EC280 Digital System Design

Assignment 2

Authors:
Anirudh BH (16EC105)
Manan Sharma (16EC118)

March 18, 2019

Question 1

Construct an FSM that checks if a binary number is divisible by 3. Specifically, your FSM should take input bits sequentially starting from LSB and output REM= 1 if the number is not divisible and REM = 0 if the number is divisible. Write a synthesisable Verilog code for the same.

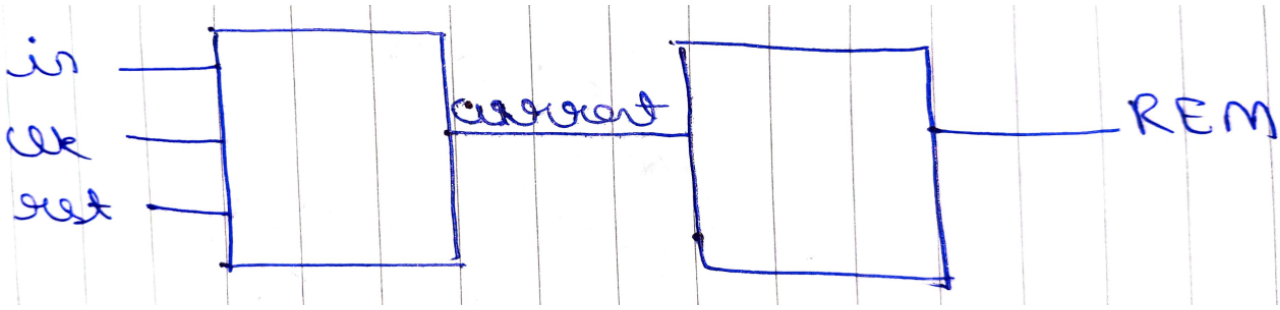

Figure 1: Block Diagram for Q1

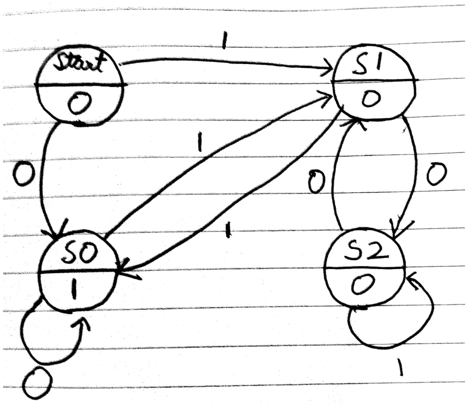

Figure 2: FSM for Q1

Code 1: Verilog Code for Q1

```
`timescale 1ns / 1ps
//Moore machine implementation for divisbility by 3 check.
module remainder3(
    input in,           //input
    output REM,
    input clk,
    input rst           //active high reset
    );
    localparam [1:0] start=2'b00, s0=2'b01, s1=2'b10, s2=2'b11;
    reg [1:0] current;
    always @(posedge clk, posedge rst)begin
    if(rst) begin                   //reset condition
    current<=start;
    end
    else
        case(current)               //FSM starts
        start: if(in==1'b1) current <= s1;
                else current <= s0;
        s0: if(in==1'b1) current <= s1;
            else current<=s0;
        s1: if(in==1'b1) current <= s0;
            else current <=s2;
```

```
        s2: if(in==1'b1) current <=s2;
            else current<=s1;
        default: current <= s0;
        endcase
    end
    assign REM = (current==s0)? 1'b1 : 1'b0;
endmodule
```

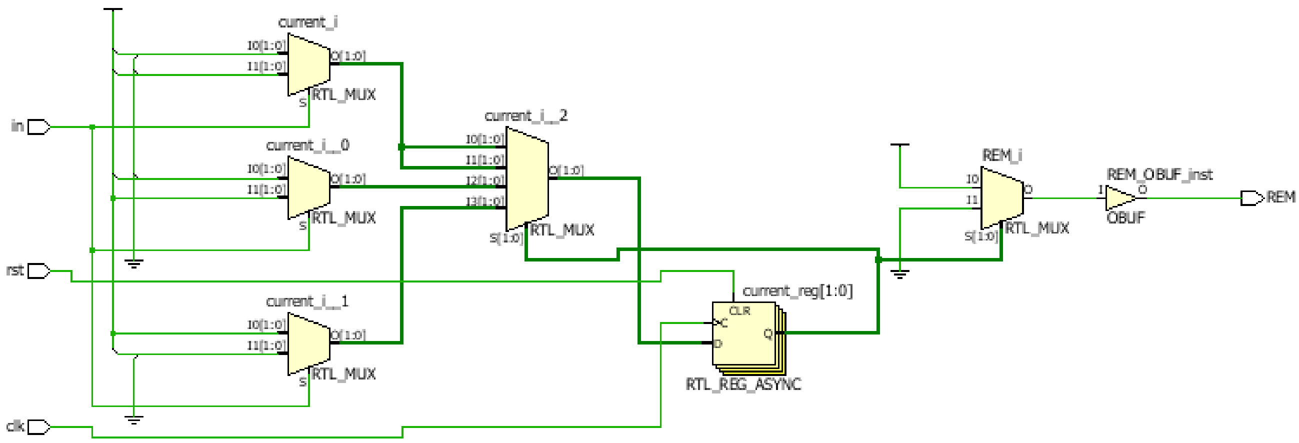

Figure 3: Elaborated Design for Q1

Code 2: Testbench for Question 1

```
`timescale 1ns / 1ps
//Testbench for divisibility by 3 check
module testBench_q1();
reg in, clk, rst;
wire REM;

remainder3 rem( .in(in), .REM(REM), .clk(clk), .rst(rst));

initial begin
clk = 1'b1;
rst = 1'b1; #100
rst =1'b0; in = 1'b0; #50;
in = 1'b0; #50;
in = 1'b1; #50;


in = 1'b0; rst = 1'b1; #100;
rst = 1'b0;
in = 1'b0; #100;
in = 1'b0; #100;
in = 1'b1; #250;
$finish;
end

always begin
clk = #50 ~clk;
end
endmodule
```

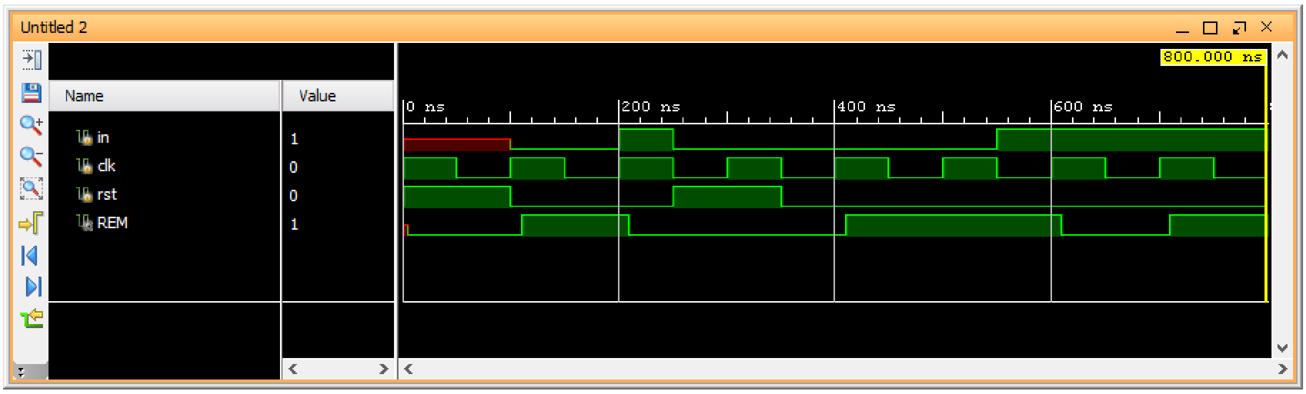

Figure 4: Post Implementation Output for Q1

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	LUT Flip Flop Pairs (63400)	Bonded IOB (210)	BUFGCTRL (32)
remainder3	2	2	1	2	2	4	1

Figure 5: Resource Utilization for Q1

Question 2

Design a finite state machine for 8-bit restoring division. Write a synthesisable verilog code for the same.

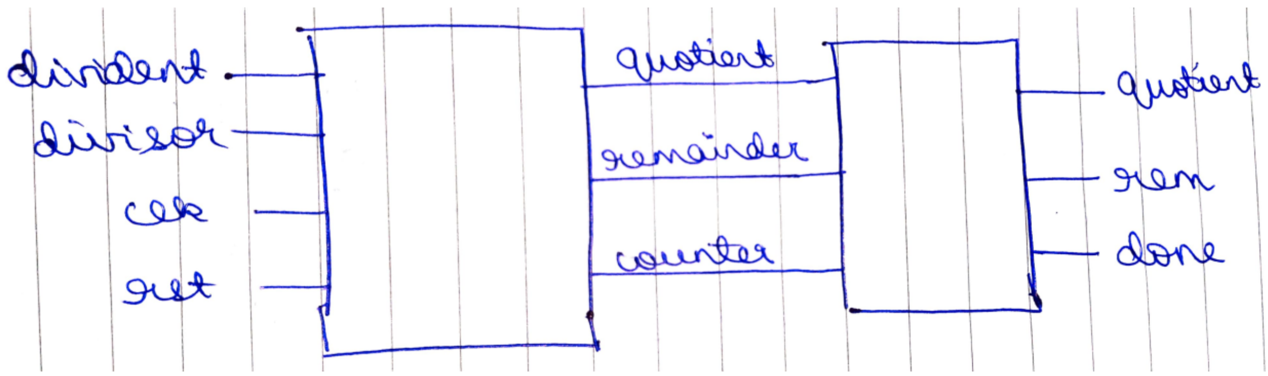

Figure 6: Block Diagram for Q2

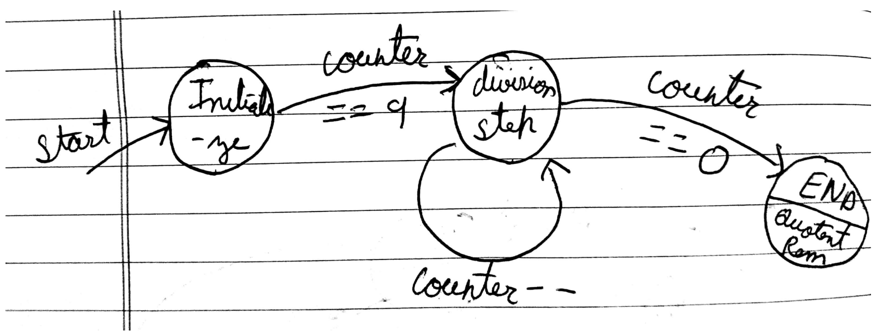

Figure 7: FSM for Q2

Code 3: Verilog Code for Q2

```
`timescale 1ns / 1ps
```

```
// Restoring 8 bit divison
module restoring8bit(
    input [7:0] divident,
    input [7:0] divisor,
    output reg [7:0] quotient,
    output reg [7:0] rem,
    input clk,
    output reg done,            //done becomes 1 when operation is complete
    input rst                   //active low reset
    );
    reg [15:0] remainder;
    reg msbpartial;
    reg [16:0] neg_divisor, temp;
    reg [3:0] counter;          //counter to keep track of the no of clock cycles
        elapsed

    always @(posedge clk, negedge rst)
    begin
    if (rst==0)
    begin
        done <= 0;
        counter <= 9;   // 8 clocks for ouput and 1 clock to load
        quotient <= 0;
        msbpartial <= 0;
        rem <= 0;
    end
    else
    begin
        if(counter == 9)        //inital loading of data
        begin
            remainder <= divident;
            neg_divisor <= {(~divisor + 1),8'b00000000};
            counter <= counter - 1;
        end
        else if(counter!=0)
            begin
            {msbpartial,remainder} = ({msbpartial,remainder}<<1);
            temp = ({msbpartial,remainder}) + neg_divisor;
            quotient = (quotient<<1);
            if(temp[16]==1)
                quotient = quotient;
            else
                quotient = (quotient + 1'b1);
            {msbpartial,remainder} = (temp[16])?{msbpartial,remainder}:temp;
            counter = counter - 1;
            if (counter==0)
                begin
                        done=1;
                        rem = remainder[15:8];
                end
            else
                done=0;
            end
         else           //holding the results
             begin
             quotient = quotient;
             rem = rem;
             end
        end
    end
endmodule
```

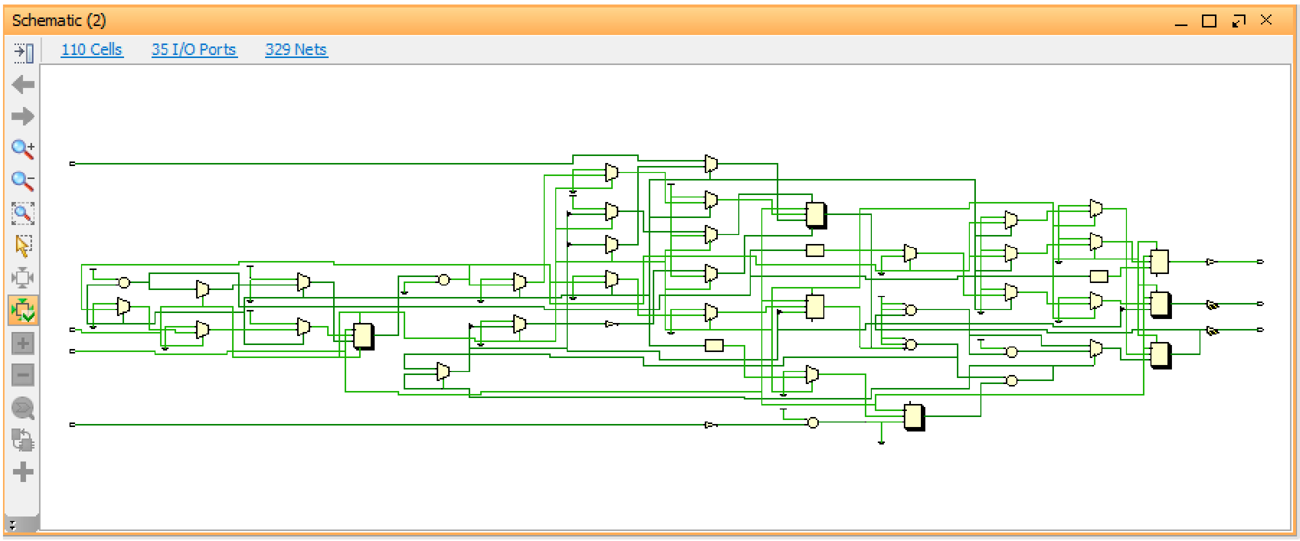

Figure 8: Elaborated Design for Q2

Code 4: Testbench for Q2

```
`timescale 1ns / 1ps
//Testbench for restoring division
module q2testbench();
reg [7:0] divident;
reg [7:0] divisor;
wire [7:0] quotient;
wire [7:0] rem;
reg clk;
wire done;
reg rst;

restoring8bit res(.divident(divident), .divisor(divisor), .quotient(quotient), .
    rem(rem), .clk(clk), .done(done), .rst(rst));

initial begin
clk = 0;
rst=0;#100;
rst=1;
divisor = 4;
divident = 15;#1100;

rst=0;#100;
rst=1;
divisor = 2;
divident = 16;#1100;

rst=0;#100;
rst=1;
divisor = 8;
divident = 239;#1100;

rst=0;#100;
rst=1;
divisor = 1;
divident = 255;#1100;
$finish;
end

always begin
```

```
clk = #50 ~clk;
end
endmodule
```

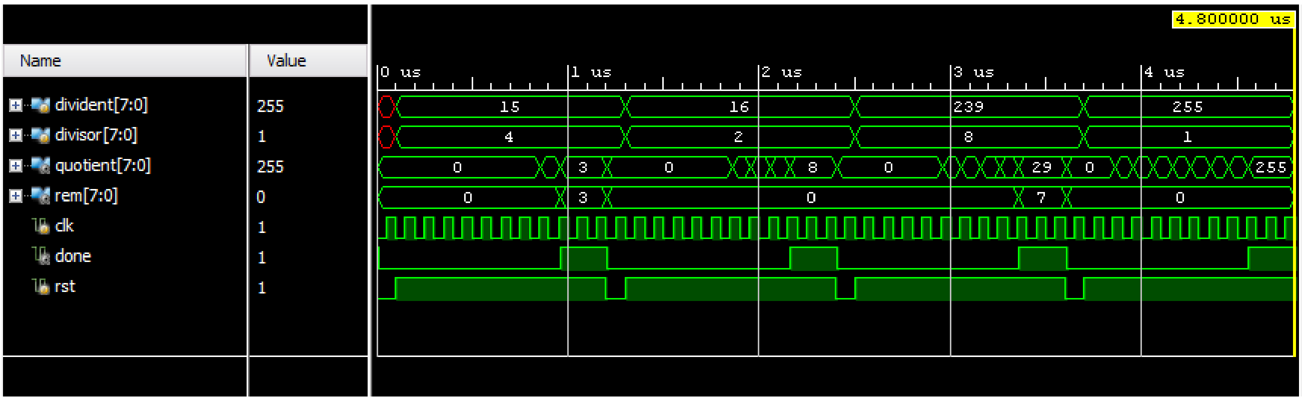

Figure 9: Post Implementation Output for Q2

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	LUT Flip Flop Pairs (63400)	Bonded IOB (210)	BUFGCTRL (32)
restoring8bit	37	46	18	37	53	35	1

Figure 10: Resource Utilization for Q2

Question 3

Design a finite state machine for 8-bit non restoring division. Write a synthesisable verilog code for the same.

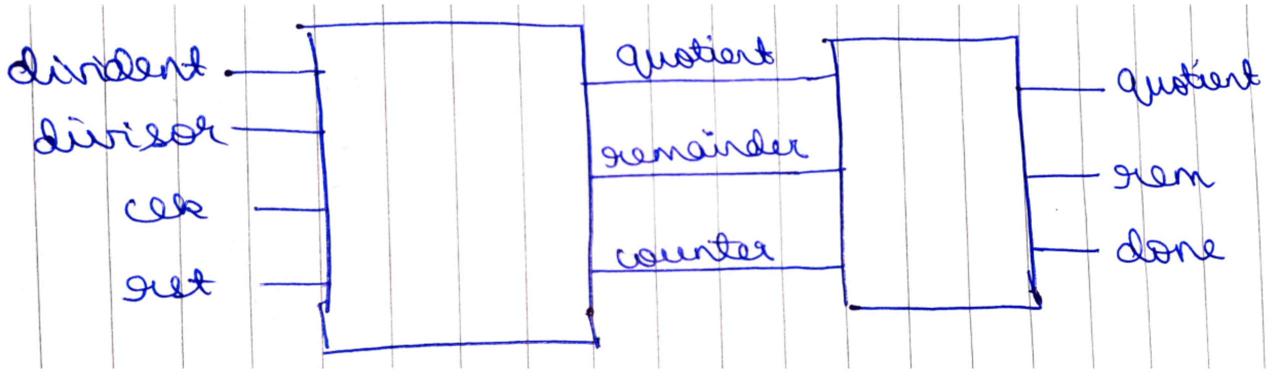

Figure 11: Block Diagram for Q3

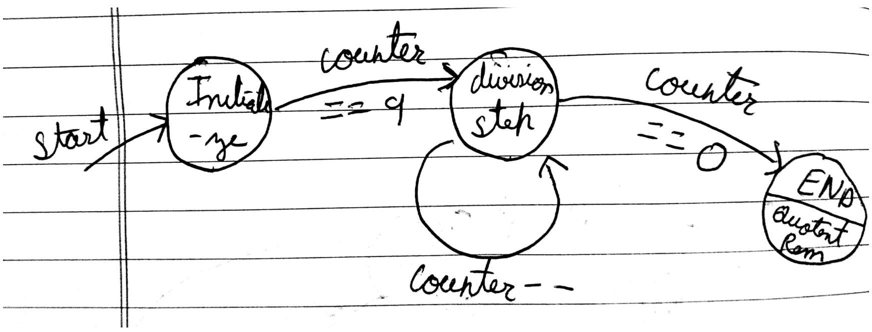

Figure 12: FSM for Q3

Code 5: Verilog Code for Q3

```
`timescale 1ns / 1ps
//Nonrestoring 8 bit divison
module nonrestoring8bit(
    input [7:0] divident,
    input [7:0] divisor,
    output reg [7:0] quotient,
    output reg [7:0] rem,
    input clk,
    output reg done,            //done becomes 1 when operation is complete
    input rst                   //active low reset
    );
    reg msbpartial;
    reg [16:0] neg_divisor;
    reg [3:0] counter;          //counter to keep track of the no of clock cycles
        elapsed
    reg [15:0] remainder;
    always @(posedge clk, negedge rst)
    begin
    if (rst==0)
    begin
        done <= 0;
        counter <= 9;   // 8 clocks for ouput and 1 clock to load
        quotient <= 0;
        msbpartial <= 0;
        rem <= 0;
    end
    else
    begin
        if(counter == 9)        //to load the data initally
        begin
            remainder <= divident;
            neg_divisor <= {(~divisor + 1),8'b00000000};
            counter <= counter -1;
        end
        else if(counter>0)
            begin
            {msbpartial,remainder} = ({msbpartial,remainder}<<1);
            if(msbpartial == 0)
                {msbpartial,remainder} = ({msbpartial,remainder}) + neg_divisor;
            else
                {msbpartial,remainder} = ({msbpartial,remainder}) - neg_divisor;
            quotient = (quotient<<1);
            if(msbpartial==1)
                quotient = quotient;
            else
                quotient = (quotient + 1'b1);
            counter = counter - 1;
            if(counter==0)
                begin
                    done=1;
                    remainder = (msbpartial) ? (remainder - neg_divisor):
                        remainder;
                    rem = remainder[15:8];
                end
            else
                done=0;
            end
         else   //hold the results
             begin
             quotient = quotient;
```

```
                rem = rem;
            end
        end
    end
endmodule
```

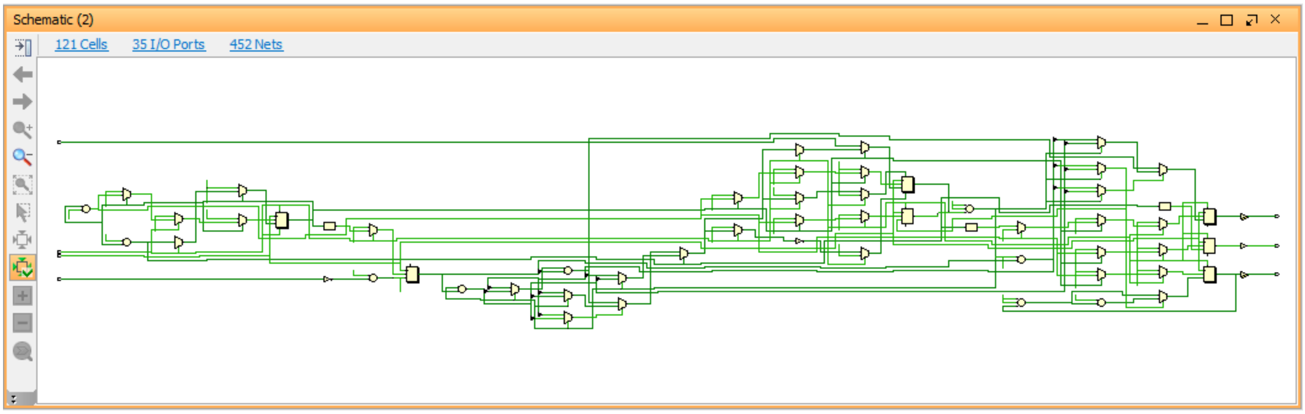

Figure 13: Elaborated Design for Q3

Code 6: Testbench for Q3

```
`timescale 1ns / 1ps
//Testbench for nonrestoring divison
module q3testbench();
reg [7:0] divident;
reg [7:0] divisor;
wire [7:0] quotient;
wire [7:0] rem;
reg clk;
wire done;
reg rst;

nonrestoring8bit res(.divident(divident), .divisor(divisor), .quotient(quotient),
    .rem(rem), .clk(clk), .done(done), .rst(rst));

initial begin
clk = 0;
rst=0;#100;
rst=1;
divisor = 4;
divident = 15;#1100;

rst=0;#100;
rst=1;
divisor = 2;
divident = 16;#1100;

rst=0;#100;
rst=1;
divisor = 8;
divident = 239;#1100;

rst=0;#100;
rst=1;
divisor = 1;
divident = 255;#1100;
$finish;
```

```
end

always begin
clk = #50 ~clk;
end
endmodule
```

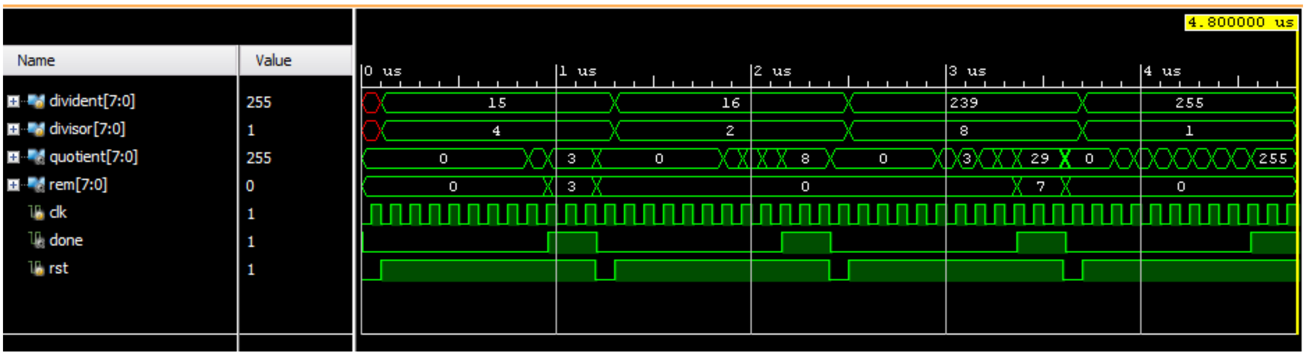

Figure 14: Post Implementation Output for Q3

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	LUT Flip Flop Pairs (63400)	Bonded IOB (210)	BUFGCTRL (32)
nonrestoring8bit	76	46	29	76	86	35	1

Figure 15: Resource Utilization for Q3

Question 4

Design a finite-state machine that illustrates the operation of a digital watch with two function buttons. Each successive push of button 1 causes the watch to change from displaying the time, to setting the hours, to setting the minutes and back to displaying the time again and so on. Button 2 allows the user to increment either the hours or the minutes when the watch is in the appropriate state.

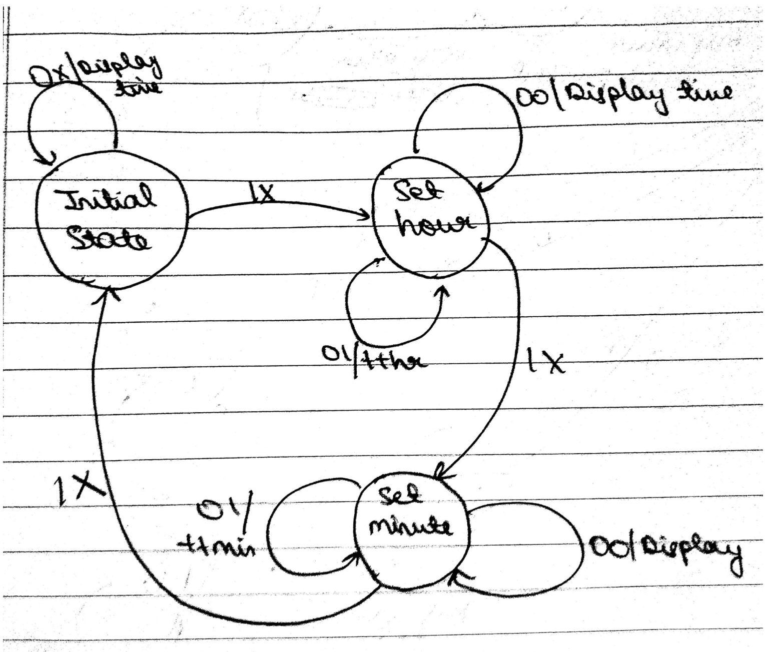

Figure 16: FSM for Q4

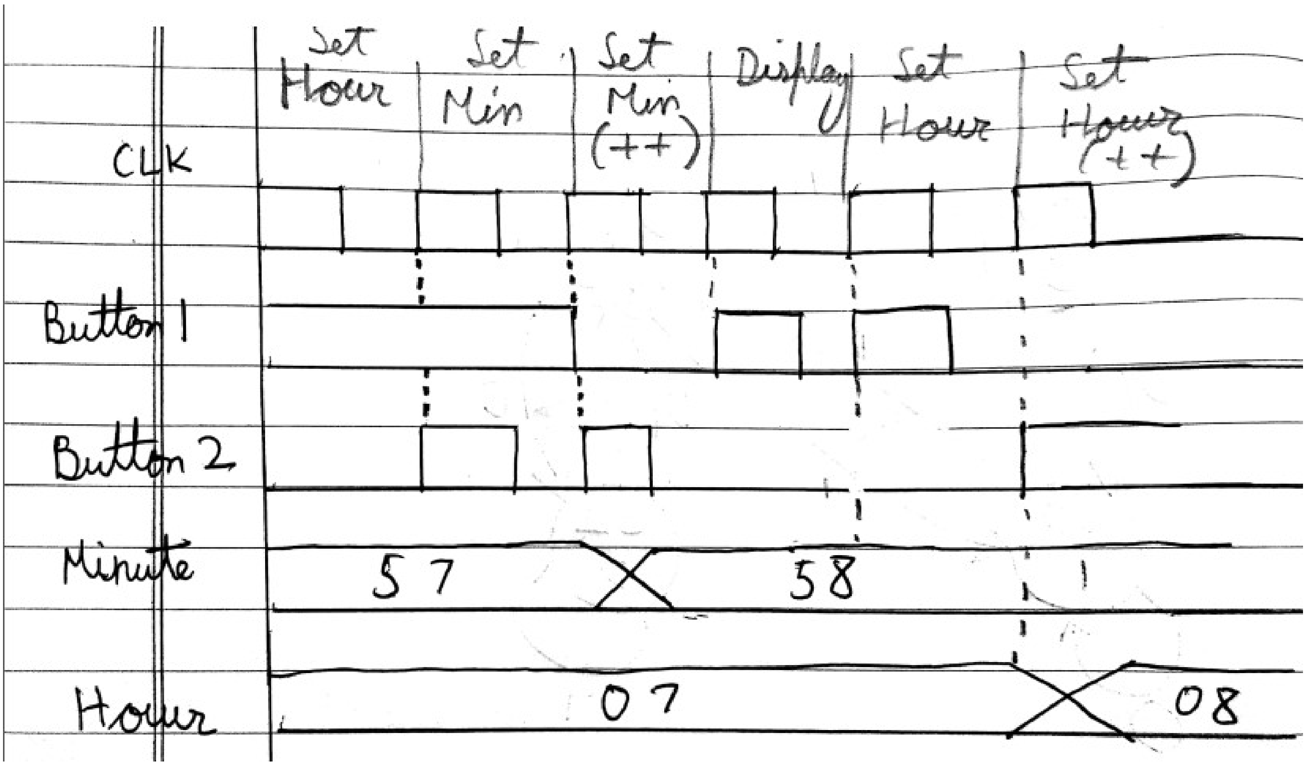

Figure 17: Timing Diagram for Q4

Question 5

Design an FSM for a digital hardware circuit used to control an automatic teller machine that performs three tasks: tells the user the balance of his bank account, permits the user to withdraw an amount of money not greater than the balance on his account, and permits the user to deposit money into his account.

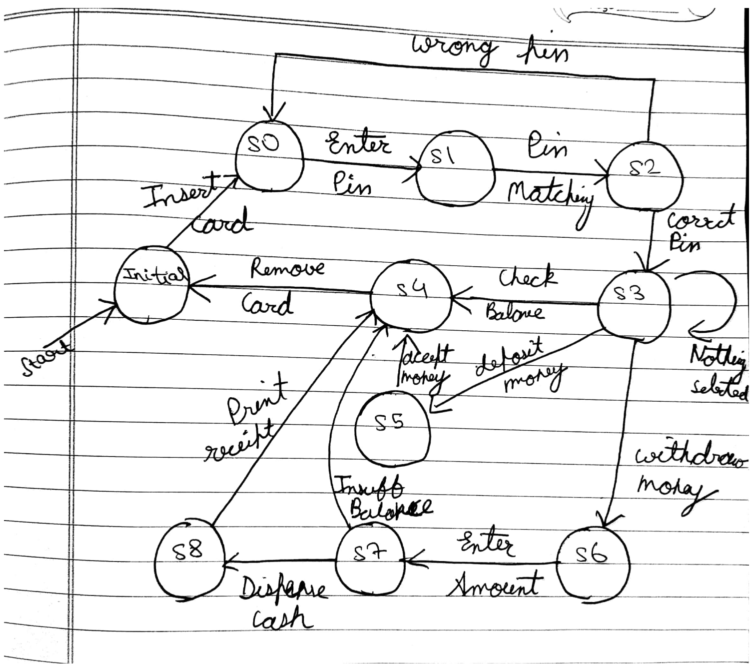

Figure 18: FSM for Q5

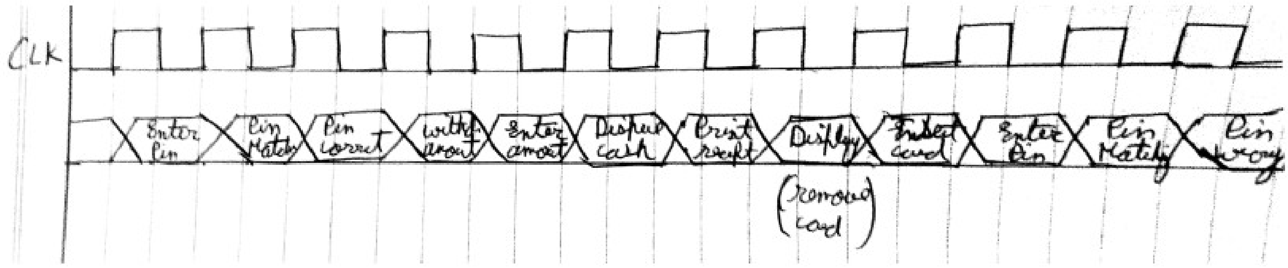

Figure 19: Timing Diagram for Q5

Question 6

The overall objective is to create a line tracking robot. The system has two digital inputs and two digital outputs. You can simulate the system with two switches and two LEDs, or build a robot with two DC motors and two optical reflectance sensors. Both sensor inputs will be on if the machine is completely on the line. One sensor input will be on and the other off if the machine is just going off the track. If the machine is totally off the line, then both sensor inputs will be off. Implement the controller using a finite state machine. Choose a Moore or Mealy format as appropriate.

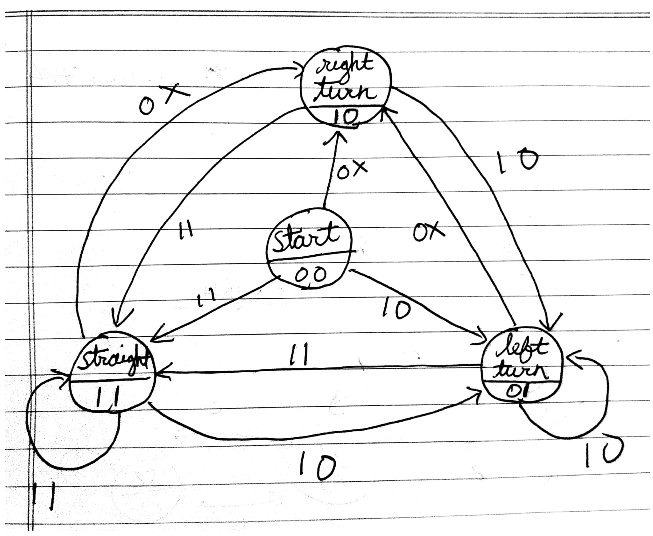

Figure 20: FSM for Q6

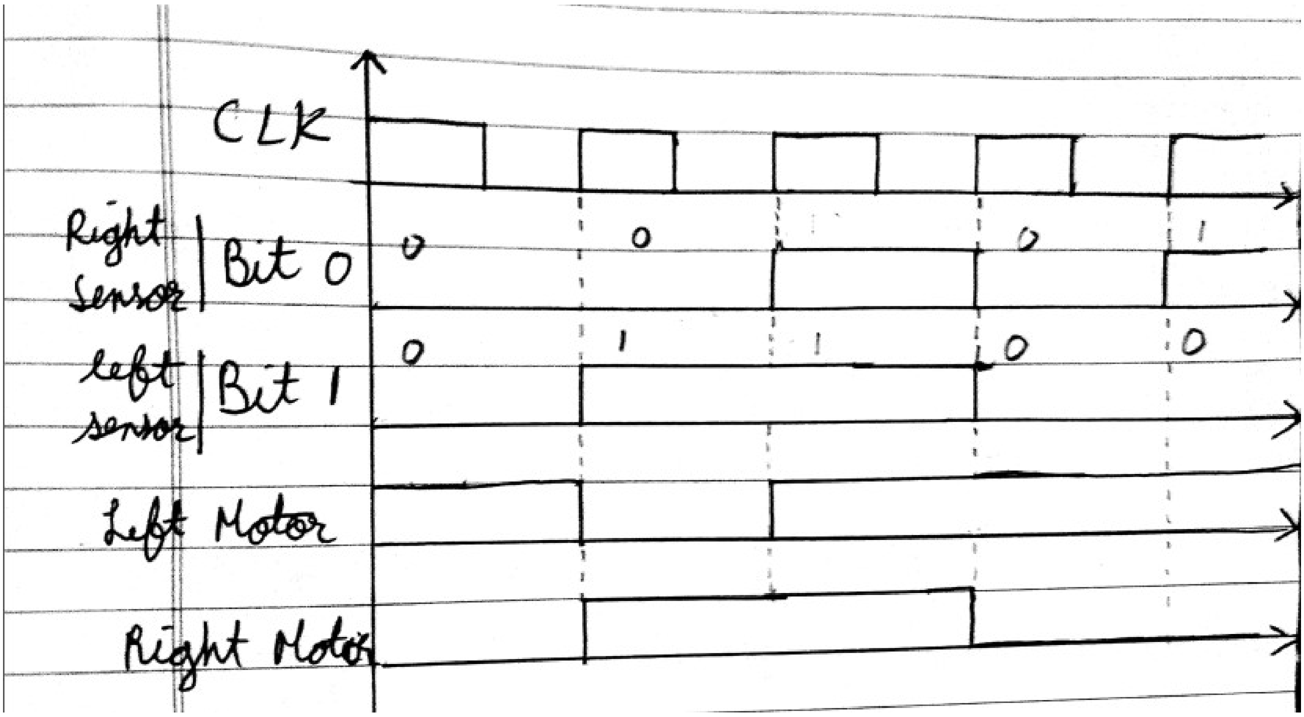

Figure 21: Timing Diagram for Q6

Question 7

Consider a FSM that will receive input from a keypad and lock/unlock a door:
1. The keypad has digits 0...9.
2. On power up, the door is locked.
3. As soon as the sequence 1, 1, 3, 8 is entered, the door must be unlocked.
4. Once in a not locked state, when 0 is entered, the door is immediately locked and the FSM returns to a state in which it is waiting for a code.
5. As soon as the sequence 1, 1, 3, 0 is entered, the FSM sounds an alarm and the door is permanently locked.
6. Sequences other than the two listed above are ignored.
7. The events are: 0, 1, ...9.
8. The actions are: LOCK, UNLOCK, ALARM and none (X).
Draw the diagram that describes the behaviour of this FSM.

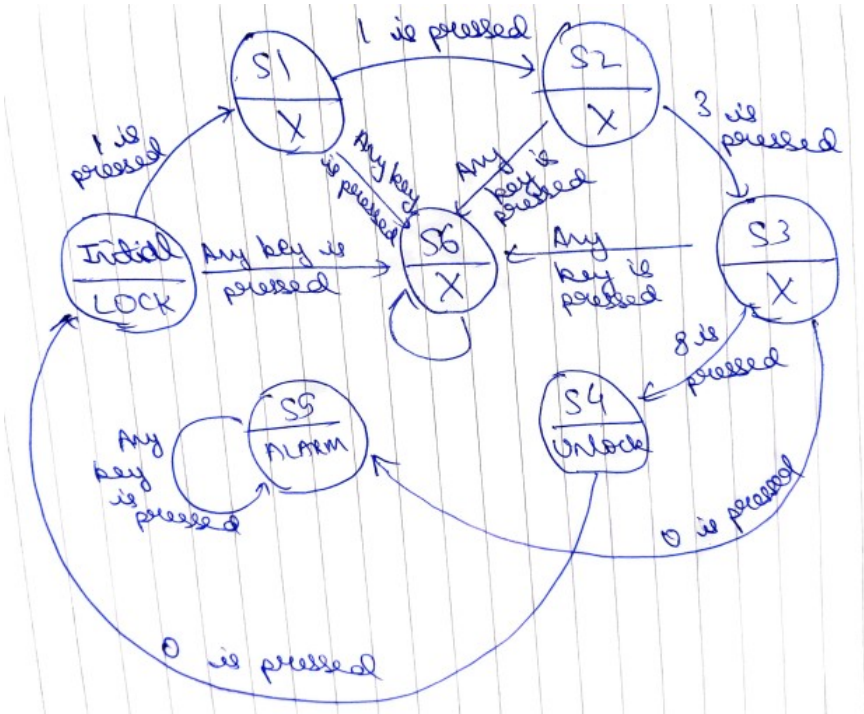

Figure 22: FSM for Q7

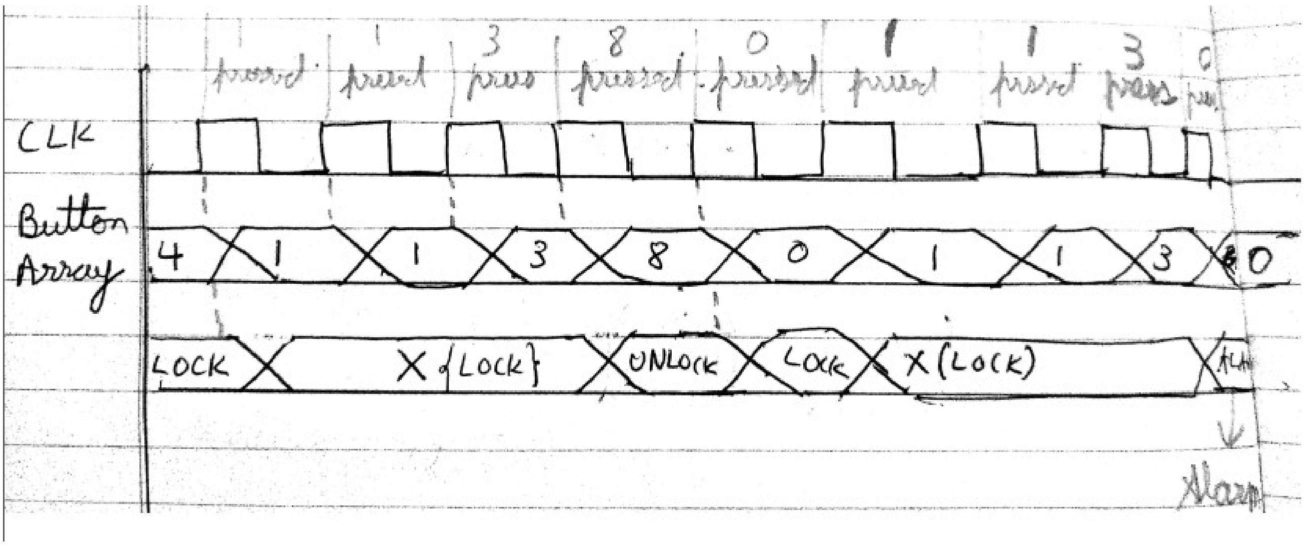

Figure 23: Timing Diagram for Q7