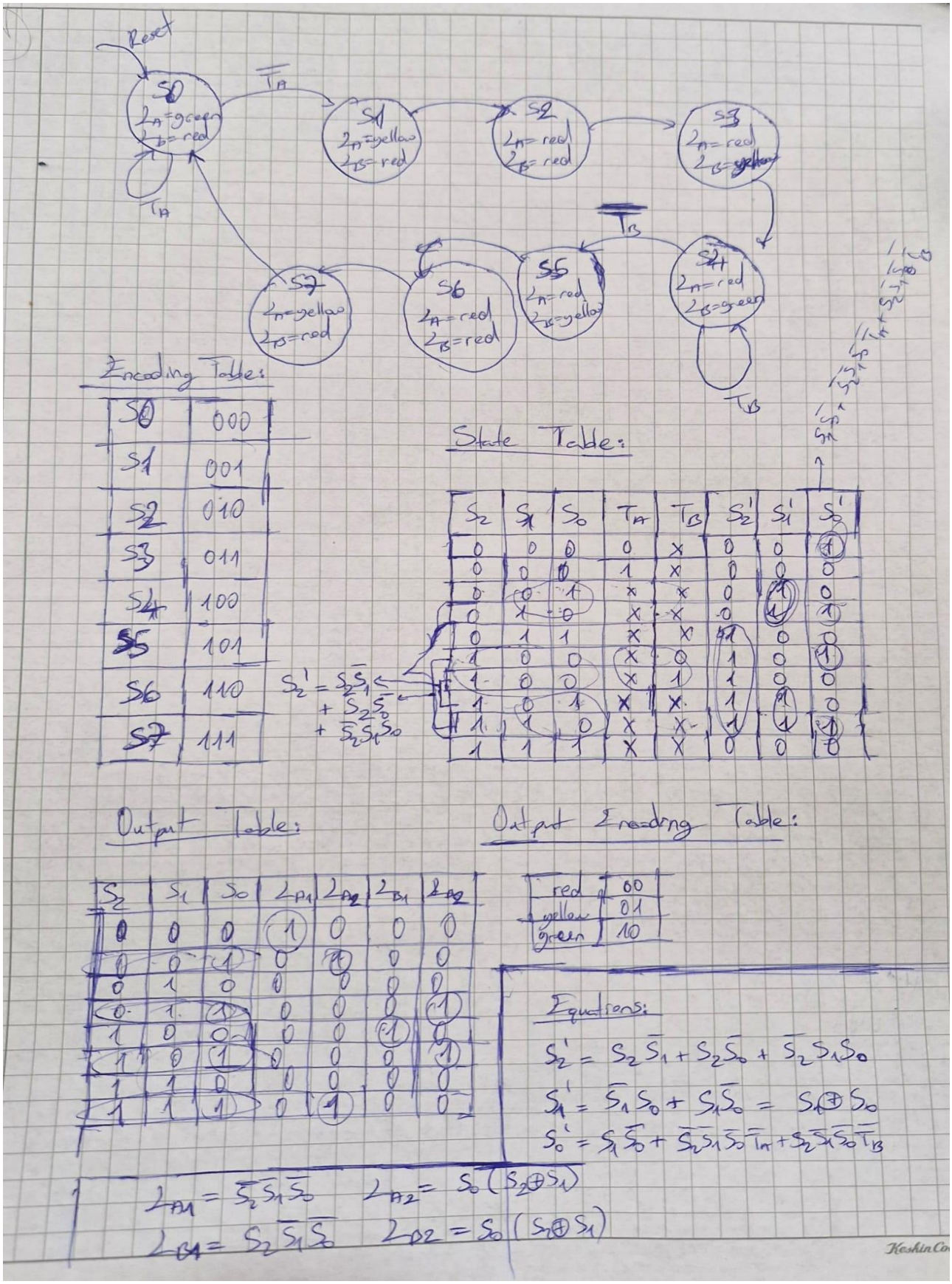
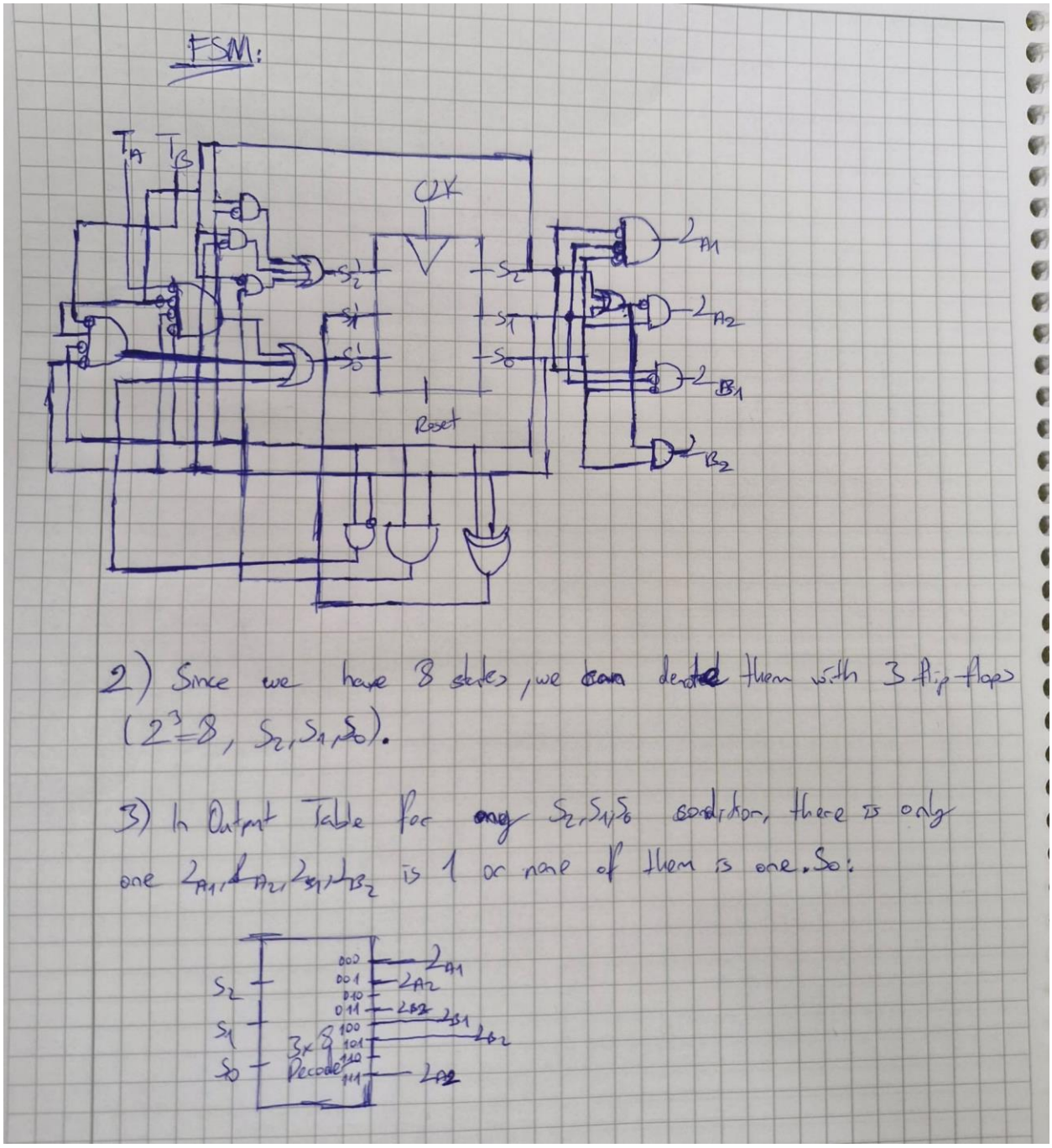
CS 223 Digital Design
Section:2
Laboratory Assignment 5

Alp Eren
Köken
22202876

1)


Reset
S0
Z_A=green
Z_B=red
S1
Z_A=yellow
Z_B=red
S2
Z_A=red
Z_B=red
S3
Z_A=red
Z_B=yellow
S4
Z_A=red
Z_B=green
S5
Z_A=red
Z_B=yellow
S6
Z_A=red
Z_B=red
S7
Z_A=yellow
Z_B=red
$\overline{T_A}$
T_A
T_B
$\overline{T_B}$
Encoding Table:
S0 000
S1 001
S2 010
S3 011
S4 100
S5 101
S6 110
S7 111
State Table:
S_2 S_1 S_0 T_A T_B S_2' S_1' S_0'
$S_2' = S_2\bar{S}_1 + S_2\bar{S}_0 + \bar{S}_2 S_1 S_0$
Output Table:
S_2 S_1 S_0 Z_{A1} Z_{A2} Z_{B1} Z_{B2}
Output Encoding Table:
red 00
yellow 01
green 10
Equations:
$S_2' = S_2\bar{S}_1 + S_2\bar{S}_0 + \bar{S}_2 S_1 S_0$
$S_1' = \bar{S}_1 S_0 + S_1\bar{S}_0 = S_1 \oplus S_0$
$S_0' = S_1\bar{S}_0 + \bar{S}_2\bar{S}_1 S_0 \bar{T}_A + S_2\bar{S}_1\bar{S}_0\bar{T}_B$
$Z_{A1} = \bar{S}_2 S_1 \bar{S}_0$ $Z_{A2} = S_0(S_2 \oplus S_1)$
$Z_{B1} = S_2\bar{S}_1\bar{S}_0$ $Z_{B2} = S_0(S_2 \oplus S_1)$

2,3)



4)

```
module trafficLights(input clk,reset,Ta, Tb,output La1, La2, Lb1, Lb2);

logic divided_clk;

wire S2,S1,S0,newS2,newS1,newS0;

clock_divider c(clk,divided_clk);

dFlipFlop d1(divided_clk,reset,newS2,S2);

dFlipFlop d2(divided_clk,reset,newS1,S1);
```

```
dFlipFlop d3(divided_clk,reset,newS0,S0);


assign newS2 = (S2&S1)|(S2&S0)|(S0&S1&S2);
assign newS1 = S1^S0;
assign newS0 = (S1&S0)|(Ta&S0&S1&S2)|(Tb&S0&~S1&S2);


assign La1 = (S0&S1&S2);
assign La2 = (S0&(S2^S1));
assign Lb1 = (S0&S1&S2);
assign Lb2 = (S0&(S1^S2));
endmodule


module clock_divider(
    input logic clk,
    output logic newClk
    );

    logic [26:0] count;

    always_ff @(posedge clk)begin
    if (count<100000000 - 1)
        count <= count + 1;
    else begin
        newClk <= ~ newClk;
        count <= 0;
        end
    end
endmodule
```

```
module dFlipFlop(
    input logic clk,
    input logic reset,
    input logic D,
    output logic Q
    );

always_ff @(posedge clk)
if(reset) Q <= 0;
else Q <= D;

endmodule
```

Testbench:

```
module trafficLights_tb;

  reg clk,reset,Ta,Tb;
  wire La1, La2, Lb1, Lb2;

  trafficLights dut(clk,reset,Ta,Tb,La1,La2,Lb1,Lb2);

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

  initial begin
    reset = 1;
    Ta = 0;
    Tb = 0;
    #20 reset = 0;
    #50 Ta = 1;
```

```
    #50 Ta = 0;
    #50 Tb = 1;
    #50 Tb = 0;
    #200;

  end
endmodule
```

5)

In terms of recursion, we are gonna use n-1 bit gray code to generate n bit gray code.If we shift n-1 bit gray codes 1 bit to right and add appropriate bit to missing MSB, we will be done.Normally, we put 0 to MSB and get first half of the n bit gray codes. Then, we put 1 to MSB, but this time we must use (n-1)-bit gray codes in the reverse order.

To make these process shorter we can XOR the binary code value with its 1 bit right-shifted version. This converts the binary counter value into an equivalent Gray Code.

6)

```
module grayCode4(input clk, reset, en, load,input logic [3:0] pLoad,output logic [3:0] result);
    logic [3:0] code;
    always_ff @(posedge clk or posedge reset) begin
        if (reset)
            code<=4'b0000;
        else if (load && en)
            code<=pLoad;
        else if (en)
            code<=code+1;
    end
 assign result = code^(code >> 1);

endmodule
```

Testbench:

```
module grayCode4_tb;
```

```
reg clk,reset,en,load;
logic [3:0] pLoad;
logic [3:0] result;
grayCode4 dut (clk,reset,en,load,pLoad,result);

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

initial begin
    reset = 1;
    en = 0;
    load = 0;
    pLoad = 4'b0000;

    #10 reset = 0;

    #10 load = 1; pLoad = 4'b1010;
    #10 load = 0;

    #10 en = 1;
    #40 en = 0;

    #10 load = 1; pLoad = 4'b1100;
    #10 load = 0;

    #10 en = 1;
    #40 en = 0;

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

```
        #20;
    end
endmodule
```

7)

```
module grayCode8(input clk, reset, en, load,input logic [7:0] pLoad,output logic [7:0] result);

    logic [7:0] code;

    always_ff @(posedge clk or posedge reset) begin
        if (reset)
            code<=8'b00000000;
        else if (load && en)
            code<=pLoad;
        else if (en)
            code<=code+1;
    End

    assign result = code^(code >> 1);

endmodule
```

TestBench:

```
module grayCode8_tb;

    reg clk, reset, en, load;
    logic [7:0] pLoad;
    logic [7:0] result;

    grayCode8 dut (clk, reset, en, load, pLoad, result);
```

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

initial begin
    reset = 1;
    en = 0;
    load = 0;
    pLoad = 8'b00000000;

    #10 reset = 0;

    #10 load = 1; pLoad = 8'b10101010;
    #10 load = 0;

    #10 en = 1;
    #80 en = 0;

    #10 load = 1; pLoad = 8'b11001100;
    #10 load = 0;

    #10 en = 1;
    #80 en = 0;

    #10 reset = 1;
    #10 reset = 0;

    #20;
end
```

```
endmodule
```