CS 223 Digital Design

Bilkent University

Fall 2024-25

Laboratory Assignment 5

Traffic Light System and Gray Code Counter

Nabeeha Khan

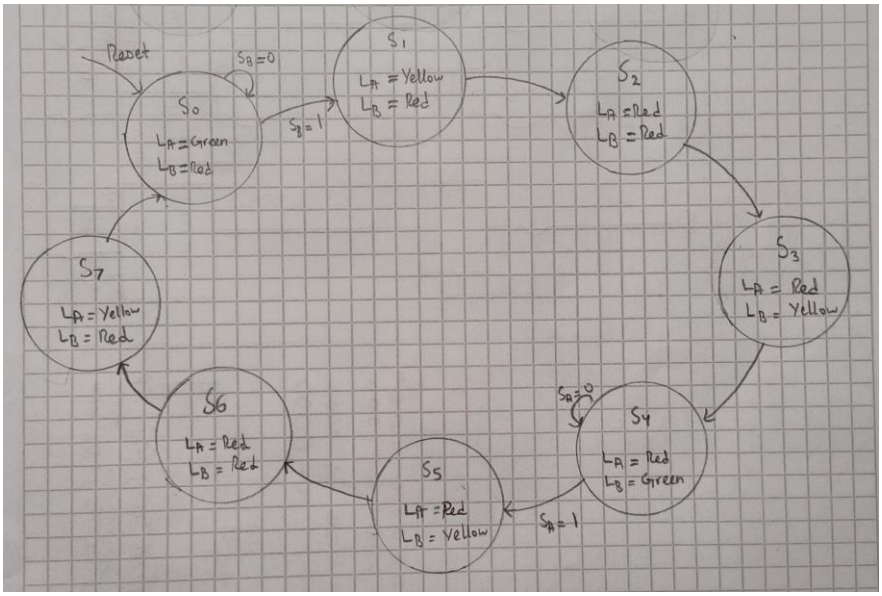
22301304

Section: 03

Date: 9th December 2024

Traffic Light System:

1. State Transition Diagram:



2. State Encodings:

Current State S	**Encodings $S_{2:0}$**
S_0	000
S_1	001
S_2	010
S_3	011
S_4	100
S_5	101
S_6	110
S_7	111

3. State Transition Table:

Current State			**Inputs**		**Next State**		
S_2	**S_1**	**S_0**	**S_A**	**S_B**	**S'_2**	**S'_1**	**S'_0**
0	0	0	X	0	0	0	0
0	0	0	X	1	0	0	1
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	0	X	1	0	0
1	0	0	1	X	1	0	1
1	0	1	X	X	1	1	0

1	1	0	X	X	1	1	1
1	1	1	X	X	0	0	0

4. Output Encodings:

Output	Encodings $L_{2:0}$
Red	111
Yellow	001
Green	011

5. Output Table:

Current State			Output					
S_2	S1	S_0	L_{A2}	L_{A1}	L_{A0}	L_{B2}	L_{B1}	L_{B0}
0	0	0	0	1	1	1	1	1
0	0	1	0	0	1	1	1	1
0	1	0	1	1	1	1	1	1
0	1	1	1	1	1	0	0	1
1	0	0	1	1	1	0	1	1
1	0	1	1	1	1	0	0	1
1	1	0	1	1	1	1	1	1
1	1	1	0	0	1	1	1	1

6. Next State and Output Equations:

Next State:

$S'_2 = S_2S_0` + S_2`S_1S_0 + S_2S_1`$

$S'_1 = S_1 \oplus S_0$

$S'_0 = S_1S_0` + S_2`S_1`S_0`S_B + S_2S_1`S_0`S_A$

Output:

$L_{A2} = S_2 \oplus S_1 + S_2S_0`$

$L_{A1} = S_0` + S_2 \oplus S_1$

$L_{A0} = 1$

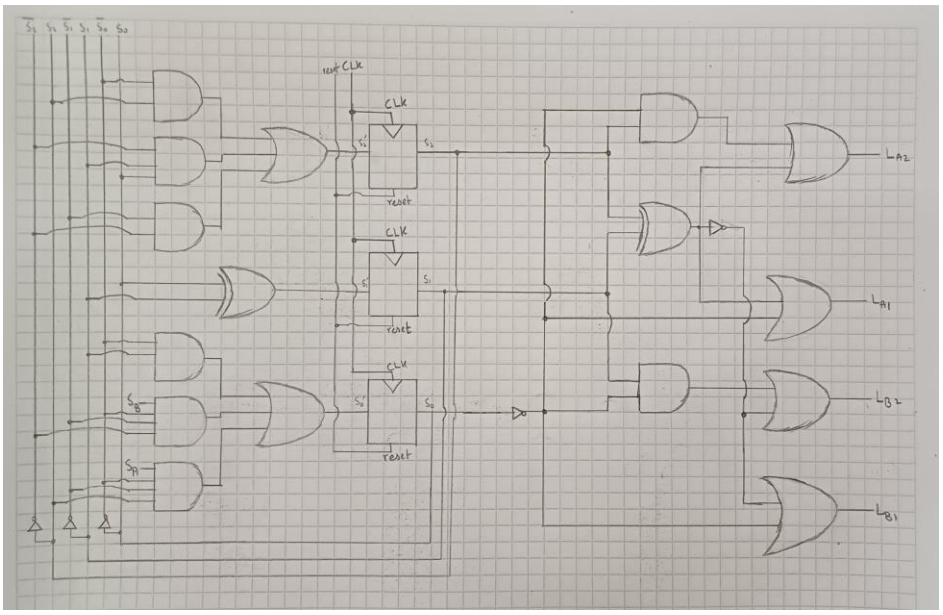
$L_{B2} = S_1S_0` + (S_2 \oplus S_1)`$

$L_{B1} = S_0` + (S_2 \oplus S_1)`$

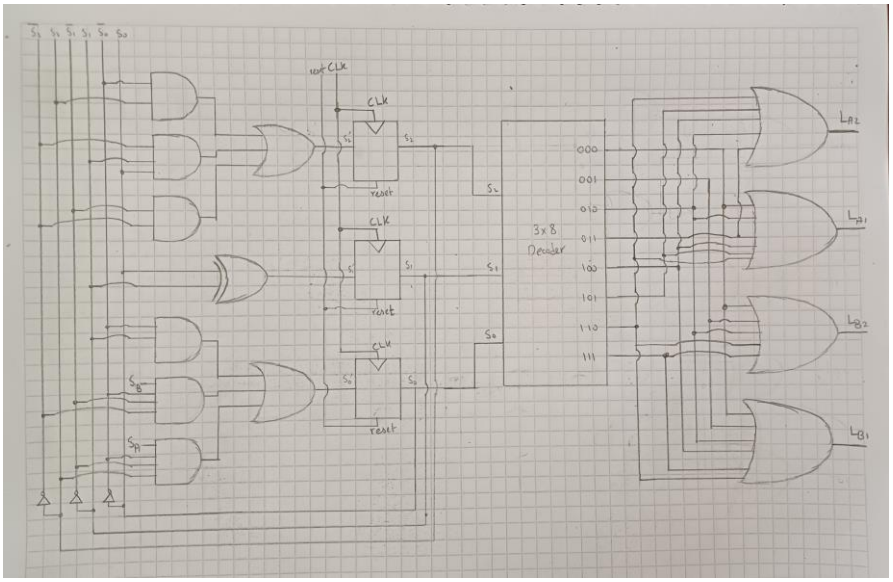
$L_{B0} = 1$

7. FSM Schematic:

We would need **3 Flip-flops** for this schematic.



8. FSM Schematic using Decoders:



9. SystemVerilog Module and Testbench:

//Module for clock divider

module clock_cycle(input logic clk, output logic clk_div);

```
localparam constantNumber = 150000000;
logic [31:0] count;

always @(posedge(clk))
begin
if (count == constantNumber - 1)
      count <= 32'b0;
   else
      count <= count + 1;
end

always @ (posedge(clk))
begin
   if (count == constantNumber - 1)
      clk_div <= ~clk_div;
   else
      clk_div <= clk_div;
end


endmodule


//Module for Traffic Lights System
module TrafficLights(input logic clk, reset, sa, sb,
                     output logic [2:0] la, lb);
   logic clkslow;


   clock_cycle clksl(clk, clkslow);


   typedef enum logic [2:0] {s0, s1, s2, s3, s4, s5, s6, s7} Statetype;
   Statetype [1:0] state, nextstate;


   parameter g = 3'b011;
   parameter y = 3'b001;
   parameter r = 3'b111;


   always_ff @(posedge clkslow, posedge reset)
      if(reset) state <= s0;
```

```
        else state <= nextstate;

    always_comb
    case(state)
        s0: if(sb) nextstate = s1;
            else   nextstate = s0;
        s1:        nextstate = s2;
        s2:        nextstate = s3;
        s3:        nextstate = s4;
        s4: if(sa) nextstate = s5;
            else   nextstate = s4;
        s5:        nextstate = s6;
        s6:        nextstate = s7;
        s7:        nextstate = s0;
        default:   nextstate = s0;
    endcase

    always_comb
    case(state)
        s0:      {la, lb} = {g, r};
        s1:      {la, lb} = {y, r};
        s2:      {la, lb} = {r, r};
        s3:      {la, lb} = {r, y};
        s4:      {la, lb} = {r, g};
        s5:      {la, lb} = {r, y};
        s6:      {la, lb} = {r, r};
        s7:      {la, lb} = {y, r};
        default: {la, lb} = {g, r};
    endcase
endmodule
```

```
//Testbench
module TrafficLight_TB();
    logic clk, reset, sa, sb;
    logic [2:0] la;
    logic [2:0] lb;

    TrafficLights dut(clk, reset, sa, sb, la, lb);

    always
        begin
            clk <= 1; #5;
            clk <= 0; #5;
        end

    initial
        begin
            reset <= 1; #20;
            reset <= 0; #20;
            sa <= 1; sb <= 0; #100;
            sa <= 0; sb <= 0; #100;
            sa <= 0; sb <= 1; #100;
            sa <= 1; sb <= 1; #100;
        end
endmodule
```

Grey Code:

1. Algorithm for n-bit Grey Code:

To generate a n-bit grey code recursively, we have two steps: base case and recursive step. For base case which is set at n = 1, we return an array of strings “0” and “1”. In recursive step, which is for n > 1, we call the function again for n – 1 which produces an array of n-1 bit grey code as strings, then we append this array first with “0” in the same order they appear in array to get array1 and then append with “1” in reverse order they appear in array to get array2. Then we return array1 + array2, which is the array of n bit grey code. The pseudo code is given below:

```
GreyCode(n)

if n = 1: return [“0”, “1]

else: prevGrey = GreyCode(n – 1)

                array1 = [“0” + prevGrey[n-2: 0] ]

                array2 = [“1” + prevGrey[0: n-2] ]

    return array1+array2
```

2. SystemVerilog Module for 4-bit Grey Code:

```
//Module for clock divider

module clock_cycle1Hz (input logic clk, output logic clk_div );

localparam constantNumber = 50000000;
logic [31:0] count;

always @(posedge clk)
begin
   if (count == constantNumber - 1)
      count <= 32'b0;
   else
      count <= count + 1;
end

always @(posedge clk)
begin
   if (count == constantNumber - 1)
      clk_div <= ~clk_div;
   else
      clk_div <= clk_div;
end


endmodule

//Module for upcounter register
```

```
module upcountRegister(
    input logic clk, reset, load_en, enable,
    input logic [3:0] load,
    output logic [3:0] q
    );

    always_ff @(posedge clk) begin
        if(reset)
            q <= 4'b0;
        else if(load_en)
            q <= load;
        else if(enable)
            q <= q + 1;
    end
endmodule

//Module for binary to grey code converter
module binaryToGrey(
    input logic [3:0] binary,
    output logic [3:0] grey
    );

    always_comb begin
        grey[3] = binary[3];
        grey[2] = binary[3] ^ binary[2];
        grey[1] = binary[2] ^ binary[1];
        grey[0] = binary[1] ^ binary[0];
    end
endmodule
```

```
//Module for 4-bit Grey code up counter

module greyCode4bit( input logic clk, reset, load_en, enable, input logic [3:0] load, output logic [3:0] grey );

logic clk_slow;
clock_cycle1Hz clockdiv(clk, clk_slow);

logic [3:0] binary;
upcountRegister register(clk_slow, reset, load_en, enable, load, binary);

binaryToGrey converter(binary, grey);

endmodule


//Testbench

module grayCode4bit_TB();

logic clk, reset, load_en, enable;
logic [3:0] load;
logic [3:0] grey;

grayCode4bit uut (clk, reset, load_en, enable, load, grey);

initial clk = 0;
always #5 clk = ~clk;

initial begin
   reset = 1; enable = 0; load_enable = 0; load = 4'b0; #10;

   reset = 0; #10:
   enable = 1; #100;

   load = 4'b0010; load_enable = 1; #10;

   load = 4'b1111; load_enable = 1; #10;

   load = 4'b0111; load_enable = 1; #10;
   load_enable = 0; #50
end

endmodule
```

3. SystemVerilog Module for 8-bit Grey Code:

```
 //Module for clock divider

module clock_cycle1Hz (input logic clk, output logic clk_div );
```

```
localparam constantNumber = 50000000;
logic [31:0] count;

always @(posedge clk)
begin
   if (count == constantNumber - 1)
      count <= 32'b0;
   else
      count <= count + 1;
end

always @(posedge clk)
begin
   if (count == constantNumber - 1)
      clk_div <= ~clk_div;
   else
      clk_div <= clk_div;
end


endmodule

//Module for upcounter register

module upcountRegister(

   input logic clk, reset, load_en, enable,

   input logic [7:0] load,

   output logic [7:0] q

   );


   always_ff @(posedge clk) begin

      if(reset)

         q <= 8'b0;

      else if(load_en)

         q <= load;

      else if(enable)

         q <= q + 1;

   end

endmodule
```

```
//Module for binary to grey code converter
module binaryToGrey(
    input logic [7:0] binary,
    output logic [7:0] grey
    );

    always_comb begin
        grey[7] = binary[7];
        grey[6] = binary[7] ^ binary[6];
        grey[5] = binary[6] ^ binary[5];
        grey[4] = binary[5] ^ binary[4];
        grey[3] = binary[4] ^ binary[3];
        grey[2] = binary[3] ^ binary[2];
        grey[1] = binary[2] ^ binary[1];
        grey[0] = binary[1] ^ binary[0];
    end
endmodule

//Module for 8-bit Grey code up counter
module greyCode8bit( input logic clk, reset, load_en, enable,
            input logic [7:0] load,
            output logic [7:0] grey
            );
logic clk_slow;
clock_cycle1Hz clockdiv(clk, clk_slow);

logic [7:0] binary;
upcountRegister register(clk_slow, reset, load_en, enable, load, binary);

binaryToGrey converter(binary, grey);
endmodule
```

```
//Testbench

module grayCode8bit_TB();

logic clk, reset, load_en, enable;
logic [7:0] load;
logic [7:0] grey;

grayCode8bit uut (clk, reset, load_en, enable, load, grey);

initial clk = 0;
always #5 clk = ~clk;

initial begin
    reset = 1; enable = 0; load_en = 0; load = 8'b0; #10;

    reset = 0; #10:
    enable = 1; #100;

    load = 4'b00011010; load_en = 1; #10

    load = 4'b00001111; load_en = 1; #10;

   load = 4'b10000000; load_en = 1; #10;
    load_en = 0; #50
end

endmodule
```