CS223 – Digital Design

Ahmet Faruk Ulutaş

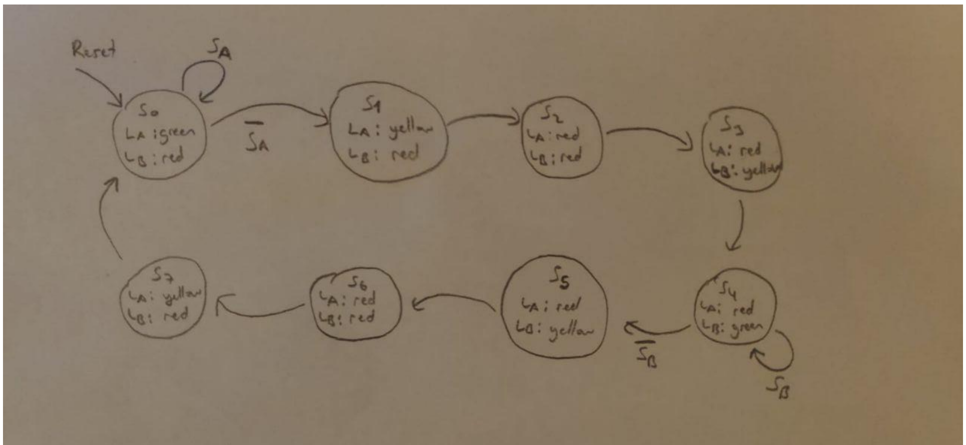
21803717

Section 1

Lab 4

Preliminary Work

A)



Moore State Machine Transition Diagram

State encoding

STATE	ENCODING
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

State Transition Table

Current State S	Inputs S_A	S_B	Next State S'
S_0	1	X	S_0
S_0	0	X	S_1
S_1	X	X	S_2
S_2	X	X	S_3
S_3	X	X	S_4
S_4	X	1	S_4
S_4	X	0	S_5
S_5	X	X	S_6
S_6	X	X	S_7
S_7	X	X	S_0

Output Table

Current State S_2	S_1	S_0	Outputs L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	0	1	0
0	0	1	0	1	1	0
0	1	0	1	0	1	0
0	1	1	1	0	0	1
1	0	0	1	0	0	0
1	0	1	1	0	0	1
1	1	0	1	0	1	0
1	1	1	0	1	1	0

Output encoding

Output	Encoding $L_{1:0}$
Green	00
Yellow	01
Red	10

State Transition Table with Encoding

Current State S			Inputs		Next State S'		
S_2	S_1	S_0	S_A	$\bar{S}_B$	S_2'	S_1'	S_0'
0	0	0	0	X	0	0	1
0	0	0	1	X	0	0	0
0	0	1	X	X	0	1	0
0	1	0	X	X	0	1	1
0	1	1	X	X	1	0	0
1	0	0	X	0	1	0	1
1	0	0	X	1	1	0	0
1	0	1	X	X	1	1	0
1	1	0	X	X	1	1	1
1	1	1	X	X	0	0	0

$$S_2' = \bar{S}_2 S_1 S_0 + S_2 \bar{S}_1 + S_2 \bar{S}_0$$

$$S_1' = \bar{S}_2 \bar{S}_1 S_0 + \bar{S}_2 S_1 \bar{S}_0 + S_2 \bar{S}_1 S_0 + S_2 S_1 \bar{S}_0$$

$$S_0' = \bar{S}_2 \bar{S}_1 \bar{S}_0 \bar{S}_A + S_1 \bar{S}_0 + S_2 \bar{S}_1 S_0 \bar{S}_B$$

Output Table with encoding

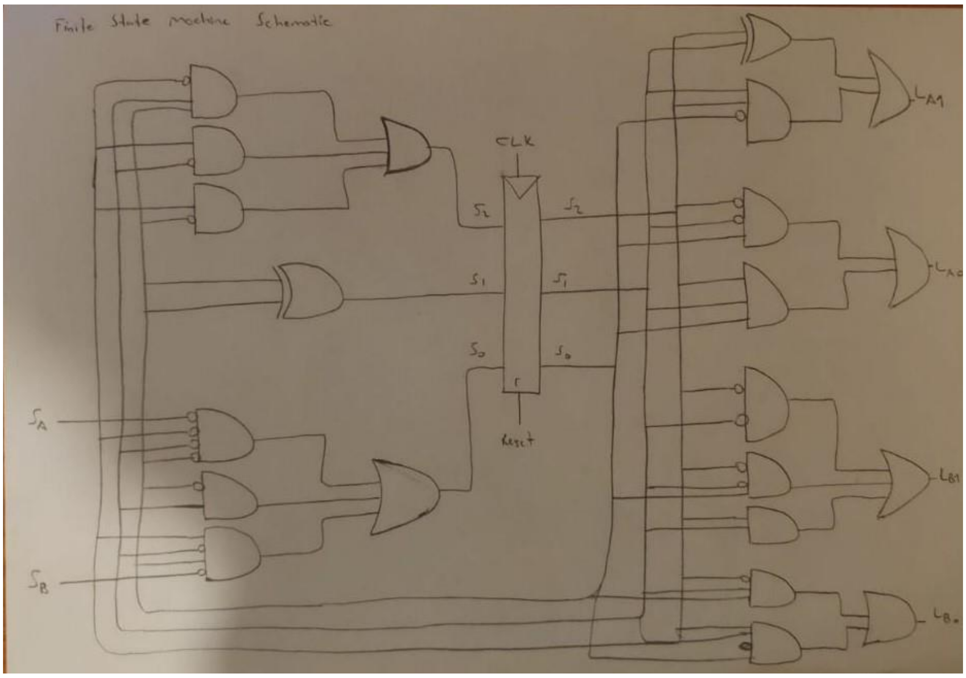
Current State S			Outputs			
S_2	S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	0	1	0
0	0	1	0	1	1	0
0	1	0	1	0	1	0
0	1	1	1	0	0	1
1	0	0	1	0	0	0
1	0	1	1	0	0	1
1	1	0	1	0	1	0
1	1	1	0	1	1	0

$$L_{A1} = \bar{S}_2 S_1 + S_2 \bar{S}_1 + S_2 S_1 \bar{S}_0$$

$$L_{A0} = \bar{S}_2 \bar{S}_1 S_0 + S_2 S_1 S_0$$

$$L_{B1} = \bar{S}_2 \bar{S}_1 + \bar{S}_2 S_1 \bar{S}_0 + S_2 S_1$$

$$L_{B0} = \bar{S}_2 S_1 S_0 + S_2 \bar{S}_1 S_0$$



B) We need as many flip flops as state variables. There are 3 variables here, so we need 3 flops.

C) To get 3 seconds from 100 mHz, we first know that the mhz is 10 ns. We write the 3 seconds in ns. This makes 3,000,000,000 ns. If we then divide this into 10 ns pieces, it makes 300,000,000 ns.

```
module three_second_clock( input clock, reset, output clock_);

    integer maxRisingEdgeNumber <= 300000000;

    int counter <= 0;


    intial begin

        clock_ <= 0;

    end


    always @ ( posedge(clock) )

    begin

        counter <= counter + 1;

        if ( counter < maxRisingEdgeNumber )

            clock_ = 0;

        else

            begin

                counter <= 0;

                clock_ <= 1;

            end


    end

endmodule
```

D)

```
module trafficLightSystem ( input logic clock, reset, SA, SB, output logic [1:0]LA, LB);
typedef enum logic [2:0] {S0, S1, S2, S3, S4, S5, S6, S7} stateType;
typedef enum logic [1:0] {green, yellow, red} trafficLight;

stateType [2:0] currentState, nextState;
trafficLight [1:0] LA_, LB_;

always_ff @ (posedge clock, posedge reset)
        if ( !reset)
                currentState <= nextState;
        else
                currentState <= S0;

always_comb
        case ( currentState)
                S0: begin
                        LA_ = green;
                        LB_ = red;
                        if ( !SA)
                                nextState = S1;
                         else
                                nextState = S0;
                end
                S1: begin
                        LA_ = yellow;
                        LB_ = red;
                        nextState = S2;
                end
```

```
S2: begin
    LA_ = red;
    LB_ = red;
    nextState = S3;
end
S3: begin
    LA_ = red;
    LB_ = yellow;
    nextState = S4;
end
S4: begin
    LA_ = red;
    LB_ = green;
    if ( !TA) nextState = S5;
    else nextState = S4;
end
S5: begin
    LA_ = red;
    LB_ = yellow;
    nextState = S6;
end
S6: begin
    LA_ = red;
    LB_ = red;
    nextState = S7;
end
S7: begin
    LA_ = yellow;
    LB_ = red;
    nextState = S0;
end
```

```
                        default: nextState = S0;
 endcase

 assign LA = LA_;
 assign LB = LB_;
endmodule
```

Testbench

```
module testbench_trafficLightSystem();
            logic clock, reset, SA, SB; logic [1:0] LA, LB;
            trafficLights dut(clock, reset, SA, SB, LA, LB);
            initial begin
                        clock = 0; SA = 1; SB = 1; reset = 1; #20; reset = 0; #20;
                        reset = 0; SA = 0; SB = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        reset = 1; #10; reset = 0; #10;

                        reset = 0; SA = 0; SB = 1; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
                        clock = 0; #1; clock = 1; #10;
```

```
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        reset = 1; #10; reset = 0; #10;

        reset = 0; SA = 1; SB = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        reset = 1; #10; reset = 0; #10;

    reset = 0; SA = 1; SB = 1; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
        clock = 0; #1; clock = 1; #10;
    end
endmodule
```

E)

```
module trafficLightSystemTop( input logic clock, reset, SA, SB, output logic [2:0] LA, LB);
	logic clock_;
	typedef enum logic [1:0] {green, yellow, red} light;
	light [1:0] LA_, LB_;

	clockChanger click( clock, clock_);
	trafficLightsSystem lights( clock_, reset, SA, SB, LA_, LB_);

	always_comb
		case (LA_)
			green: LA = 3'b011;
			yellow: LA = 3'b001;
			red: LA = 3'b111;
			default: LA = 3'b011;
	endcase

	always_comb
		case (LB_)
			green: LB = 3'b011;
			yellow: LB = 3'b001;
			red: LB = 3'b111;
			default: LB = 3'b111;
	endcase
endmodule
```