Tahmini Ders İçeriği
(Tentative Couse Schedule – Syllabus)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

1. Hafta: Sayısal Sinyaller/Sistemler, İkilik Tabanda Sayılar, Taban Aritmetiği, İşaretli/Eksi Sayıların Gösterimi, Sayısal Tasarım Tarihçesi

2. Hafta: İkili Mantık Aritmetiği ve Kapıları, Bool Cebiri Teorisi ve Tanımları, Bool Fonksiyonları, Kapı-Seviyesinde Yalınlaştırma, Karnough Haritası, Önemsenmeyen Durumlar, NAND, NOR, XOR

3-4. Hafta: FPGA, Birleşik (Combinational) Devreler, Aritmetik Modüller, Decoder, Encoder, Mux, Verilog HDL

5. Hafta: Ardışık (Sequential) Devreler, Mandal (Latch), Flip-Flop, Yazmaçlar (Registers)

Lab Sınavı (265/264L)

6. Hafta: Durum Makinaları, Örnek Tasarımlar, Sayaçlar (Counters)

7. Hafta: **(24-28 Ekim)** FSM Örnekleri, RTL (Register Transfer Level) ASMD (Algorithmic State Machine and Datapath) Tasarımları

8. Hafta: **(31 Ekim – 4 Kasım)** Durağan Zaman Analizi (Static Timing Analysis)

Ara Sınav (265/264) **(15 Kasım)**

9. Hafta: **(7-11 Kasım)** Bellekler, FPGA'da Block RAM, OpenRAM

10. Hafta: **(14-18 Kasım) ???**

11-12. Hafta: **(21-25 Kasım, 28 Kasım – 2 Aralık)** Boru hattı, FPGA ve ASIC Tasarım Akışları

Final **(Aralık)** – Proje Teslimleri **(18 Aralık)**

SAYAÇLAR (COUNTERS)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module counter_2bit
(
input clk,
input rst_n,
output [1:0] count_o
);

reg [1:0] count;

always @(posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0) begin
        count <= 2'b00;
    end else begin
        count <= count + 1;
    end
end

assign count_o = count;

endmodule
```

SAYAÇLAR (COUNTERS) – VERILOG PARAMETER

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module reg_param
#(
parameter N=32
)
(
input [N-1:0] d_i,
input rst_n,
input clk,
output [N-1:0] q_o
);

reg [N-1:0] q;

always @(posedge clk or negedge rst_n) begin
    if (rst_n == 1'b0) begin
        q <= {N{1'b0}};
    end
    else begin
        q <= d_i;
    end
end

assign q_o = q;

endmodule
```

```
module counter_param
#(
parameter N = 8
)
(
input clk,
input rst_n,
output [N-1:0] count_o
);

reg [N-1:0] count;

always @(posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0) begin
        count <= {N{1'b0}};
    end else begin
        count <= count + 1;
    end
end

assign count_o = count;

endmodule
```

YUKARI/AŞAĞI YÜKLEMELİ AKTİF SAYAÇ

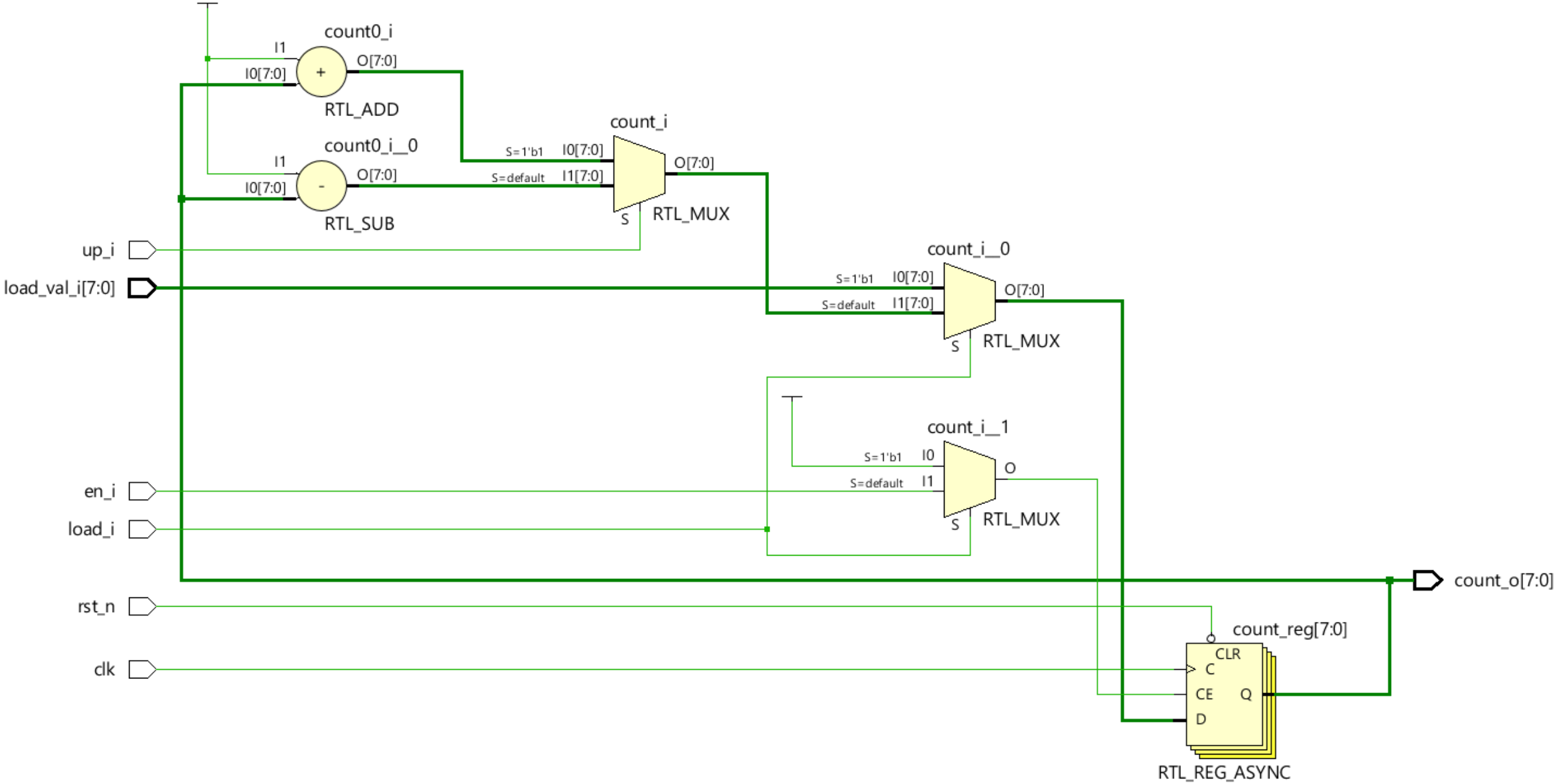
TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module counter_up_down_load
#(
parameter N = 8
)
(
input clk,
input rst_n,
input load_i,
input en_i,
input up_i,
input [N-1:0] load_val_i,
output [N-1:0] count_o
);
```

```
reg [N-1:0] count;

always @(posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0) begin
        count <= {N{1'b0}};
    end else begin
        if (load_i == 1'b1) begin
            count <= load_val_i;
        end
        else if (en_i == 1'b1) begin
            if (up_i == 1'b1) begin
                count <= count + 1;
            end
            else begin
                count <= count - 1;
            end
        end
        else begin
            count <= count;
        end
    end
end

assign count_o = count;

endmodule
```

YUKARI/AŞAĞI YÜKLEMELİ AKTİF SAYAÇ

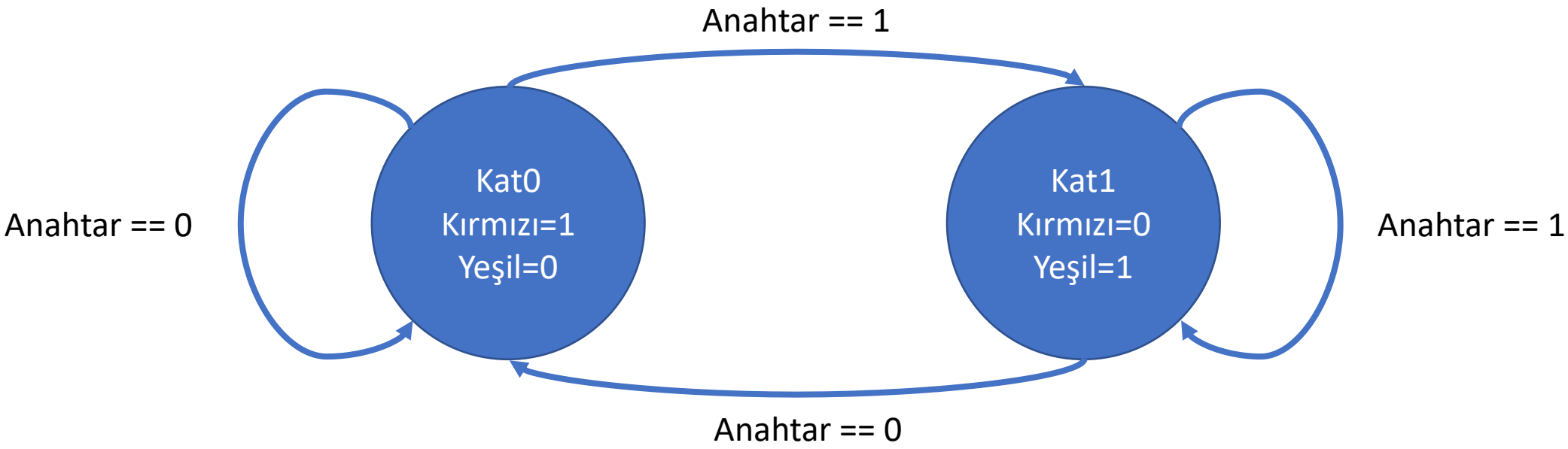
TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi



FSM ve SENKRON SIRALI DEVRE TASARIMI

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

FSM_Örnek1: 2 katlı bir binada çalışan asansör bir adet anahtar (switch) ile kontrol edilmektedir. Anahtar yukarı yönde ise asansör 1. kata, aşağı yönde ise 0. kata gitmektedir. Eğer anahtarın konumu, kat ile uyumlu ise asansör sabit kalmaktadır. 2 adet LED ile de asansörün hangi katta olduğu belirtilmektedir. 0. katta iken kırmızı led yanıyor ve yeşil led sönüyor, 1. katta iken kırmızı led sönüyor ve yeşil led yanıyor.



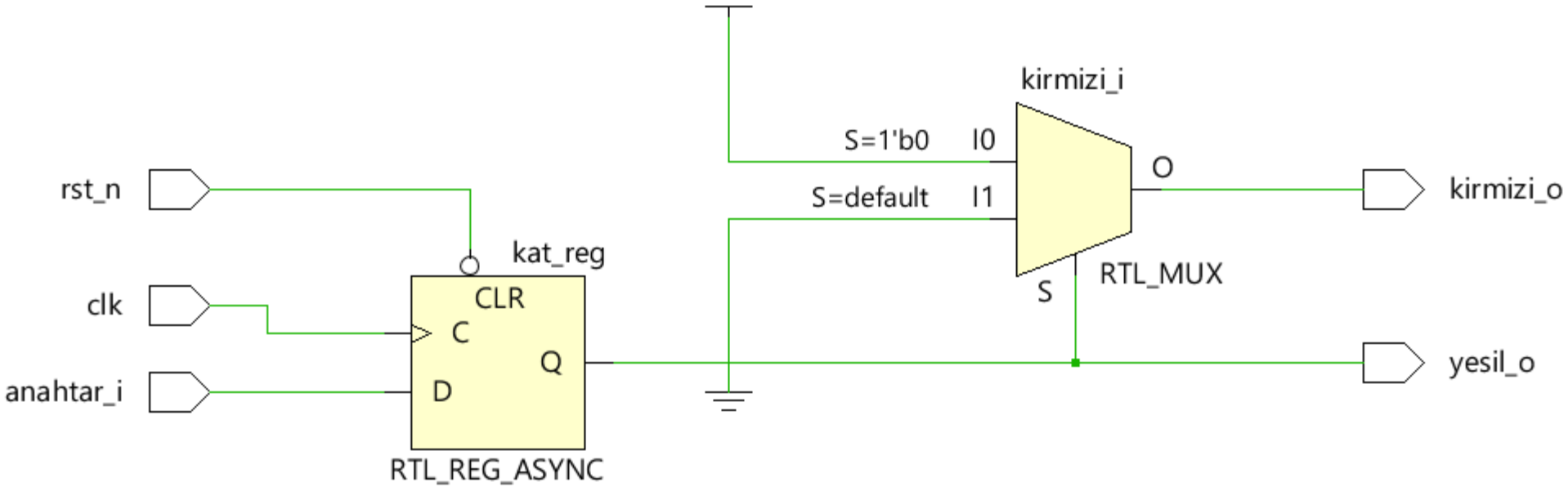
FSM ve SENKRON SIRALI DEVRE TASARIMI

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

Giriş	Durum(t)	Çıkış		Durum(t+1)
anahtar	Kat	kırmızı	yeşil	Kat_next
0	0	1	0	0
0	1	0	1	0
1	0	1	0	1
1	1	0	1	1

kırmızı = !kat
yeşil = kat
Kat_next = anahtar

2 durum → 1 FF



FSM ve SENKRON SIRALI DEVRE TASARIMI

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module fsm_ornek1
(
input clk,
input rst_n,
input anahtar_i,
output kirmizi_o,
output yesil_o
);

localparam kat0 = 1'b0;
localparam kat1 = 1'b1;

reg kat;
reg kirmizi, yesil;
```

```
always @(posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0) begin
        kat <= kat0;
    end
    else begin
        case (kat)
            kat0 : begin
                if (anahtar_i == 1'b0) begin
                    kat <= kat0;
                end
                else begin
                    kat <= kat1;
                end
            end
            kat1 : begin
                if (anahtar_i == 1'b0) begin
                    kat <= kat0;
                end
                else begin
                    kat <= kat1;
                end
            end
            default: begin
                kat <= kat0;
            end
        endcase
    end
end
```

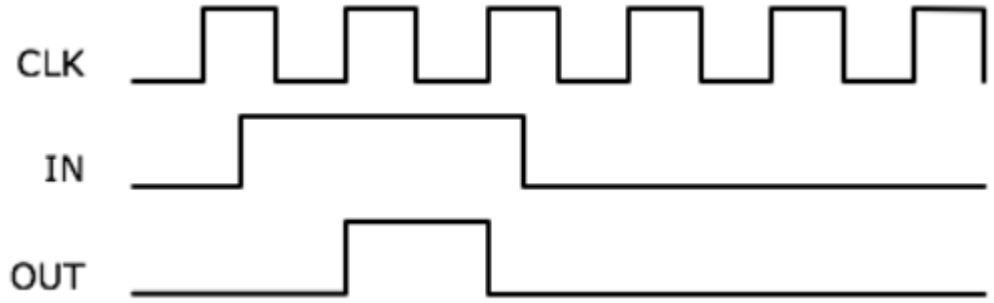
```
always@(*) begin
    if (kat == kat0) begin
        kirmizi = 1'b1;
        yesil   = 1'b0;
    end
    else begin
        kirmizi = 1'b0;
        yesil   = 1'b1;
    end
end

assign kirmizi_o = kirmizi;
assign yesil_o = yesil;

endmodule
```

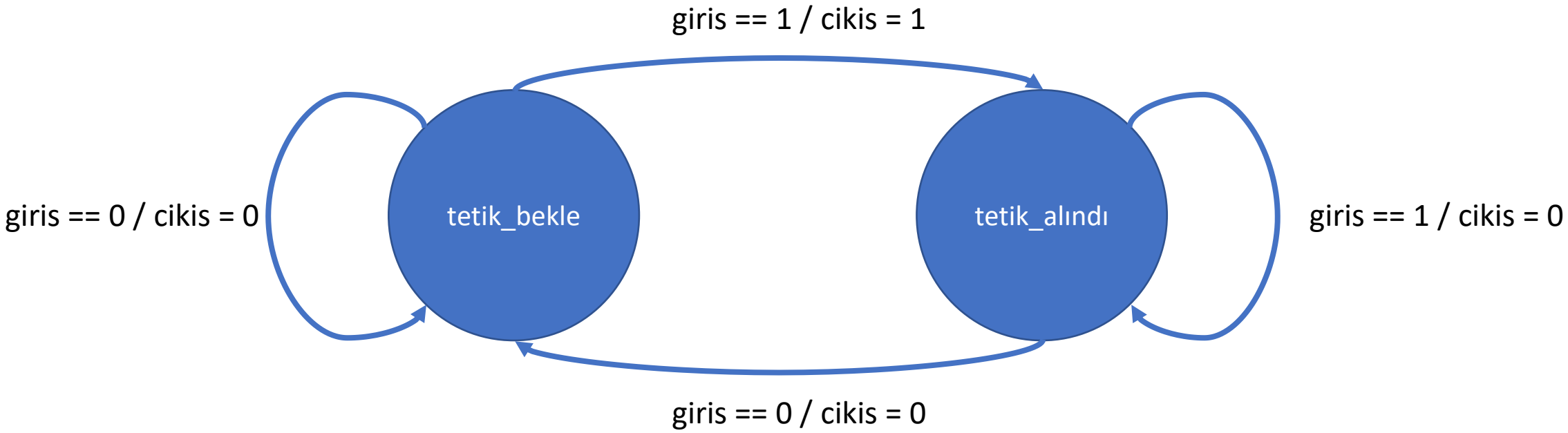
FSM ve SENKRON SIRALI DEVRE TASARIMI

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

FSM_Örnek2 (rise-edge detector): Giriş sinyalinin 0'dan 1'e çıkma durumunu tespit ettiği zaman 1 saat boyunca çıkış sinyali '1' olur. Diğer bütün durumlarda çıkış '0' olur.

Giriş	Durum(t)	Çıkış	Durum(t+1)
giris	state	cikis	state_next
0	0	0	0
0	1	0	0
1	0	1	1
1	1	0	1





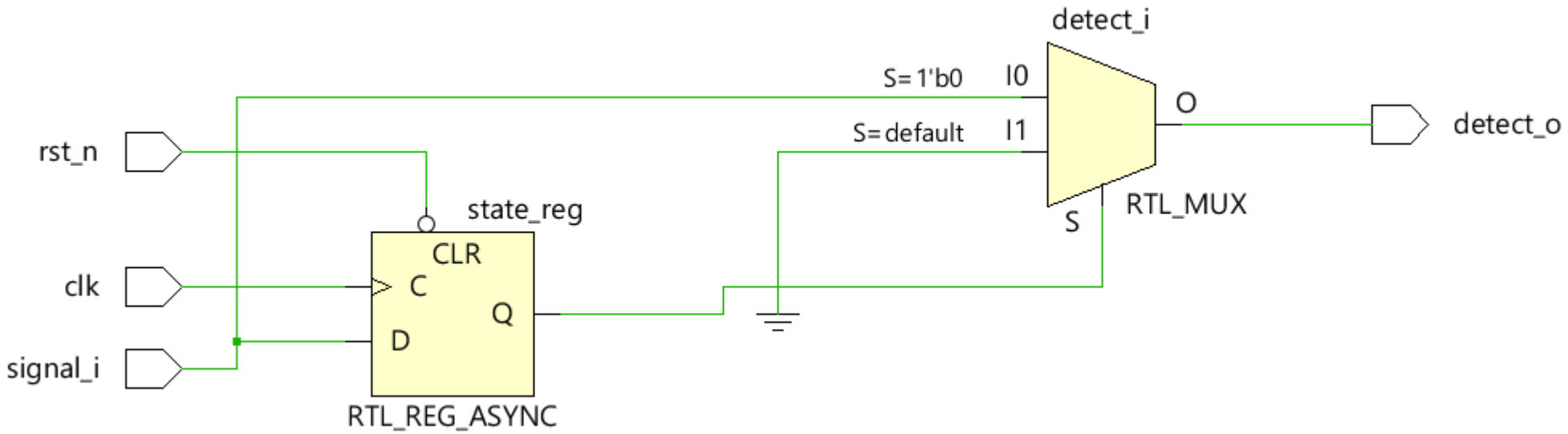
FSM ve SENKRON SIRALI DEVRE TASARIMI

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

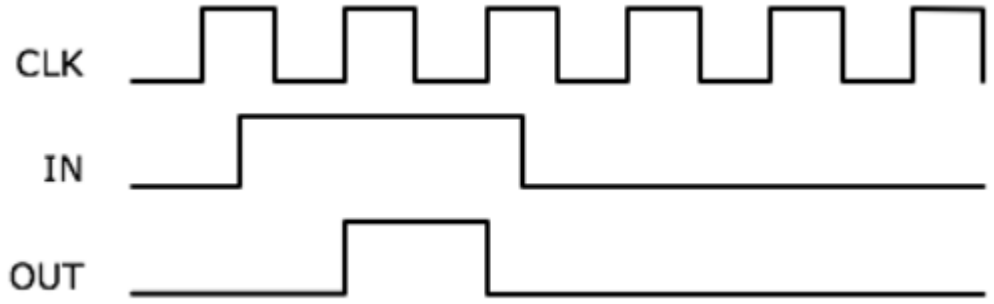
```
module fsm_ornek2
(
input clk,
input rst_n,
input signal_i,
output detect_o
);

localparam tetik_bekle  = 1'b0;
localparam tetik_alindi = 1'b1;

reg state;
reg detect;

always @(posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0) begin
        state <= tetik_bekle;
    end
    else begin
        case (state)
            tetik_bekle : begin
                if (signal_i == 1'b0) begin
                    state <= tetik_bekle;
                end
                else begin
                    state <= tetik_alindi;
                end
            end
            tetik_alindi : begin
                if (signal_i == 1'b0) begin
                    state <= tetik_bekle;
                end
                else begin
                    state <= tetik_alindi;
                end
            end
            default: begin
                state <= tetik_bekle;
            end
        endcase
    end
end
```

```
always@(*) begin
    if (state == tetik_bekle) begin
        if (signal_i == 1'b0) begin
            detect = 1'b0;
        end
        else begin
            detect = 1'b1;
        end
    end
    else begin
        detect = 1'b0;
    end
end

assign detect_o = detect;

endmodule
```

FSM ve SENKRON SIRALI DEVRE TASARIMI



Bu tasarımda sorun var. Eğer saatin yükselen kenarından hemen önce signal girişi 0'dan 1'e yükselirse, çıkış çok kısa bir süreliğine '1' olacaktır. Oysa ki istenilen davranış tam olarak bu değil. Demek ki farklı bir durum makinası tasarımı gerekiyor.

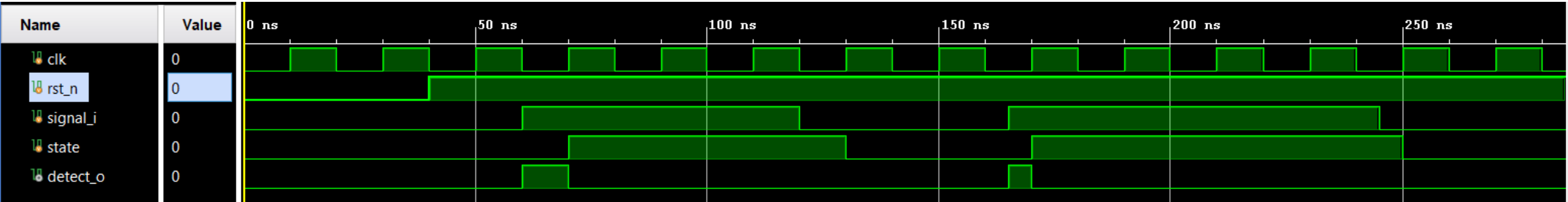


FSM ve SENKRON SIRALI DEVRE TASARIMI

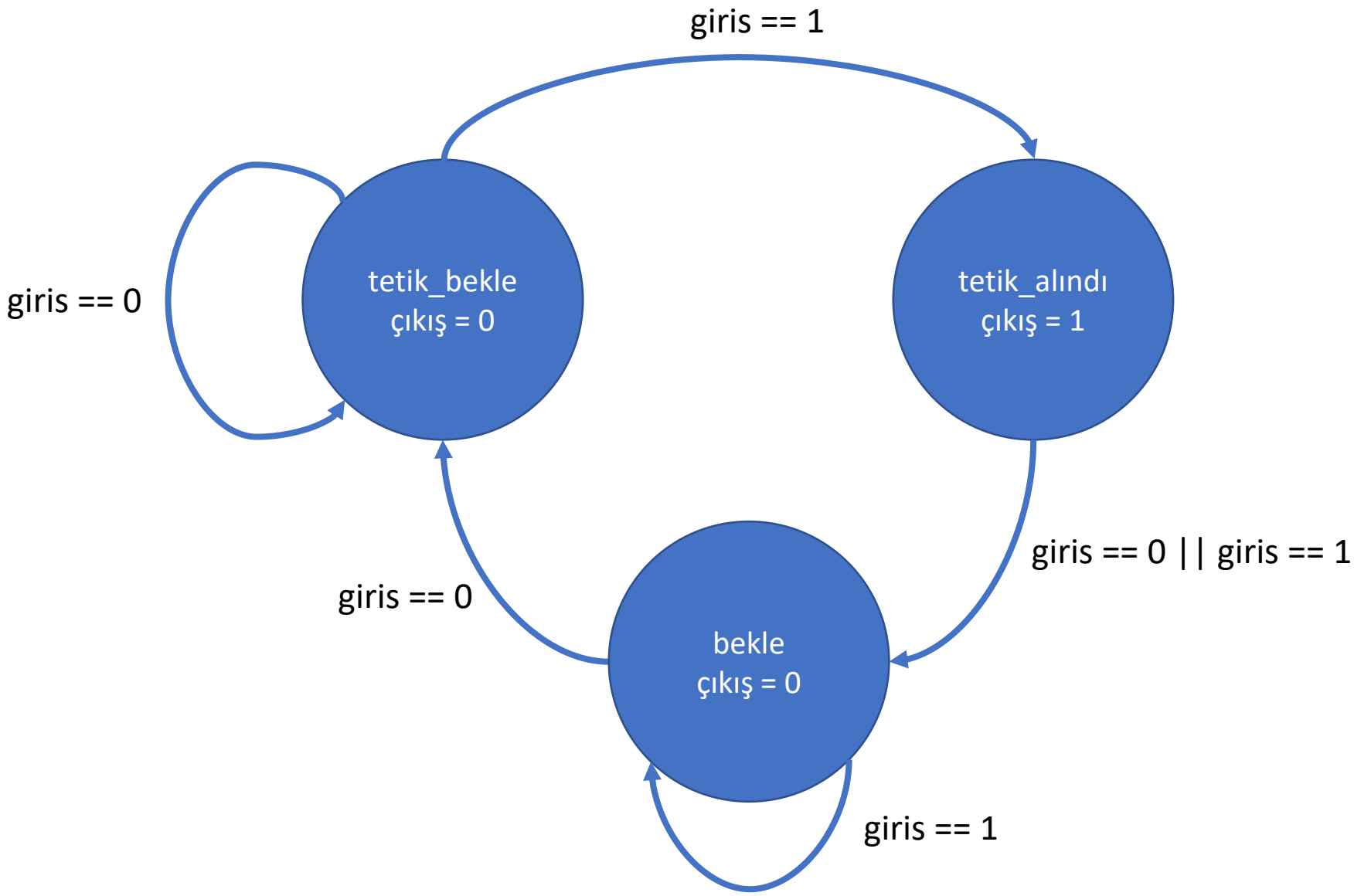
TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module tb_fsm_ornek2;

reg clk;
reg rst_n;
reg signal_i;
wire detect_o;

localparam real c_clk_period = 20;

fsm_ornek2 DUT
(
.clk        (clk)     ,
.rst_n      (rst_n)   ,
.signal_i   (signal_i),
.detect_o   (detect_o)
);

always begin
    // #10 clk = !clk;
    #(c_clk_period/2) clk = !clk;
end
```

```
initial begin
clk = 1'b0;
signal_i = 1'b0;
rst_n = 1'b0;
#40;
rst_n = 1'b1;
#20;
signal_i = 1'b1;
#60;
signal_i = 1'b0;
#45;
signal_i = 1'b1;
#80;
signal_i = 1'b0;
#40;
$finish;
end

endmodule
```



FSM ve SENKRON SIRALI DEVRE TASARIMI



FSM ve SENKRON SIRALI DEVRE TASARIMI

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

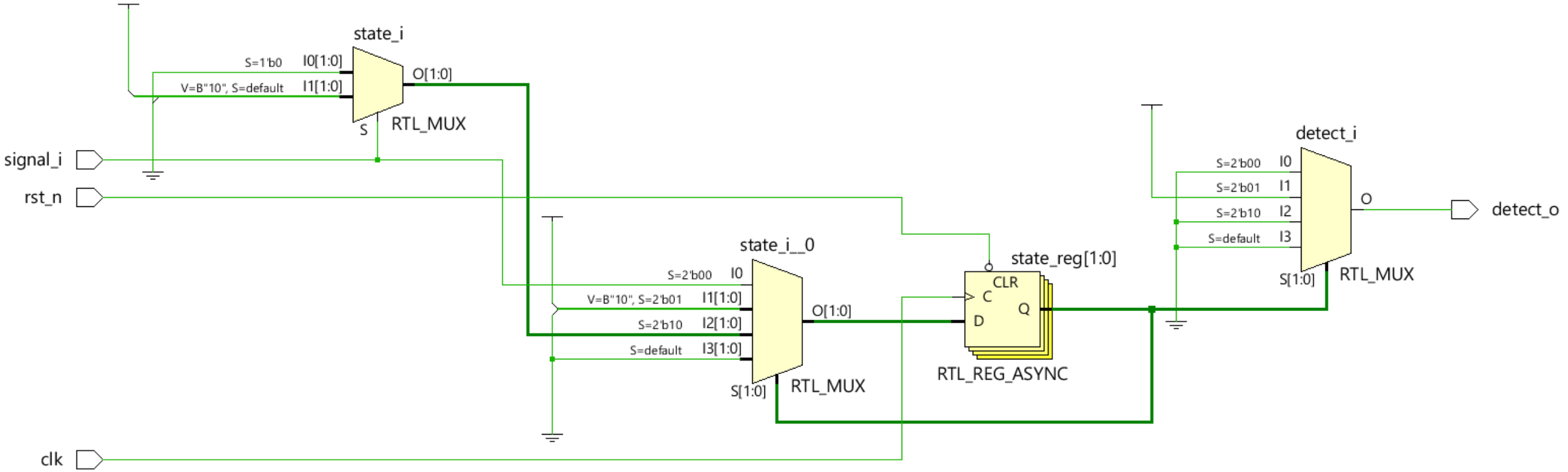
```
module fsm_ornek3
(
input clk,
input rst_n,
input signal_i,
output detect_o
);

localparam tetik_bekle  = 2'b00;
localparam tetik_alindi = 2'b01;
localparam bekle        = 2'b10;

reg [1:0] state;
reg detect;

always @(posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0) begin
        state <= tetik_bekle;
    end
```

```
case (state)
    tetik_bekle : begin
        if (signal_i == 1'b0) begin
            state <= tetik_bekle;
        end
        else begin
            state <= tetik_alindi;
        end
    end
    tetik_alindi : begin
        state <= bekle;
    end
    bekle : begin
        if (signal_i == 1'b0) begin
            state <= tetik_bekle;
        end
        else begin
            state <= bekle;
        end
    end
    default: begin
        state <= tetik_bekle;
    end
endcase
```

```
always@(*) begin
    case (state)
        tetik_bekle : begin
            detect = 1'b0;
        end
        tetik_alindi : begin
            detect = 1'b1;
        end
        bekle : begin
            detect = 1'b0;
        end
        default: begin
            detect = 1'b0;
        end
    endcase
end

assign detect_o = detect;

endmodule
```

FSM ve SENKRON SIRALI DEVRE TASARIMI

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

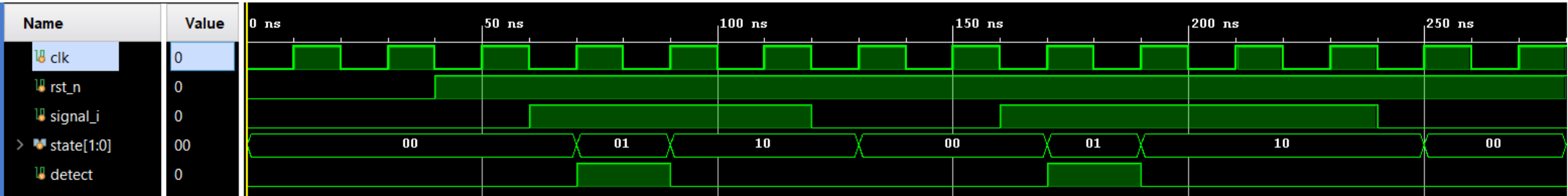


FSM ve SENKRON SIRALI DEVRE TASARIMI

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module tb_fsm_ornek3;

reg clk;
reg rst_n;
reg signal_i;
wire detect_o;

fsm_ornek3 DUT
(
.clk        (clk)     ,
.rst_n      (rst_n)   ,
.signal_i   (signal_i),
.detect_o   (detect_o)
);

always begin
    #10 clk = !clk;
end
```

```
initial begin
clk = 1'b0;
signal_i = 1'b0;
rst_n = 1'b0;
#40;
rst_n = 1'b1;
#20;
signal_i = 1'b1;
#60;
signal_i = 1'b0;
#40;
signal_i = 1'b1;
#80;
signal_i = 1'b0;
#40;
$finish;
end

endmodule
```



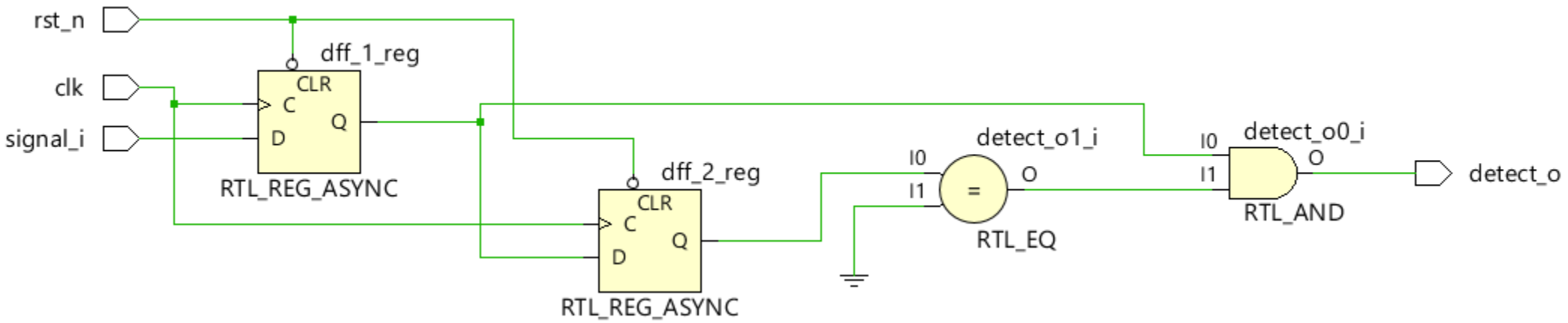
YÜKSELEN KENAR TESPİT DEVRESİ

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

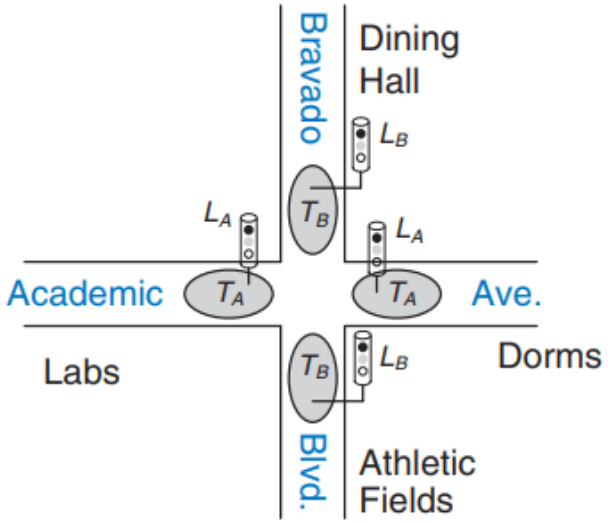
```
module rise_edge_detect
(
input clk,
input rst_n,
input signal_i,
output detect_o
);

reg dff_1, dff_2;
reg detect;
```

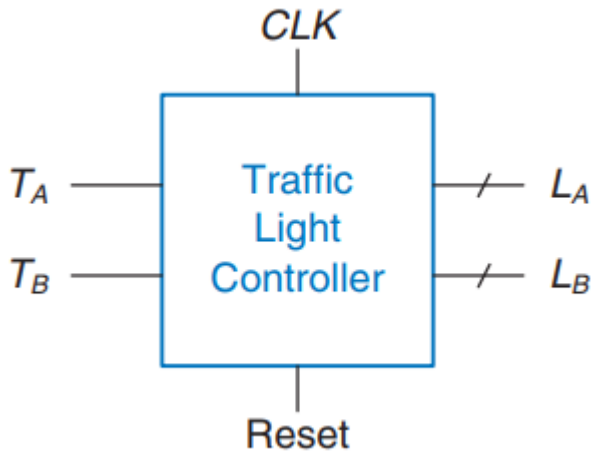
```
always @(posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0) begin
        dff_1 <= 1'b0;
        dff_2 <= 1'b0;
    end
    else begin
        dff_1   <= signal_i;
        dff_2   <= dff_1;
    end
end
```

```
always @(*) begin
    detect  = 1'b0;
    if (dff_1 == 1'b1 && dff_2 == 1'b0) begin
        detect = 1'b1;
    end
end

assign detect_o = detect;

endmodule
```



FSM ve SENKRON SIRALI DEVRE TASARIMI





Harris & Harris kitabı 3.4.1 FSM Design Example Örneği

- Akademik öğrenciler Laboratuvar ve Yurtlar arasında gidip geliyorlar
- Spor akademisi öğrencileri yemekhane ve spor alanları arasında gidip geliyorlar
- Akademik öğrenciler yolda ders çalıştıkları, spor öğrencileri de topla ilgilendikleri için yola bakmıyorlar ve kavşakta çarpışmalar yaşanıyor
- Ta ve Tb sensörleri iki caddede öğrenci olup olmadığını kontrol ediyor
- La ve Lb trafik ışıkları kırmızı sari yeşil yanabiliyor
- 5 saniyelik bir saat sinyali var
- Öğrencilerin çarpışmamaları için sensör girişleri ve trafik lamba çıkışlarını içeren bir FSM tasarlayın, FSM diyagram ve durum geçiş tablosu çizin, Verilog kodunu yazın

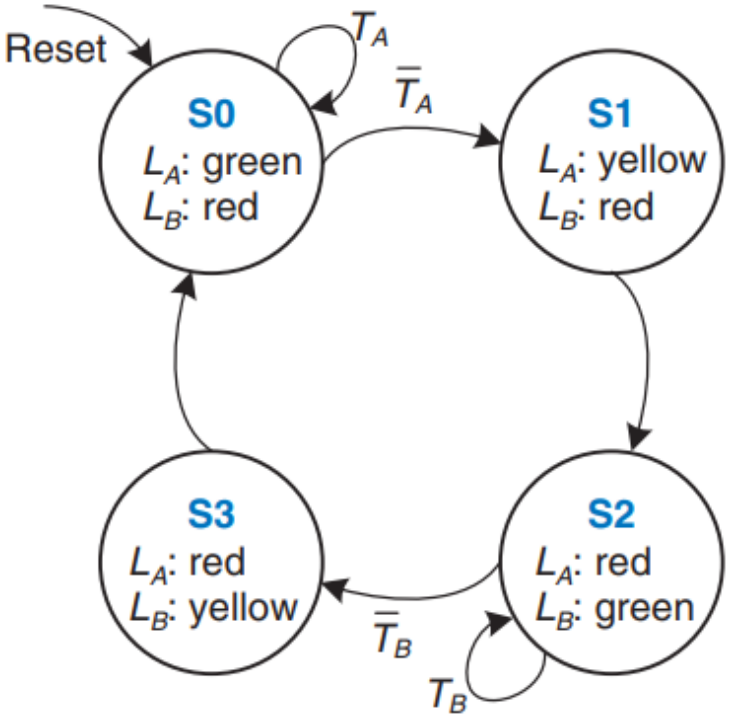


Table 3.1 State transition table

Current State S	Inputs T_A	Inputs T_B	Next State S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

Table 3.2 State encoding

State	Encoding $S_{1:0}$
S0	00
S1	01
S2	10
S3	11

Table 3.3 Output encoding

Output	Encoding $L_{1:0}$
green	00
yellow	01
red	10

$$S_1' = S_1 \oplus S_0$$
$$S_0' = \overline{S}_1\overline{S}_0\overline{T}_A + S_1\overline{S}_0\overline{T}_B$$

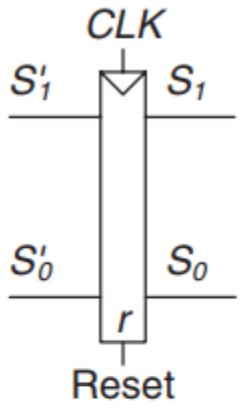
$$L_{A1} = S_1$$
$$L_{A0} = \overline{S}_1 S_0$$
$$L_{B1} = \overline{S}_1$$
$$L_{B0} = S_1 S_0$$

Table 3.4 State transition table with binary encodings

Current State S_1	Current State S_0	Inputs T_A	Inputs T_B	Next State S_1'	Next State S_0'
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

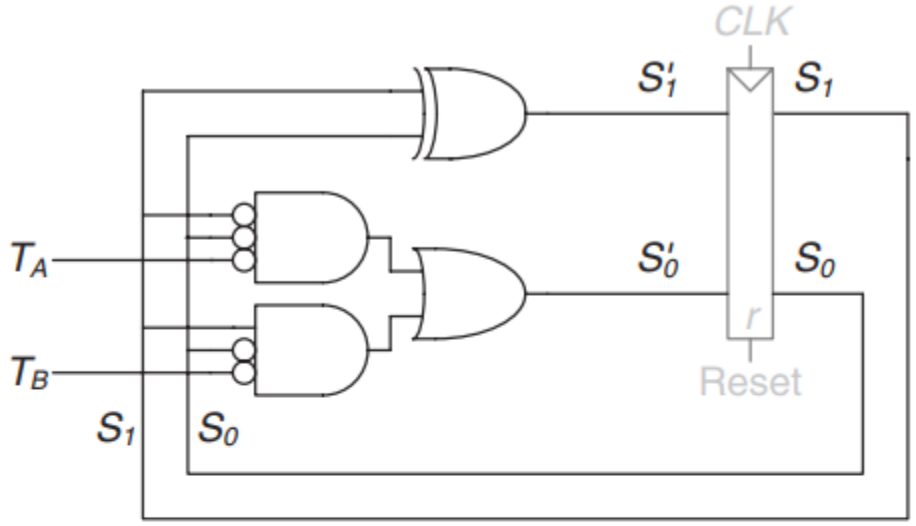
Table 3.5 Output table

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



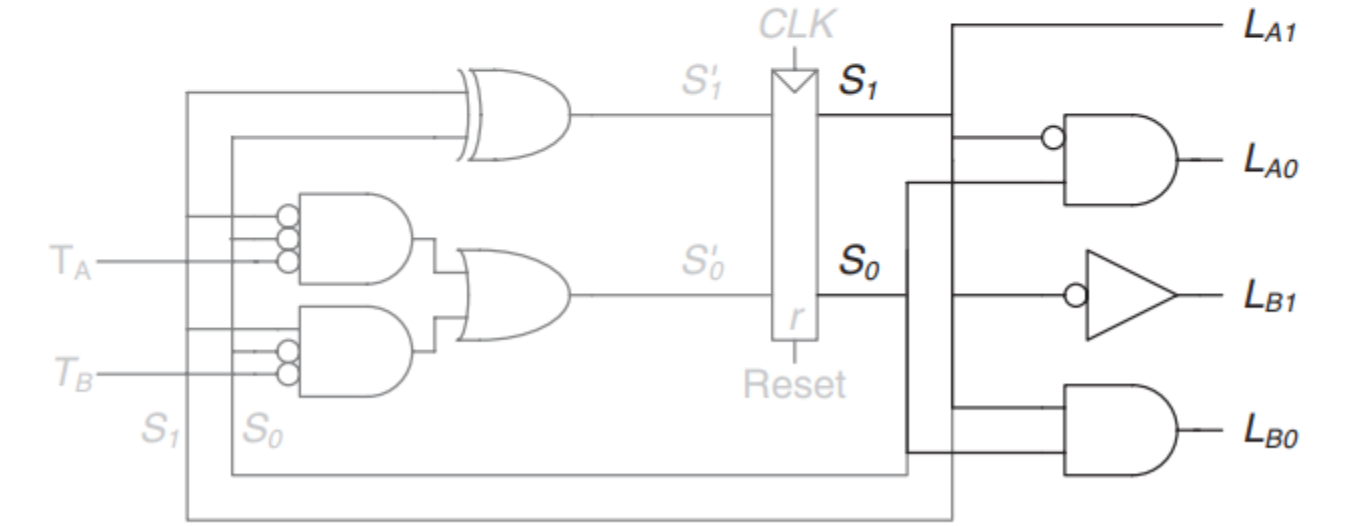
state register

(a)



inputs next state logic state register

(b)



inputs next state logic state register output logic outputs

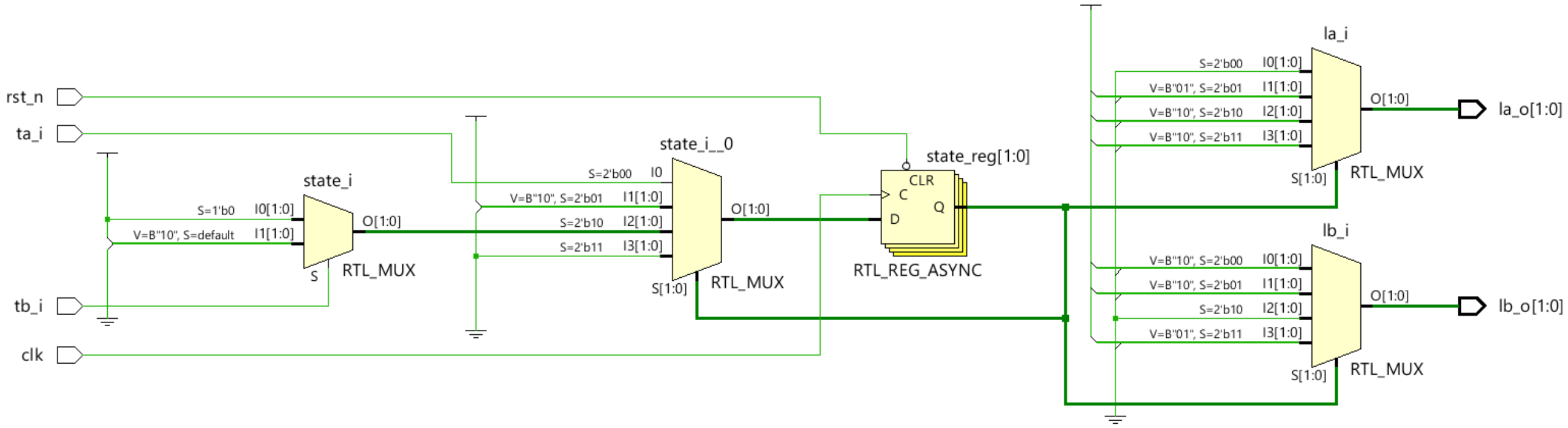
(c)

```
module fsm_ornek4
(
input clk,
input rst_n,
input ta_i,
input tb_i,
output [1:0] la_o,
output [1:0] lb_o
);

localparam S0  = 2'b00;
localparam S1  = 2'b01;
localparam S2  = 2'b10;
localparam S3  = 2'b11;

localparam green    = 2'b00;
localparam yellow   = 2'b01;
localparam red      = 2'b10;

reg [1:0] la,lb;
reg [1:0] state;

always @(posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0) begin
        state <= S0;
    end
    else begin
```

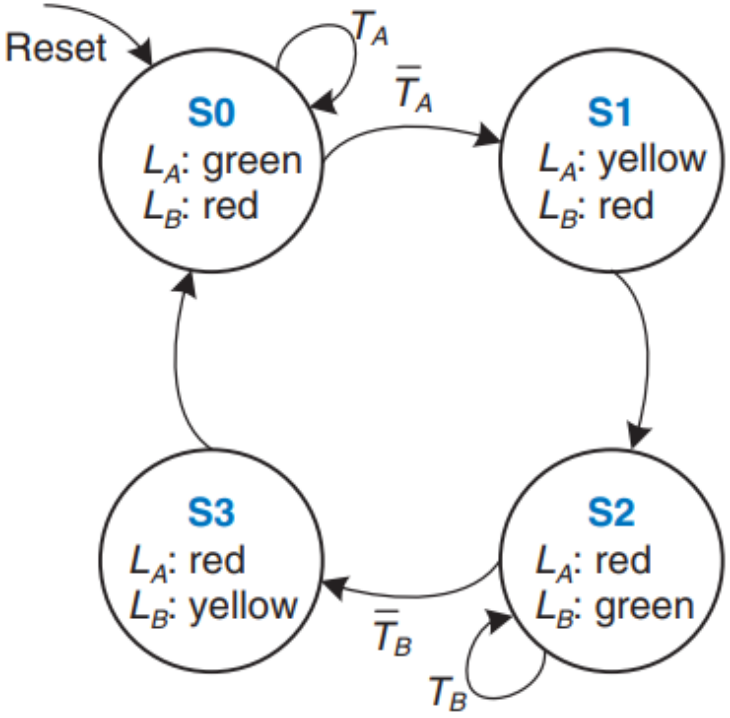
```
case (state)
    S0 : begin
        if (ta_i == 1'b0) begin
            state <= S0;
        end
        else begin
            state <= S1;
        end
    end
    S1 : begin
        state <= S2;
    end
    S2 : begin
        if (tb_i == 1'b0) begin
            state <= S3;
        end
        else begin
            state <= S2;
        end
    end
    S3 : begin
        state <= S0;
    end
    default: begin
        state <= S0;
    end
endcase
```

```
always@(*) begin
    case (state)
        S0 : begin
            la = green;
            lb = red;
        end
        S1 : begin
            la = yellow;
            lb = red;
        end
        S2 : begin
            la = red;
            lb = green;
        end
        S3 : begin
            la = red;
            lb = yellow;
        end
        default: begin
            la = green;
            lb = red;
        end
    endcase
end

assign la_o = la;
assign lb_o = lb;

endmodule
```

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi


rst_n
ta_i
tb_i
clk
state_i
S=1'b0
I0[1:0]
V=B"10", S=default
I1[1:0]
O[1:0]
S
RTL_MUX
state_i__0
S=2'b00
I0
V=B"10", S=2'b01
I1[1:0]
S=2'b10
I2[1:0]
S=2'b11
I3[1:0]
O[1:0]
S[1:0]
RTL_MUX
state_reg[1:0]
CLR
C
D
Q
RTL_REG_ASYNC
la_i
S=2'b00
I0[1:0]
V=B"01", S=2'b01
I1[1:0]
V=B"10", S=2'b10
I2[1:0]
V=B"10", S=2'b11
I3[1:0]
O[1:0]
S[1:0]
RTL_MUX
la_o[1:0]
lb_i
V=B"10", S=2'b00
I0[1:0]
V=B"10", S=2'b01
I1[1:0]
S=2'b10
I2[1:0]
V=B"01", S=2'b11
I3[1:0]
O[1:0]
S[1:0]
RTL_MUX
lb_o[1:0]

DURUM KODLAMALARI (FSM ENCODINGS)



Binary	One-Hot	Gray
00	0001	00
01	0010	01
10	0100	11
11	1000	10

Table 3.2 State encoding

State	Encoding $S_{1:0}$
S0	00
S1	01
S2	10
S3	11

Table 3.4 State transition table with binary encodings

Current State S_1	Current State S_0	Inputs T_A	Inputs T_B	Next State S_1'	Next State S_0'
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

```
localparam S0  = 2'b00;
localparam S1  = 2'b01;
localparam S2  = 2'b10;
localparam S3  = 2'b11;
```

```
reg [1:0] state;
```