FACULTY OF ENGINEERING AND TECHNOLOGY
ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT
ADVANCED DIGITAL DESIGN ENCS3310
COURSE PROJECT

Prepared by:

Abdelrhman Abed 1193191

Instructor: **Dr. Abdellatif Abu-Issa**

Section: 2

Date: 2/9/2023

Abstract

The project's goal is to create a Finite State Machine (FSM) is intended to recognize the particular sequence "1011." The FSM functions by transitioning between a number of states in response to input bits. When it starts out, it is in the first state, but as soon as it reads "1," it moves on to the second state. When it comes across "0," it then switches to the third state. The FSM advances to the fourth state, which denotes successful detection of the intended sequence, after receiving a second "1." The FSM goes back to its initial state if any unexpected input is received. The FSM architecture makes sure that input bit streams containing the "1011" sequence are correctly identified.

Table of Contents

Brief introduction and background:

- Introduction:

The main goal of the project is to develop and implement a Moore Finite State Machine (FSM) that can identify the specific binary sequence "1011" in a stream of binary input data. Finite State Machines, which are used to represent and regulate sequential activity, are crucial elements in the design of digital logic. The project builds a Moore FSM specifically designed to detect the target sequence using T-flip-flops as essential building elements.

-Background:

It is frequently necessary for digital systems to be able to identify particular patterns inside binary data streams. A computer model known as a finite state machine (FSM) offers a structured and methodical method for addressing sequential patterns. The Moore FSM is a form of FSM in which inputs cause state transitions and outputs are connected to states.

The T-flip-flop, which changes its state in response to a particular input signal, serves as the project's central component. Its actions are in line with the requirement to switch between states when certain binary patterns are recognized. A Moore FSM can be built to detect complex sequences by interconnecting numerous T-flip-flops and customizing their connections.

The number sequence "1011" serves as a real-world illustration of the project's fundamental ideas. In order to detect this sequence, the FSM must react to particular input combinations and move through many states until arriving at the ultimate acceptance state. As part of the design process, a state diagram that describes the sequence detection procedure is created. T-flip-flops are then given states, state transitions are decided upon based on input conditions, and the necessary output logic is established for each state.

The capability of accurately detecting sequences and patterns becomes increasingly important as digital systems get more complex. This project offers insight into the actual use of T-flip-flops as memory components inside a sequential logic circuit in addition to practical expertise in developing and implementing finite state machines. The research effectively exhibits the power of combining fundamental digital components to produce complex pattern recognition by successfully developing a Moore FSM that correctly detects the "1011" sequence.

Design philosophy:

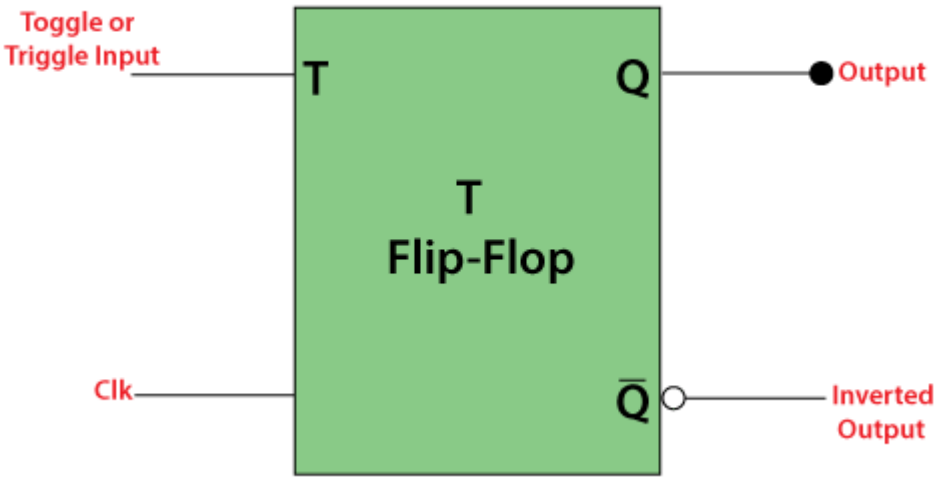
We used in project:

- T-Flip Flops:

In T flip flop, "T" defines the term "Toggle". In SR Flip Flop, we provide only a single input called "Toggle" or "Trigger" input to avoid an intermediate state occurrence. Now, this flip-flop work as a Toggle switch. The next output state is changed with the complement of the present state output. This process is known as "Toggling"'.

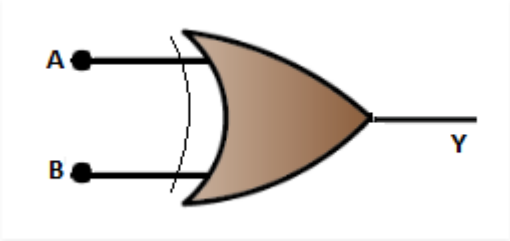
We can construct the "T Flip Flop" by making changes in the "JK Flip Flop". The "T Flip Flop" has only one input, which is constructed by connecting the input of JK flip flop. This single input is called T. In simple words, we can construct the "T Flip Flop" by converting a "JK Flip Flop". Sometimes the "T Flip Flop" is referred to as single input "JK Flip Flop".

Block diagram of the "T-Flip Flop" is given where T defines the "Toggle input", and CLK defines the clock signal input.



-Xor gate:

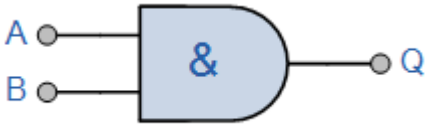
The Exclusive-OR gate or XOR gate is achieved by combining standard logic gates together. XOR gate is used extensively in error detection circuits, computational logic comparators and arithmetic logic circuits. The Exclusive OR gate gives an output only if its two inputs are dissimilar, namely if one of them is high (one) and the other is low (zero).



-And gate:

The output state of a digital logic AND gate only returns “LOW” again when ANY of its inputs are at a logic level “0”. In other words for a logic AND gate, any LOW input will give a LOW output.

The logic or Boolean expression given for a digital AND gate is that for Logical Multiplication which is denoted by a single dot or full stop symbol, (.) giving us the Boolean expression of: A.B = Q.



Results:

This is the code for the T-Flip Flops that takes the clock and reset to trigger the output q and the module was build behaviorally.

```
 //abdelrhman abed ~1193191~
module TFF (clk,reset,t,q);   //T-Flip Flops
    input clk,reset,t;
    output reg q;

    always @(posedge clk or posedge reset)
        begin
        if (reset)
            q <= 1'b0;
        else if (t)
            q <= ~q;
    end
endmodule
```

This's the module for the output detector which takes an inputs and only gives the output if it meets the sequence 1011.

```
module Det (q,det);  //DetectionLogic

    input [0:3]q;
    output reg det;


    always @(*)
        begin
        if (q[0] & ~q[1] & q[2] & q[3])
            det = 1'b1;
        else
            det = 1'b0;
    end

endmodule
```

This’s the module for the structural description of the circuit .

```
31  module SDet(clk,reset,input1,det);   //SequenceDetector
32
33      input clk,reset,input1;
34      output reg det;
35      wire t0, t1, t2, t3;
36      reg [0:3]q;
37
38      TFlipFlop tff0 (.clk(clk) , .reset(reset) , .t(t0) , .q(q[0]));
39
40      TFlipFlop tff1 (.clk(clk) , .reset(reset) , .t(t1) , .q(q[1]));
41
42      TFlipFlop tff2 (.clk(clk) , .reset(reset) , .t(t2) , .q(q[2]));
43
44      TFlipFlop tff3 (.clk(clk) , .reset(reset) , .t(t3) , .q(q[3]));
45
46      assign t0 = (q[0] ^ input1) & (~q[1]);
47      assign t1 = (q[1] ^ ~input1) & (~q[2]);
48      assign t2 = (q[2] ^ input1) & (~q[3]);
49      assign t3 = (q[3] ^ input1) & q[0];
50
51      Det1 (.q(q), .det(det));
52
53  endmodule
```

The code for the behavioral description of the circuit

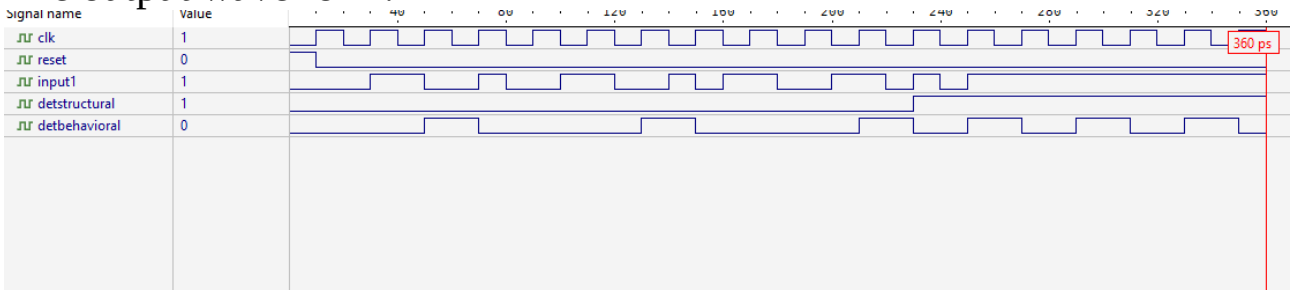
```
55  module SDetBehavioral (clk, reset, input1, det);  //SequenceDetectorBehavioral
56
57      input clk, reset, input1;
58      output reg det;
59      reg [0:3] q;
60
61      always @(posedge clk or posedge reset) begin
62          if (reset)
63              q <= 4'b0000;
64          else begin
65              q[0] <= q[0] ^ input1;
66              q[1] <= q[1] ^ ~input1;
67              q[2] <= q[2] ^ input1;
68              q[3] <= q[3] ^ input1;
69          end
70      end
71
72      always @(*) begin
73          if (q[0] & ~q[1] & q[2] & q[3])
74              det = 1'b1;
75          else
76              det = 1'b0;
77      end
78
79  endmodule
```

The test bench module for the structural and behavioral circuit modules.

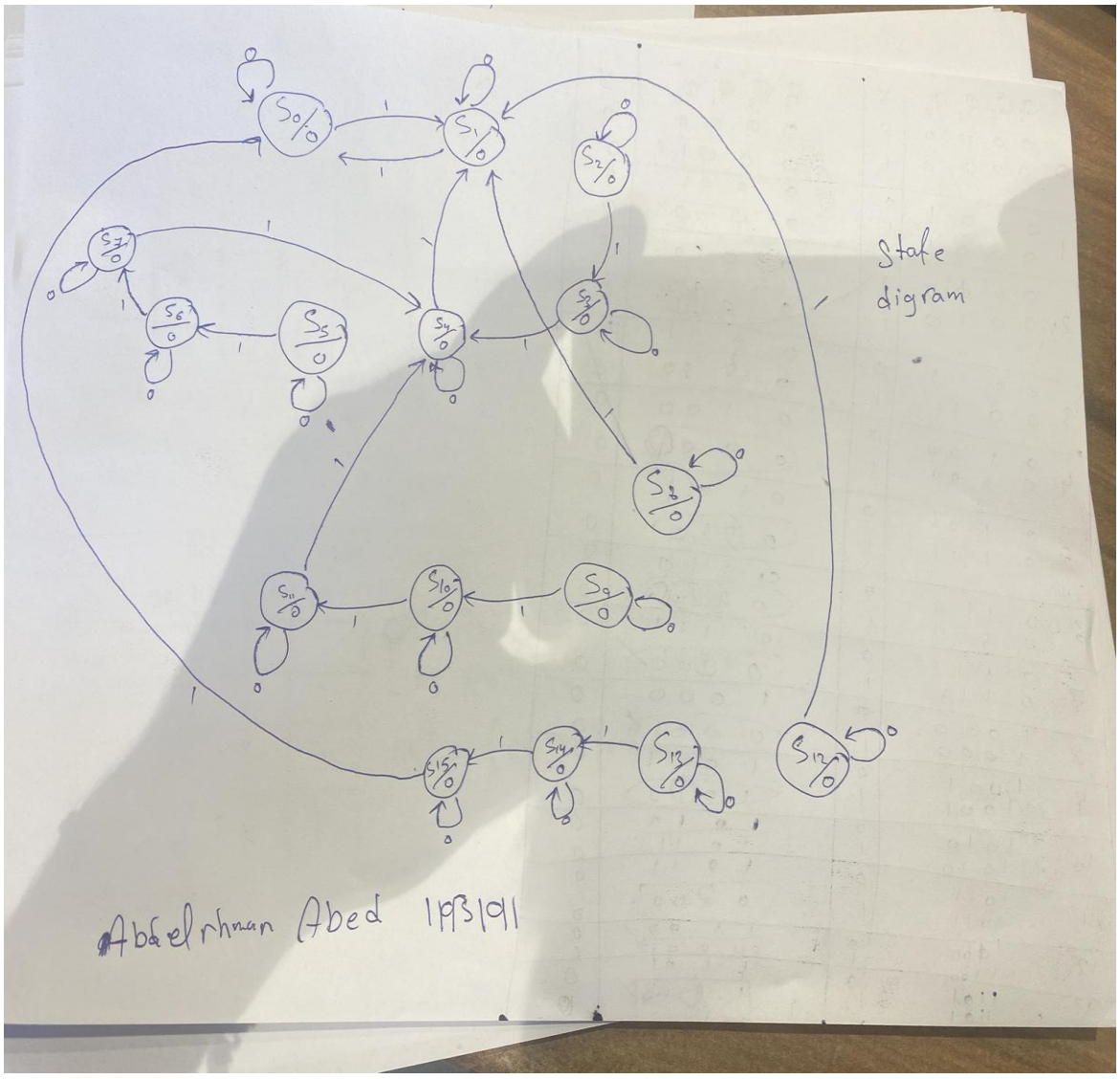
```
module SDet_TB;


    reg clk;
    reg reset;
    reg input1;


    wire detstructural;
    wire detbehavioral;


    SDet structural (.clk(clk),.reset(reset),.input1(input1), .det(detstructural));


    SDetBehavioral behavioral (.clk(clk),.reset(reset),.input1(input1),.det(detbehavioral));


    always
        #10 clk = ~clk;
```

```

    initial
        begin

        clk = 0;
        reset = 1;
        input1 = 0;

        #10 reset = 0;

        for (int i = 0; i < 16; i = i + 1)
            begin
            input1 = i;
            if (detstructural == detbehavioral)
                begin
            #10;
            end
            #10;
        end

        #100
        $finish;
    end

endmodule

```

The output wave form.



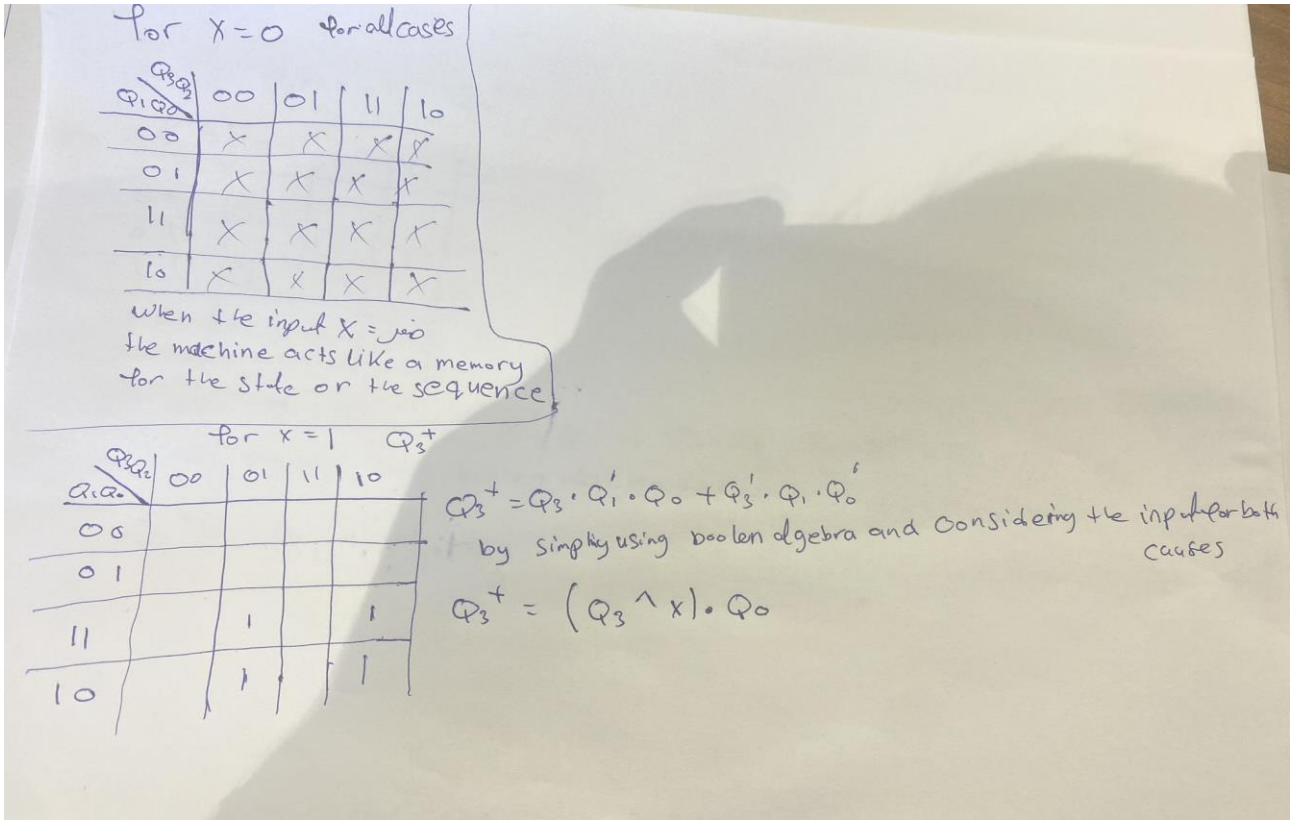
State diagram



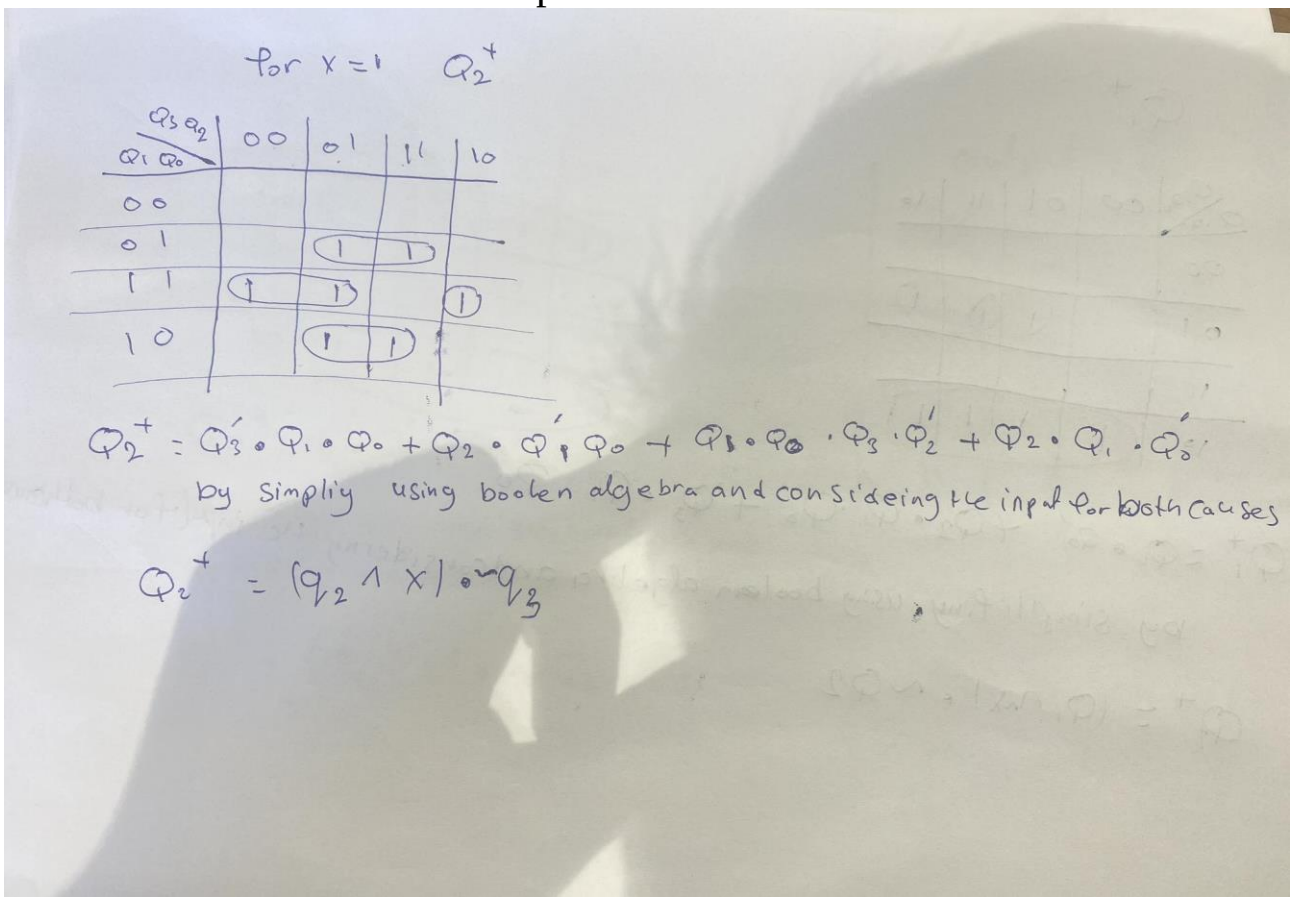
State table

state	Q[3],Q[2],Q[1],Q[0]	input	Next state	output
0	0000	0	0	0
0	0000	1	1	0
1	0001	0	1	0
1	0001	1	0	0
2	0010	0	2	0
2	0010	1	3	0
3	0011	0	3	0
3	0011	1	4	0
4	0100	0	4	0
4	0100	1	1	0
5	0101	0	5	0
5	0101	1	6	0
6	0110	0	6	0
6	0110	1	7	0
7	0111	0	7	0
7	0111	1	4	0
8	1000	0	8	0
8	1000	1	1	0
9	1001	0	9	0
9	1001	1	10	0
10	1010	0	10	0
10	1010	1	11	0
11	1011	0	11	0
11	1011	1	4	0
12	1100	0	12	0
12	1100	1	1	0
13	1101	0	13	0
13	1101	1	14	0
14	1110	0	14	0
14	1110	1	15	0
15	1111	0	15	0
15	1111	1	0	1

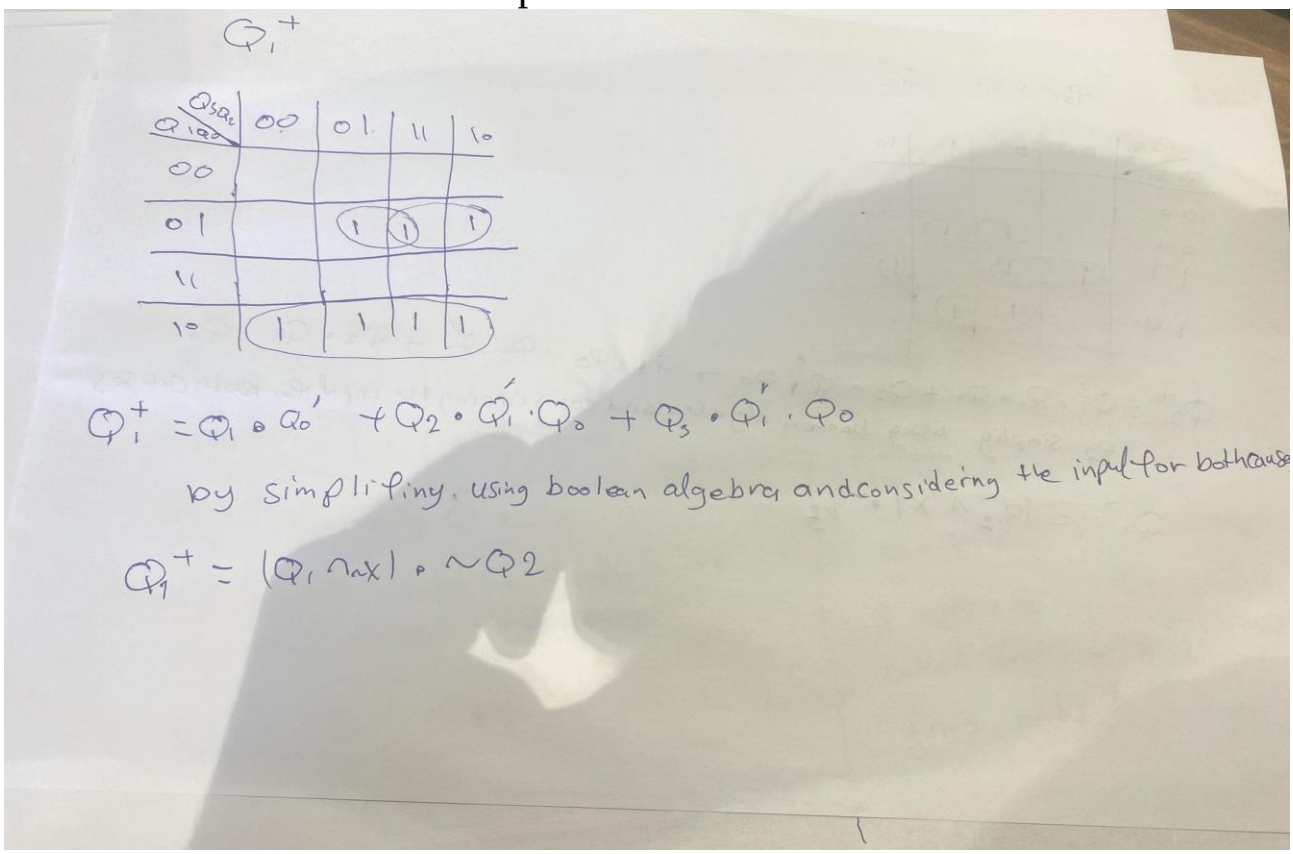
Input for the fourth TFF



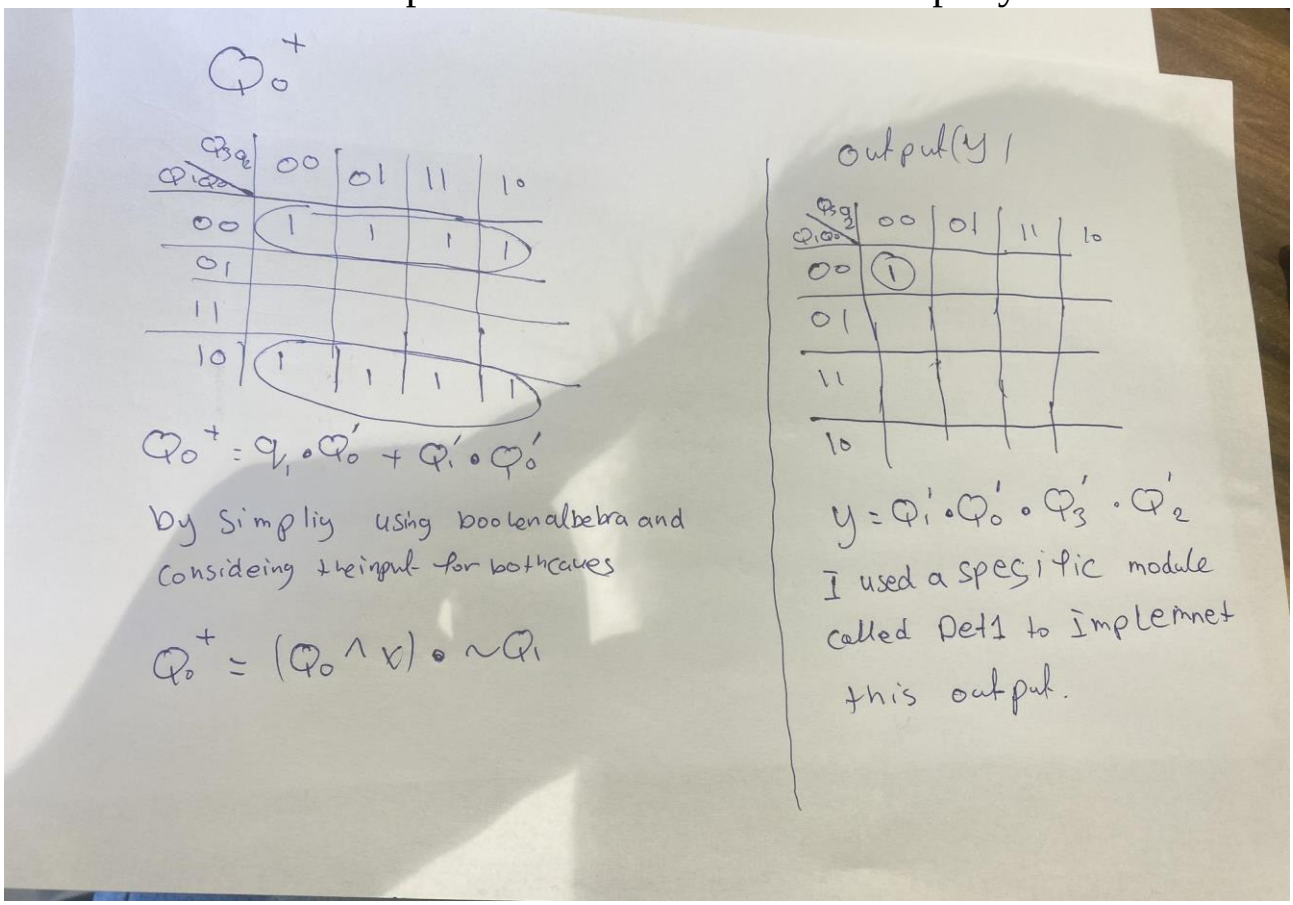
The output for the third TFF



The output for the second TFF



The output of the first TFF and the output y



Conclusions and Future works:

I concluded that there is al slight error in the modules but both of them detected the sequence as they should the error in the wave form could be from the clock's alignments between the two modules I believe that if I separate the inputs and reset and clocks and assign a specific inputs for each case they will function as required.

Also, I concluded that we can depend on the more ~fsm~ to design several circuits for any problem we want to obtain a solution for it.

For future works I want to test the theory of why the clocks resulted in change of how the output came and try to fix it.