was circular which means it returned back to the initial value.

5 CONCLUSION

We successfully implemented the modules and simulated them, but unfortunately we did not have enough time to complete part 2. The experiment was harder than the previous ones. We faced with troubles while setting initial value in part 1. Then we solved it with reset input. The other challenging point was making counter work, since it had many inputs. All in all, we managed to complete all parts and hope their answers to be correct. Lastly, we were glad to learn new concepts in Verilog.

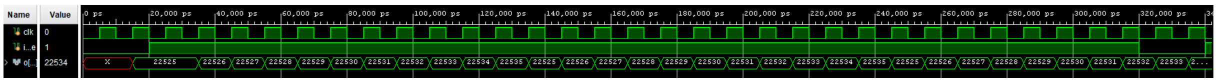

Figure 10: Simulation Output of 4-d