Tahmini Ders İçeriği
(Tentative Couse Schedule – Syllabus)

1. Hafta: Sayı sistemleri, onluk/ikilik taban sayı gösterimleri, mantıksal kapılar, computer system overview, başarım (performance)

2. Hafta: 2'lik tabanda işaretli sayılar, mikroişlemci tarihi, benchmarking, başarım,

3. Hafta: Başarım, Amdahl yasası, RISC-V development Environment, Verilog HDL ile Birleşik (Combinational) devreler

4. Hafta:, Verilog HDL ile sıralı (sequential) mantıksal devre ve sonlu durum makinası tasarımı, timing analysis

5. Hafta: Aritmetik devre tasarımları: Toplama, çıkarma, çarpma, bölme

6. Hafta: Fixed ve Floating-Point sayı gösterimleri

7. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

8. Hafta: RISC-V buyruk kümesi mimarisi (ISA) ve buyrukların tanıtımı

9. Hafta: Tek-çevrim işlemci tasarımı (single-cycle CPU)

10. Hafta: Çok-çevrim işlemci tasarımı (multi-cycle CPU)

11. Hafta: Boruhatlı işlemci tasarımı (pipelined CPU)

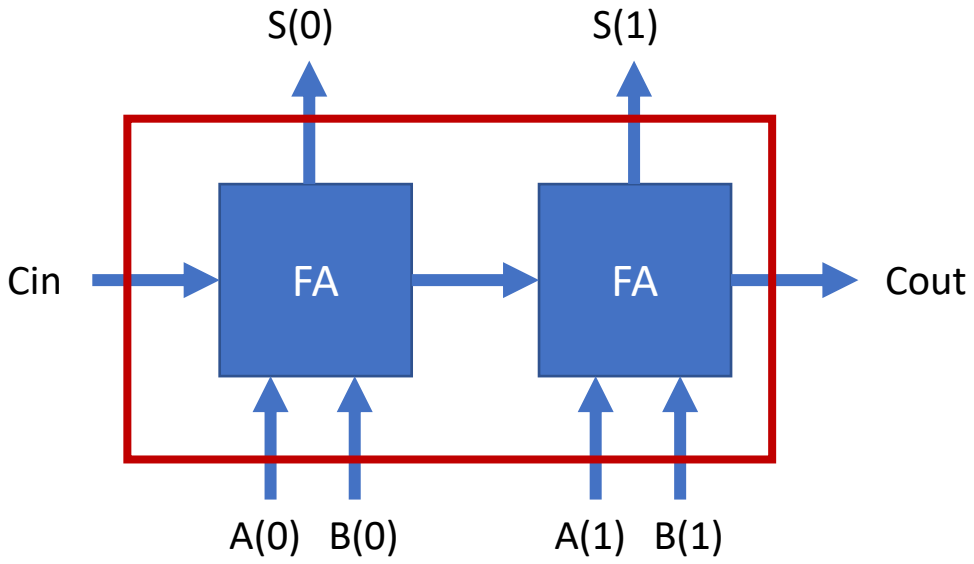
12. Hafta: Bellek sistemi ve hiyerarşisi

13. Hafta: İleri mimari konuları: Branch prediction, superscalar cpu, out-of-order execution, multi-core systems

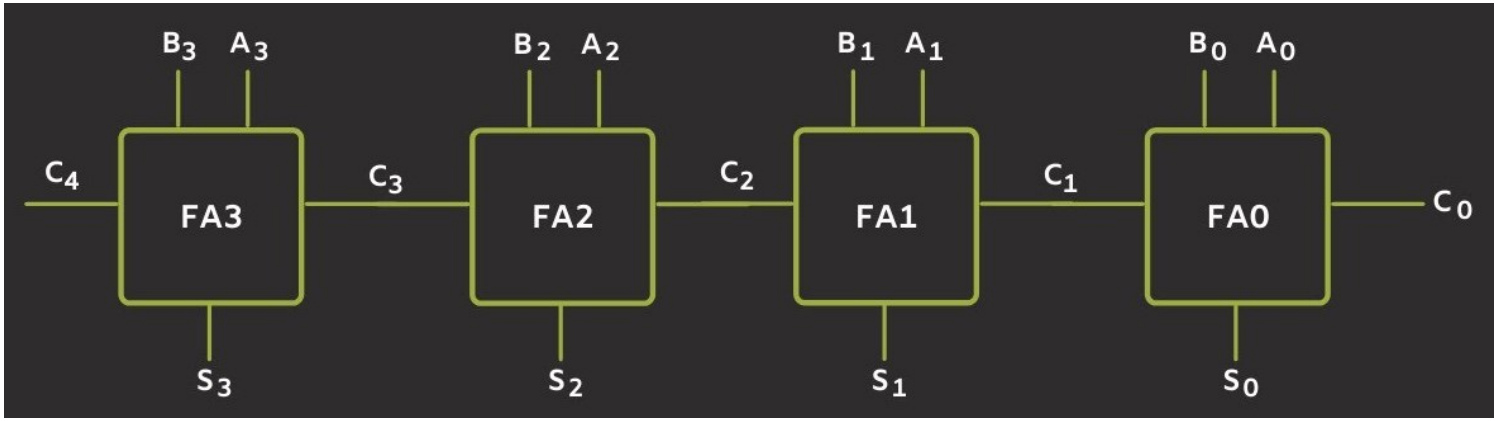
14. Hafta: Gömülü sistemler, mikrodenetleyiciler, SoCs

Ripple-Carry Adder

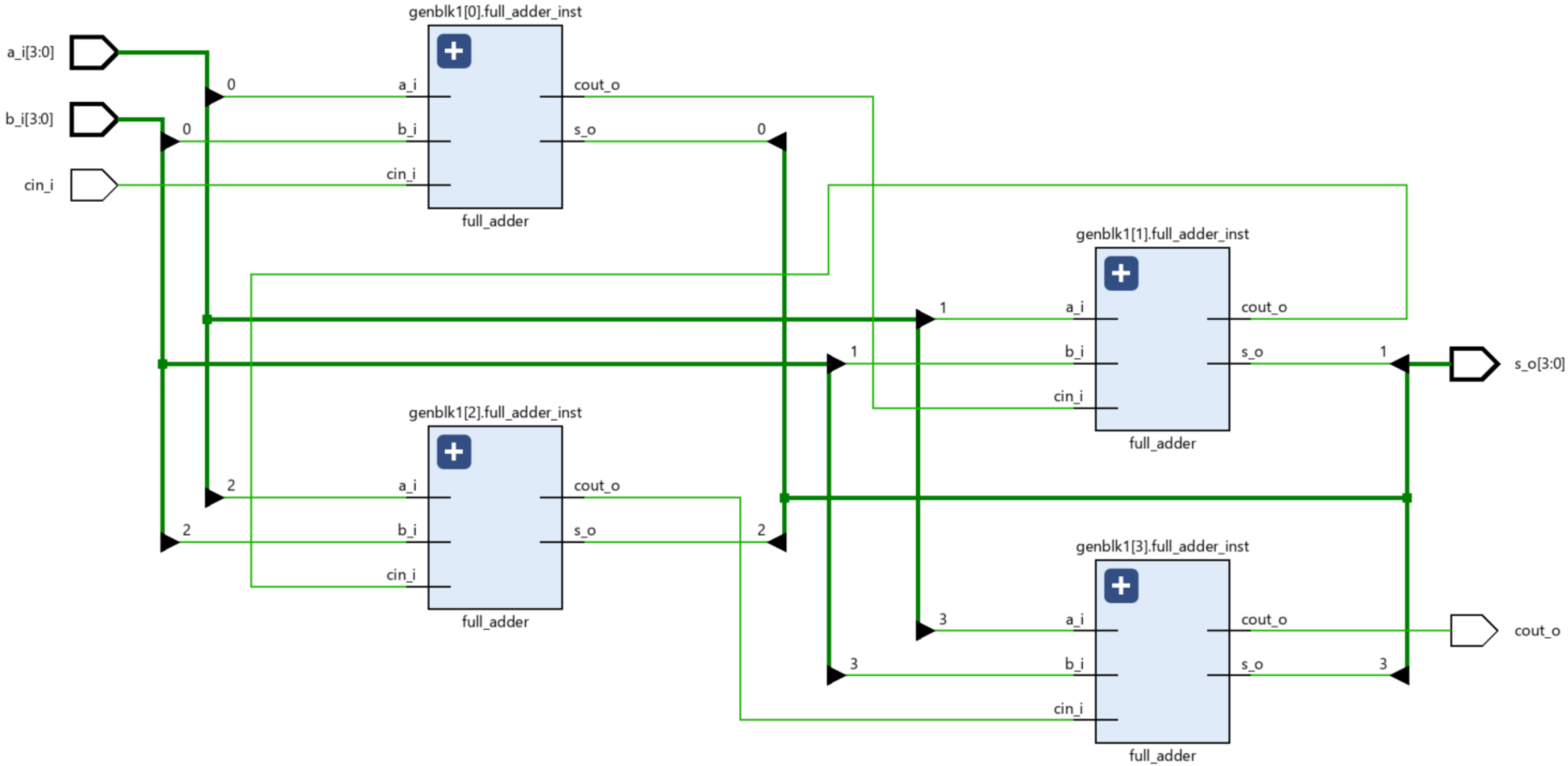
2-bit Binary Adder



4-bit Ripple-Carry Adder

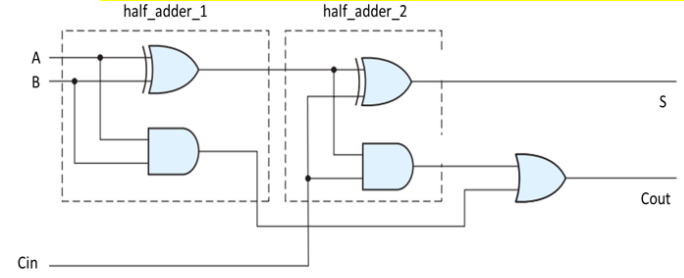


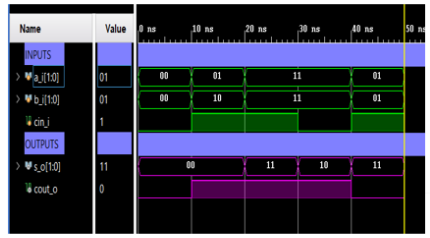

```
module ripple_carry_adder
#(parameter N = 4)
(
input [N-1:0] a_i,
input [N-1:0] b_i,
input cin_i,
output [N-1:0] s_o,
output cout_o
);

wire [N:0] carry;

genvar i;
generate
for (i=0; i<N; i=i+1) begin
    full_adder full_adder_inst
    (
        .a_i(a_i[i]),
        .b_i(b_i[i]),
        .cin_i(carry[i]),
        .s_o(s_o[i]),
        .cout_o(carry[i+1])
    );
end
endgenerate

assign carry[0] = cin_i;
assign cout_o   = carry[N];

endmodule
```

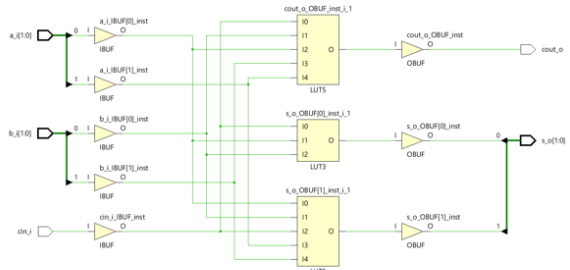

a_i[3:0]
b_i[3:0]
cin_i
genblk1[0].full_adder_inst
a_i
b_i
cin_i
cout_o
s_o
full_adder
genblk1[1].full_adder_inst
genblk1[2].full_adder_inst
genblk1[3].full_adder_inst
s_o[3:0]
cout_o
0
1
2
3

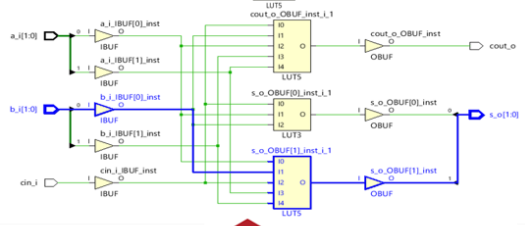
MANTIKSAL DEVRE TASARIMI (LOGIC DESIGN)

DERS8: COMBINATIONAL DEVRELER HALF FULL ADDER VERILOG





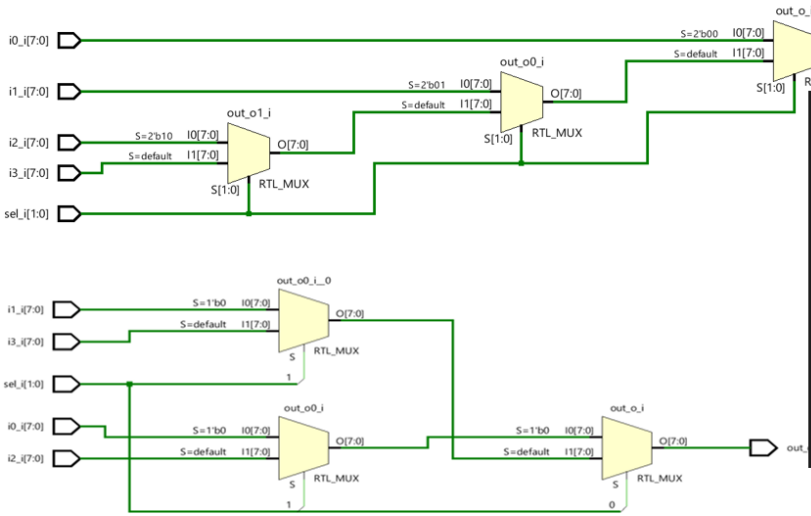




TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

MANTIKSAL DEVRE TASARIMI (LOGIC DESIGN)

DERS9: MULTIPLEXER – DECODER – VERILOG ASSIGN KEYWORD

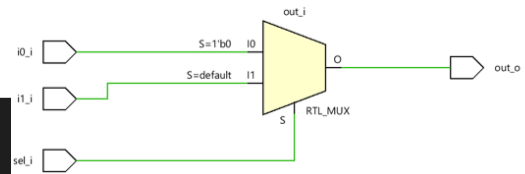



```
module mux_4_1_8bit_assign
(
input [7:0] i0_i,
input [7:0] i1_i,
input [7:0] i2_i,
input [7:0] i3_i,
input [1:0] sel_i,
output [7:0] out_o
);

//assign out_o =  (sel_i[0] == 1'b0) ?
//                (sel_i[1] == 1'b0) ? i0_i : i2_i :
//                (sel_i[1] == 1'b0) ? i1_i : i3_i;

assign out_o =  (sel_i == 2'b00) ? i0_i :
                (sel_i == 2'b01) ? i1_i :
                (sel_i == 2'b10) ? i2_i :
                                   i3_i;

endmodule
```

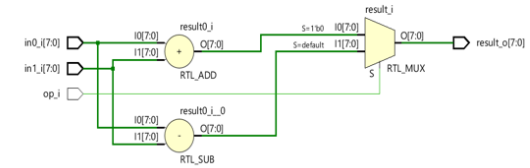
TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

MANTIKSAL DEVRE TASARIMI (LOGIC DESIGN)

DERS10: VERILOG DAVRANIŞSAL MODELLEME - ALWAYS

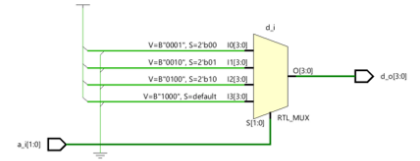



```
module mux_2_1_behav
(
input i0_i,
input i1_i,
input sel_i,
output out_o
);

reg out;

always@(i0_i or i1_i or sel_i) begin
    if (sel_i == 1'b0) begin
        out = i0_i;
    end
    else begin
        out = i1_i;
    end
end

assign out_o = out;

endmodule
```




```
module decoder_2_4_behav
(
input [1:0] a_i,
output [3:0] d_o
);

reg [3:0] d;

always @(a_i) begin
    case (a_i)
        2'b00   : d = 4'b0001;
        2'b01   : d = 4'b0010;
        2'b10   : d = 4'b0100;
        default : d = 4'b1000;
    endcase
end

assign d_o = d;

endmodule
```

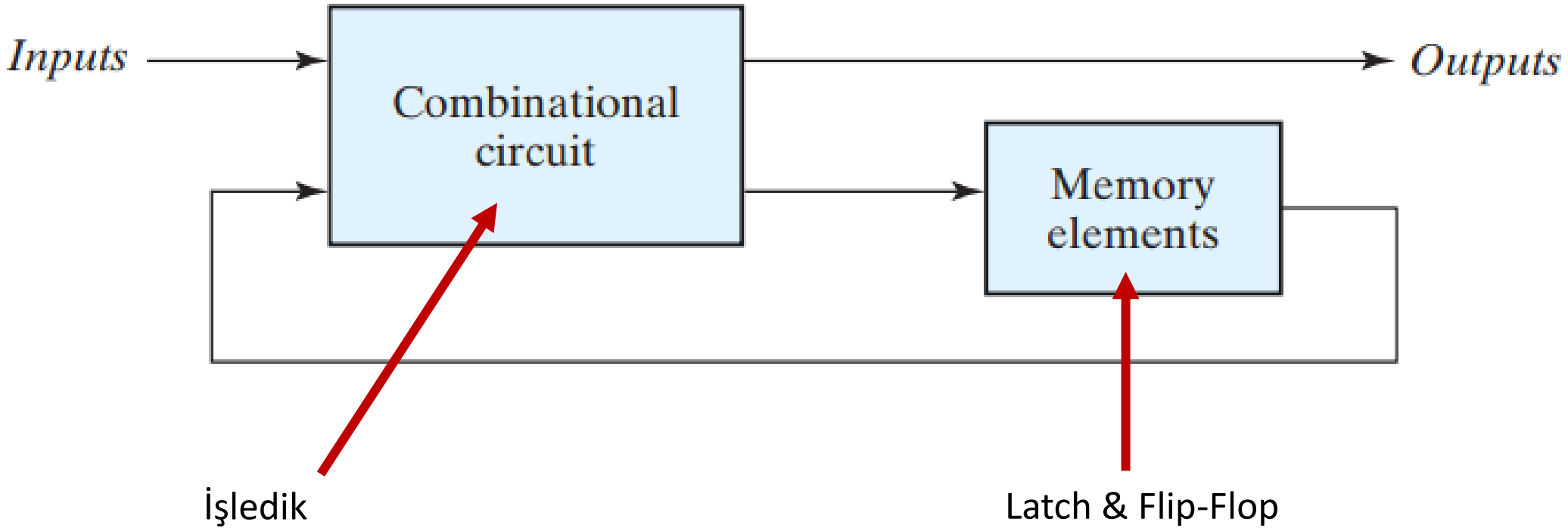
```
module add_sub_8bit
(
input [7:0] in0_i,
input [7:0] in1_i,
input op_i,
output [7:0] result_o
);

reg [7:0] result;

always @(in0_i, in1_i, op_i) begin
    if (op_i == 1'b0) begin
        result = in0_i + in1_i;
    end
    else begin
        result = in0_i - in1_i;
    end
end

assign result_o = result;

endmodule
```



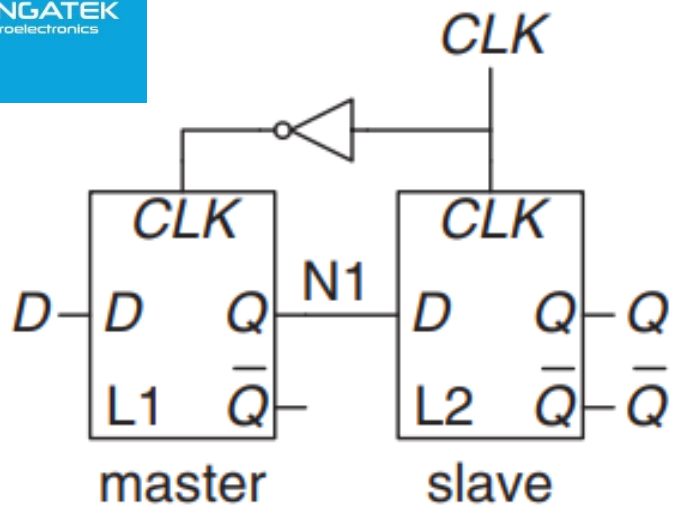
TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

Sıralı (Sequential) Devreler

Çıkış sinyali, giriş sinyallerinin hem o anki hem de geçmiş değerine bağlı olan devreler. İçerisinde bellek (memory) birimi bulundururlar. Devre içerisindeki belleğin o anki değeri, devrenin durumu (state) olarak adlandırılır.

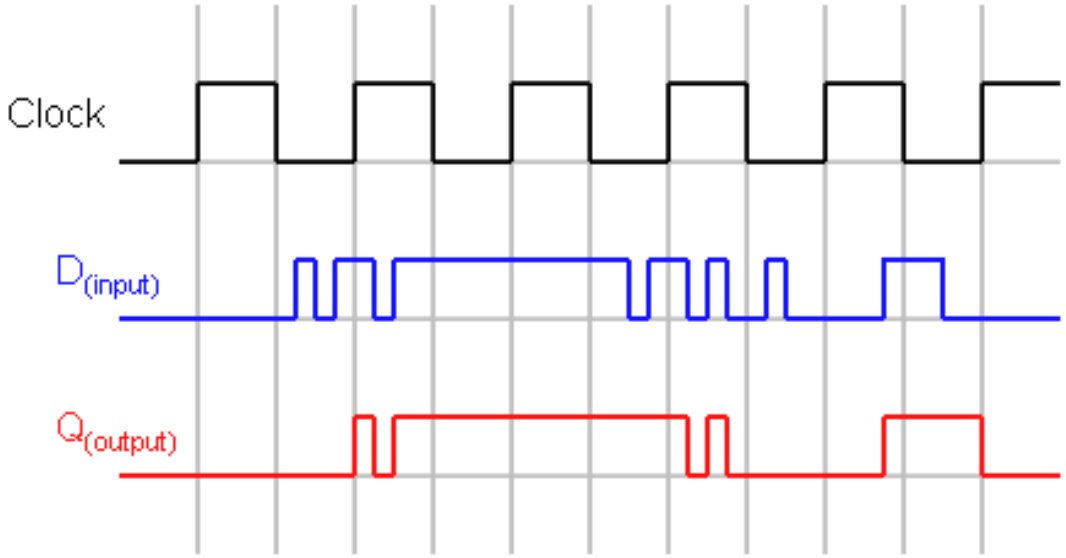


D Flip-Flop

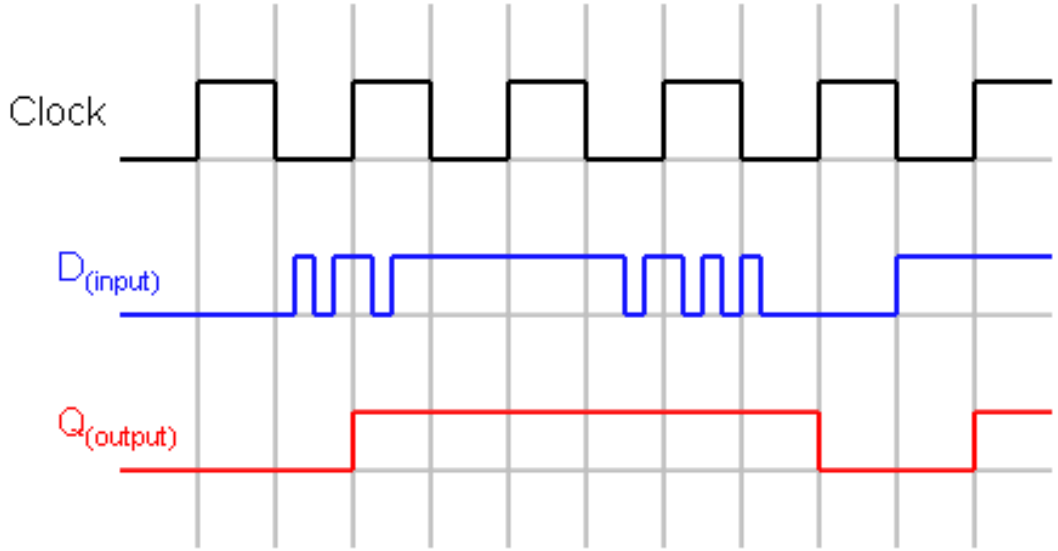


In other words, a D flip-flop copies D to Q on the rising edge of the clock and remembers its state at all other times. Reread this definition until you have it memorized; one of the most common problems for beginning digital designers is to forget what a flip-flop does.

D Latch



D Flip-Flop



D Flip-Flop Types & Register (Yazmaç)

```
module dff
(
input clk,
input d_i,
output reg q_o
);

always @(posedge clk ) begin
    q_o <= d_i;
end

endmodule
```

```
module dff
(
input clk,
input d_i,
output reg q_o
);

always @(negedge clk ) begin
    q_o <= d_i;
end

endmodule
```

```
module dff
(
input clk,
input rst,
input d_i,
output reg q_o
);

// active-high, synchronous reset
always @(posedge clk) begin
    if (rst) begin
        q_o <= 0;
    end else begin
        q_o <= d_i;
    end
end

endmodule
```

```
module dff
(
input clk,
input rstn,
input d_i,
output reg q_o
);

// active-low, synchronous reset
always @(posedge clk) begin
    if (!rstn) begin
        q_o <= 0;
    end else begin
        q_o <= d_i;
    end
end

endmodule
```

```
module dff
(
input clk,
input rst,
input d_i,
output reg q_o
);

// active-high, asynchronous reset
always @(posedge clk, posedge rst) begin
    if (rst) begin
        q_o <= 0;
    end else begin
        q_o <= d_i;
    end
end

endmodule
```

```
module dff
(
input clk,
input rstn,
input d_i,
output reg q_o
);

// active-low, asynchronous reset
always @(posedge clk, negedge rstn) begin
    if (!rstn) begin
        q_o <= 0;
    end else begin
        q_o <= d_i;
    end
end

endmodule
```

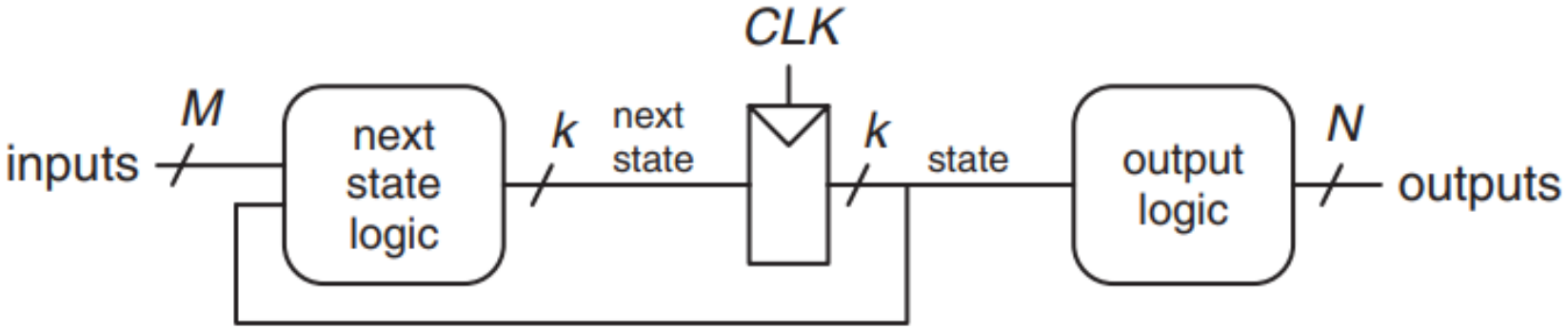
```
module register
#(parameter N = 8)
(
input clk,
input rstn,
input [N-1:0] d_i,
output [N-1:0] q_o
);

always @(posedge clk) begin
    if (!rstn) begin
        q_o <= 0;
    end else begin
        q_o <= d_i;
    end
end

endmodule
```

Sayaç (Counter) – Shift Register

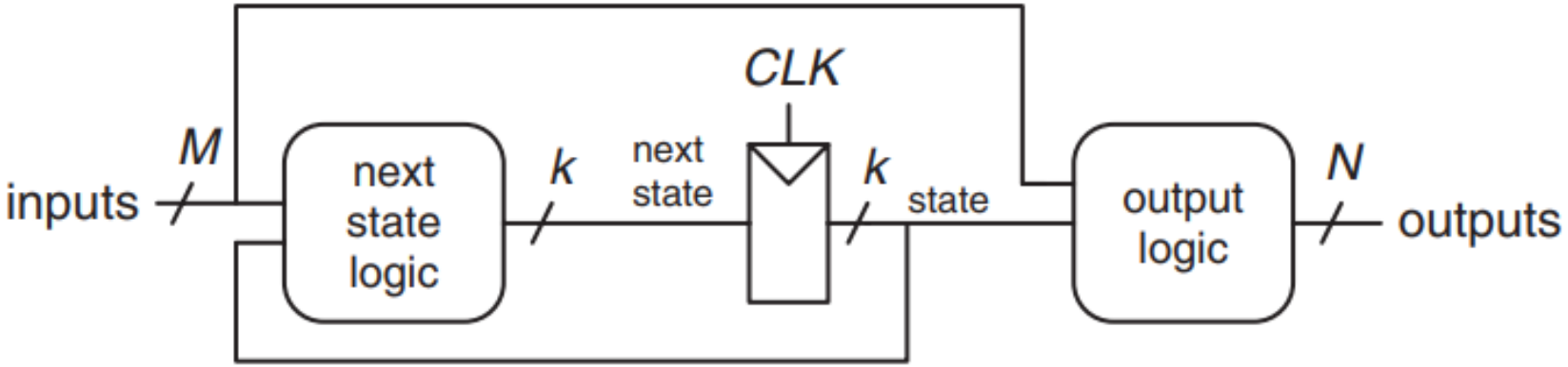
```
module counter
#(parameter N = 4)
(
input clk,
input rstn,
input en_i,
output reg [N-1:0] count_o
);

reg [N-1:0] count;

always @(posedge clk) begin
    if (!rstn) begin
        count_o <= 0;
    end else begin
        if (en_i) begin
            count_o <= count_o + 1;
        end
    end
end

endmodule
```

```
module shreg
#(parameter N = 8)
(
input clk,
input rstn,
input datain_i,
input en_i,
input load_i,
input [N-1:0] loadval_i,
output dataout_o
);

reg [N-1:0] r_reg;

always @(posedge clk) begin
    if (!rstn) begin
        r_reg <= 0;
    end else begin
        if (load_i) begin
            r_reg           <= loadval_i;
        end else if (en_i) begin
            r_reg[0]        <= datain_i;
            r_reg[N-1:1]    <= r_reg[N-2:0];
        end else begin
            r_reg           <= r_reg;
        end
    end
end

assign dataout_o = r_reg[N-1];

endmodule
```

SONLU DURUM MAKİNELERİ



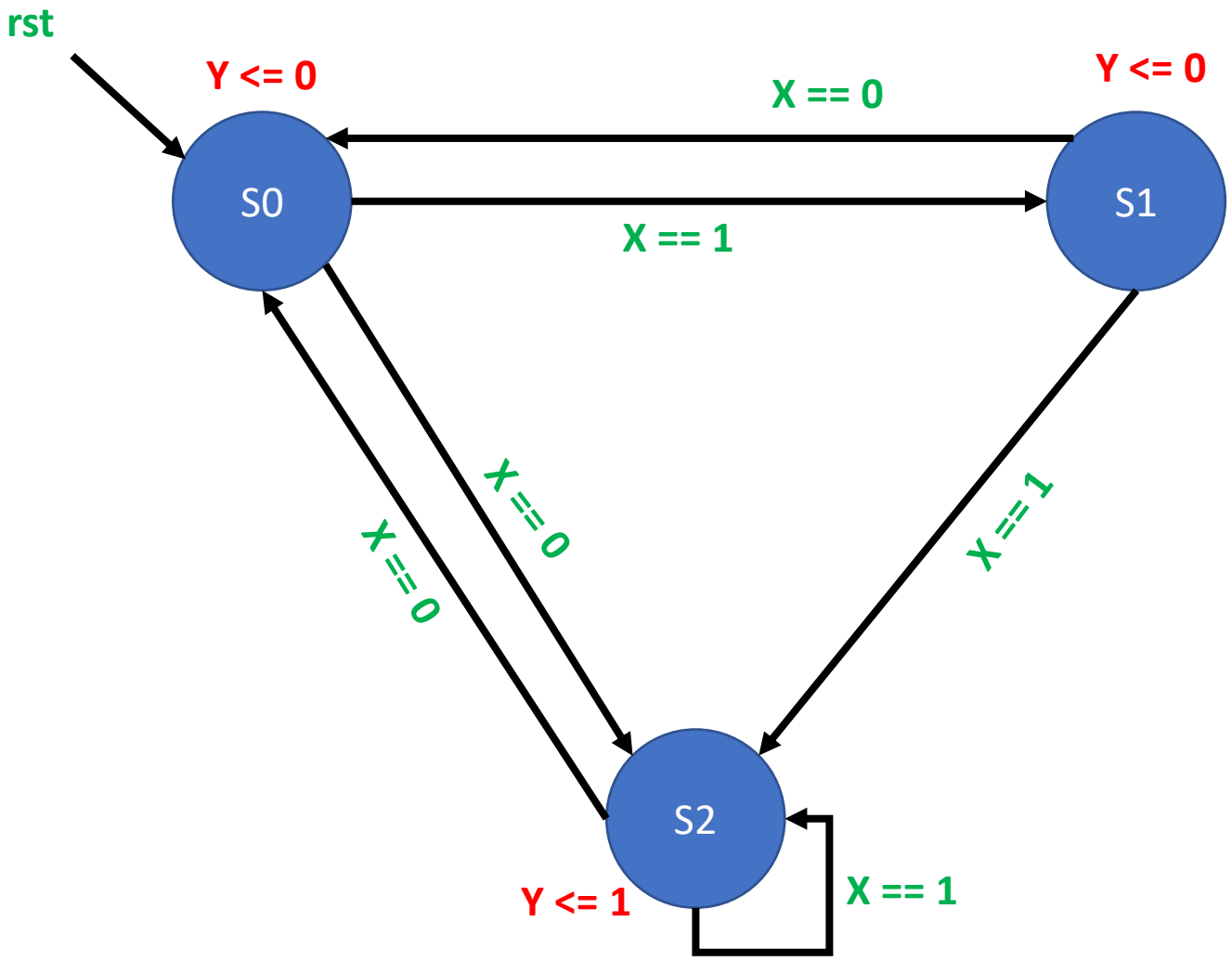
(a)



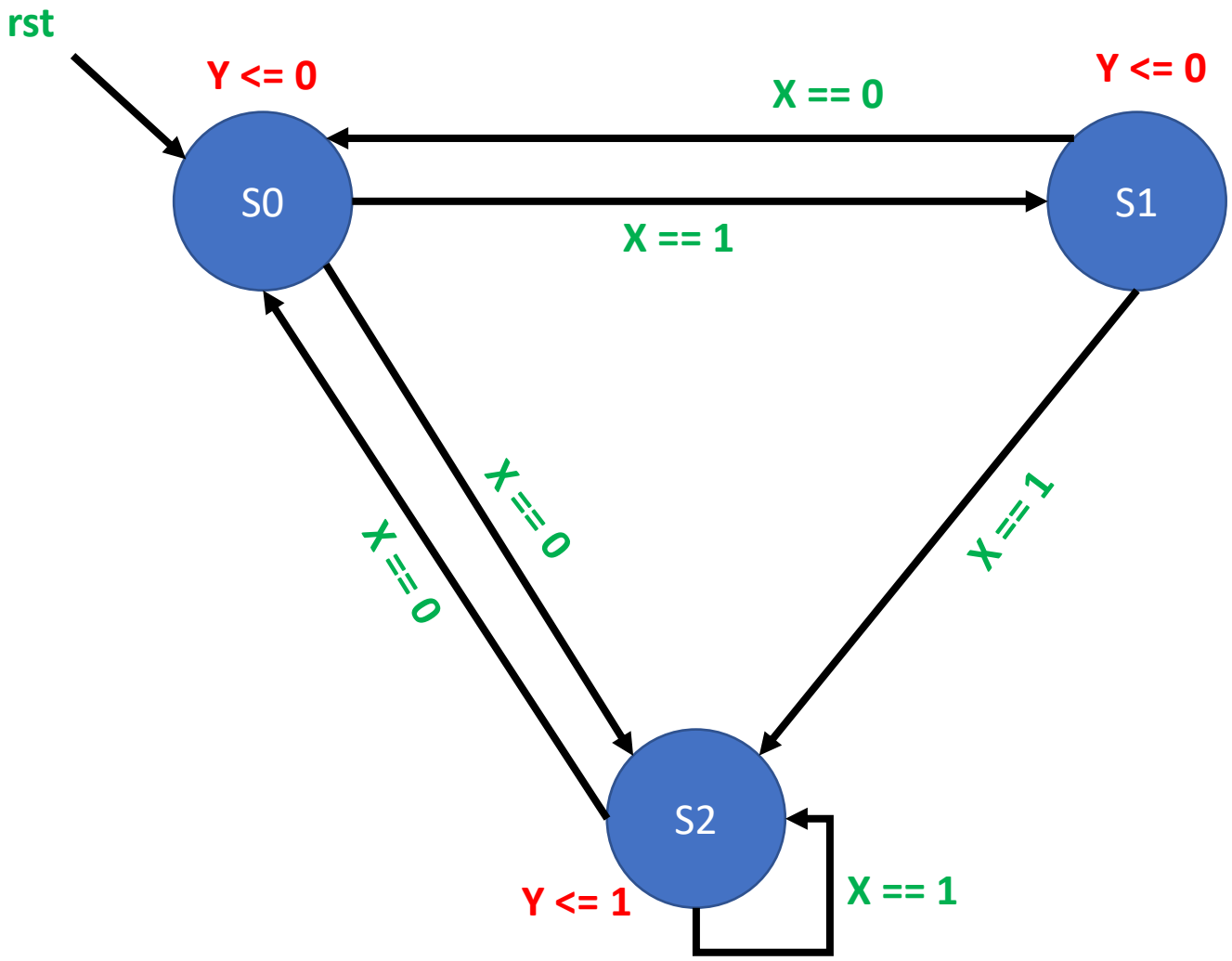
(b)

SONLU DURUM MAKİNELERİ (MOORE)

Giriş Sinyali : X
Çıkış Sinyali : Y
Durumlar : S0, S1, S2



SONLU DURUM MAKİNELERİ (MOORE)

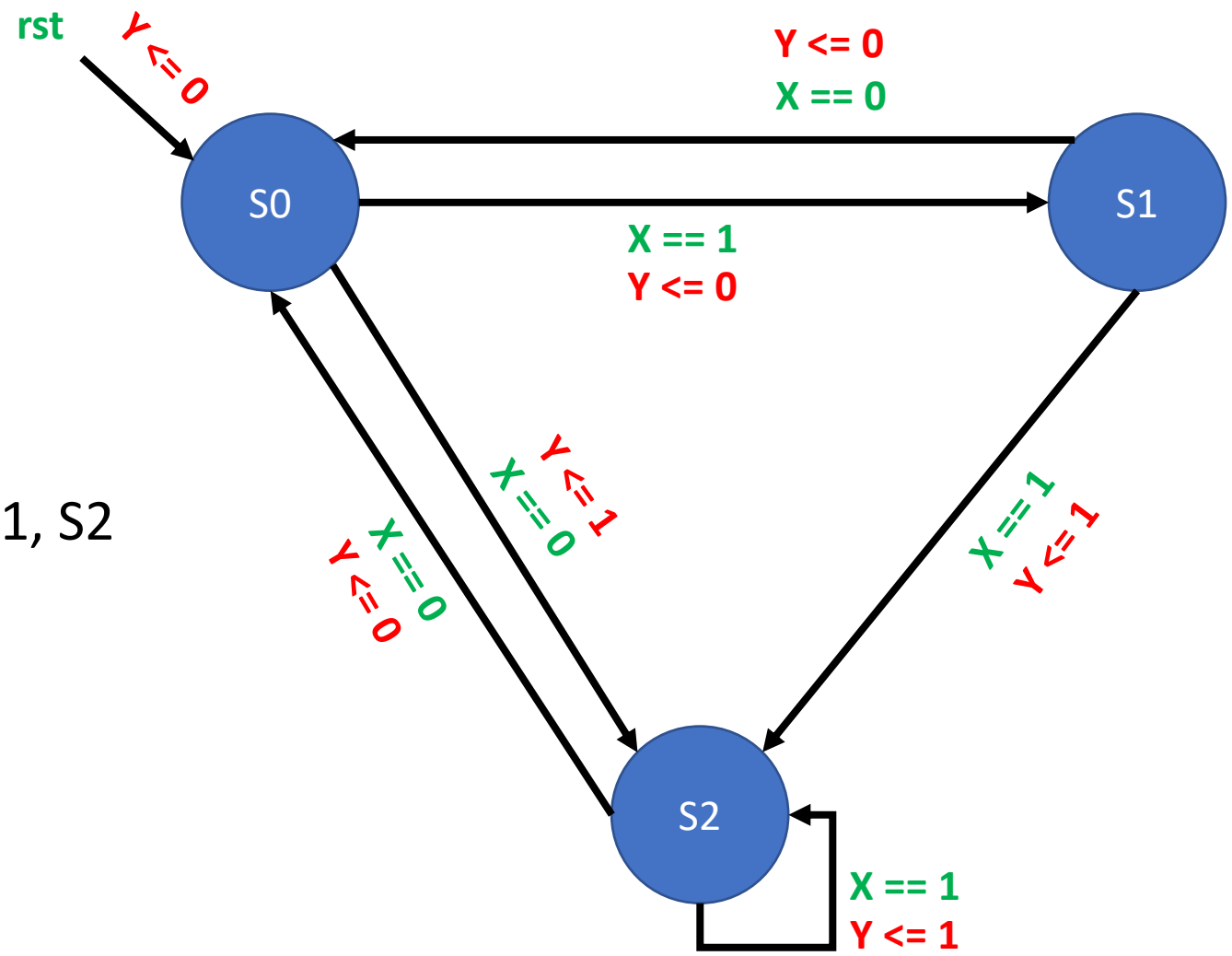



```
module fsm_moore
(
input clk,
input rstn,
input x_i,
output reg y_o
);

reg [1:0] state;

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

```
always @(posedge clk) begin
    if (!rstn) begin
        state   <= S0;
        y_o     <= 0;
    end else begin
        case (state)
            S0 : begin
                y_o <= 0;
                if (!x_i) begin
                    state <= S2;
                end else begin
                    state <= S1;
                end
            end
            S1 : begin
                y_o <= 0;
                if (!x_i) begin
                    state <= S0;
                end else begin
                    state <= S2;
                end
            end
            S2 : begin
                y_o <= 1;
                if (!x_i) begin
                    state <= S0;
                end else begin
                    state <= S2;
                end
            end
            default: begin

            end
        endcase
    end

end
```

SONLU DURUM MAKİNELERİ (MEALY)

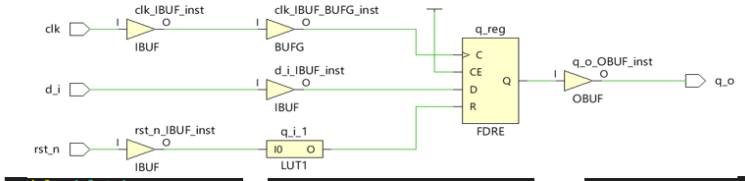
Giriş Sinyali : X
Çıkış Sinyali : Y
Durumlar : S0, S1, S2




```
always @(posedge clk) begin
    if (!rstn) begin
        state   <= S0;
        y_o     <= 0;
    end else begin
        case (state)
            S0 : begin
                if (!x_i) begin
                    y_o     <= 1;
                    state   <= S2;
                end else begin
                    y_o     <= 1;
                    state   <= S1;
                end
            end
            S1 : begin
                if (!x_i) begin
                    state   <= S0;
                    y_o     <= 0;
                end else begin
                    state   <= S2;
                    y_o     <= 1;
                end
            end
            S2 : begin
                if (!x_i) begin
                    state   <= S0;
                    y_o     <= 0;
                end else begin
                    state   <= S2;
                    y_o     <= 1;
                end
            end
            default: begin

            end
        endcase
```

MANTIKSAL DEVRE TASARIMI (LOGIC DESIGN)

DERS11: SEQUENTIAL CIRCUITS – FLIP FLOP – RESET TİPLERİ




```systemverilog
module d_latch
(
input d_i,
input clk,
output q_o
);

reg q;

always@(clk,d_i) begin
    if (clk == 1'b1) begin
        q <= d_i;
    end
end

assign q_o = q;

endmodule
```

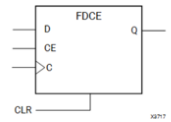
```systemverilog
module d_ff
(
input d_i,
input clk,
output q_o
);

reg q;

always@(posedge clk) begin
    q <= d_i;
end

assign q_o = q;

endmodule
```

```systemverilog
module d_ff_rst
(
input d_i,
input rst_n,    // Active-low reset
input clk,
output q_o
);

reg q;

always@(posedge clk) begin
    if (rst_n == 1'b0) begin
        q <= 1'b0;
    end
    else begin
        q <= d_i;
    end
end

assign q_o = q;

endmodule
```

FDCE

Primitive: D Flip-Flop with Clock Enable and Asynchronous Clear

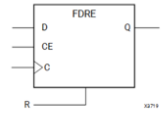
- PRIMITIVE_GROUP: REGISTER
- PRIMITIVE_SUBGROUP: SDR
- Families: UltraScale, UltraScale+



FDRE

Primitive: D Flip-Flop with Clock Enable and Synchronous Reset

- PRIMITIVE_GROUP: REGISTER
- PRIMITIVE_SUBGROUP: SDR
- Families: UltraScale, UltraScale+

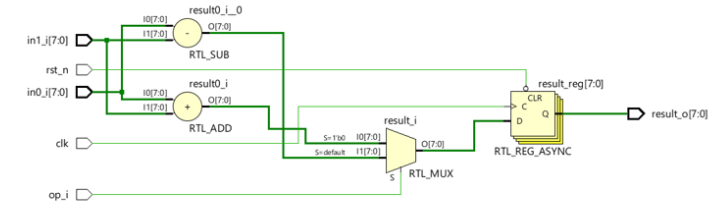


TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

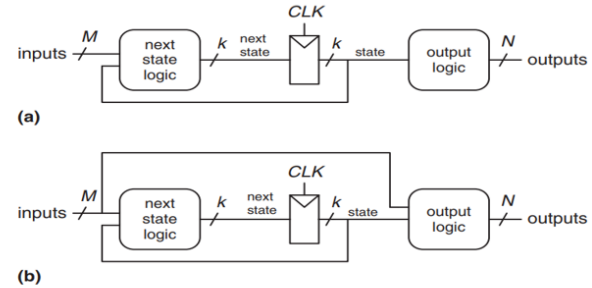
MANTIKSAL DEVRE TASARIMI (LOGIC DESIGN)

DERS12: FINITE STATE MACHINES– SYNCHRONIZATION – VERILOG PARAMETER KEYWORD

```systemverilog
module add_sub_8bit_sync
(
input [7:0] in0_i,
input [7:0] in1_i,
input op_i,
input clk,
input rst_n,
output [7:0] result_o
);

reg [7:0] result;

always @(posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0) begin
        result <= 8'b00000000;
    end
    else begin
        if (op_i == 1'b0) begin
            result <= in0_i + in1_i;
        end
        else begin
            result <= in0_i - in1_i;
        end
    end
end

assign result_o = result;

endmodule
```




```systemverilog
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
```

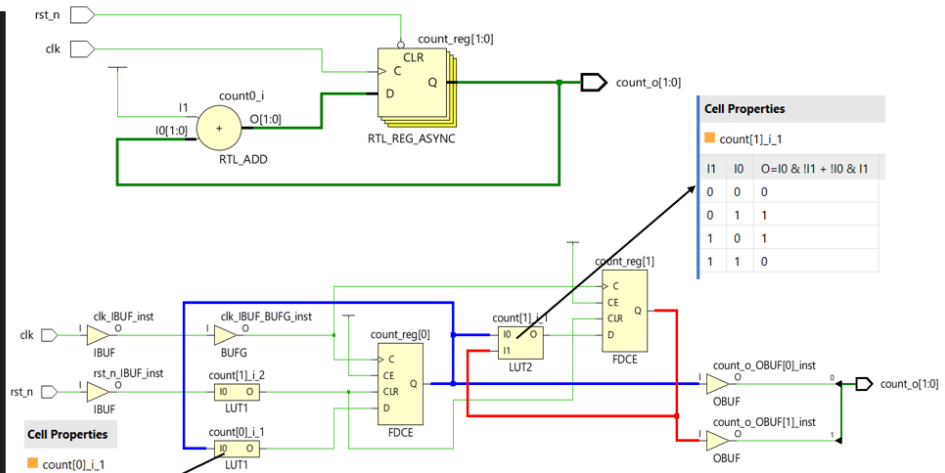


TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

MANTIKSAL DEVRE TASARIMI (LOGIC DESIGN)

DERS13: FINITE STATE MACHINE DEVRE ANALİZİ – SAYAÇLAR (COUNTERS) VERILOG

```systemverilog
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
```



Cell Properties

count[1]_i_1

I1	I0	O=I0 & !I1 + !I0 & I1
0	0	0
0	1	1
1	0	1
1	1	0

Cell Properties

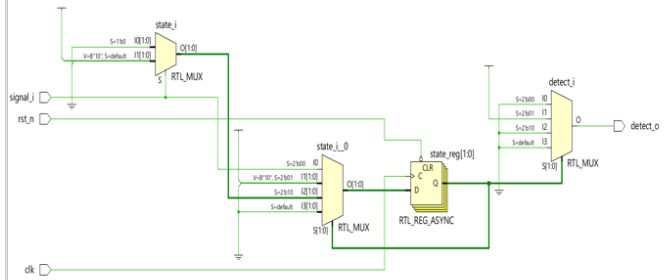
count[0]_i_1

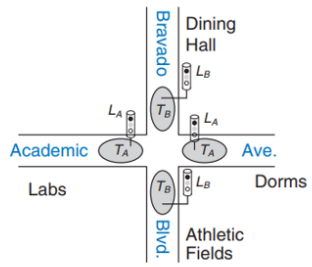
I0	O=!I0
0	1
1	0

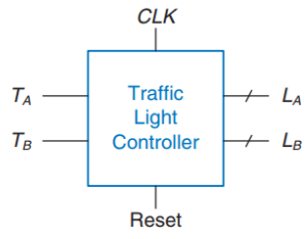
TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

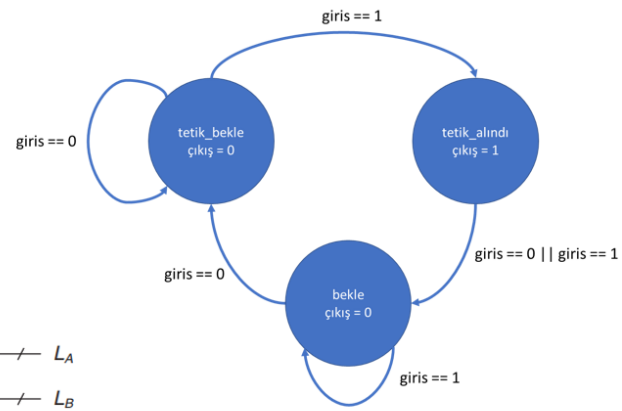
MANTIKSAL DEVRE TASARIMI (LOGIC DESIGN)

DERS14: FINITE STATE MACHINE DEVRE TASARIMI – VERILOG



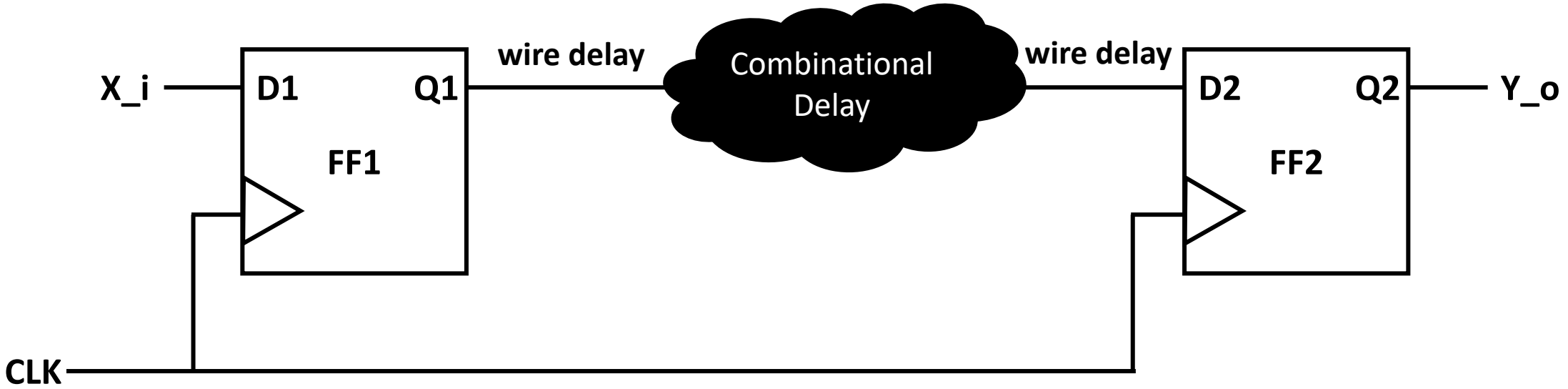






TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

STATIC TIMING ANALYSIS

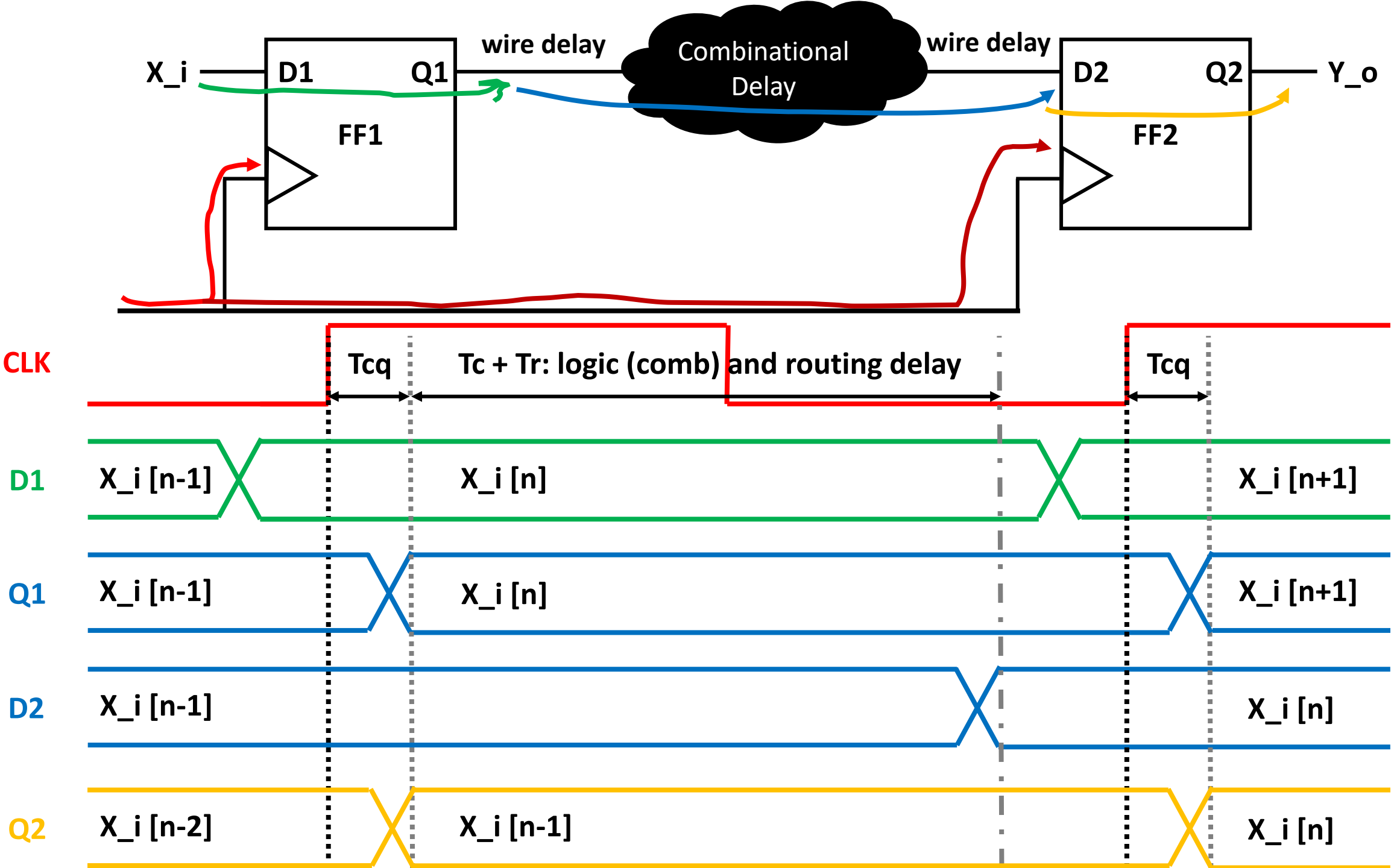


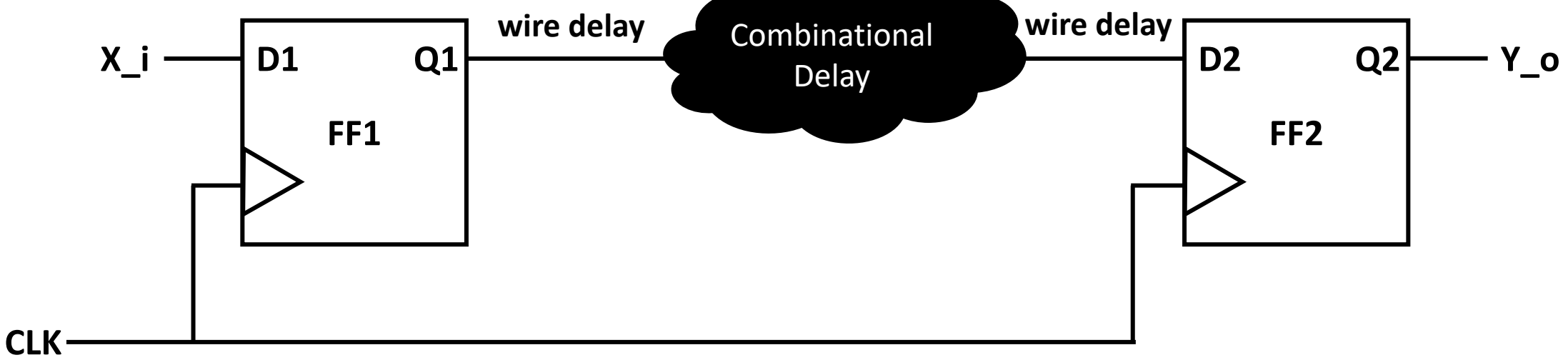
FF1: Kaynak (Source) Flip-Flop

FF2: Hedef (Destination) Flip-Flop

Combinational Delay: Sinyalin birleşik (combinational) devre elemanlarında ilerlerken meydana gelen gecikme

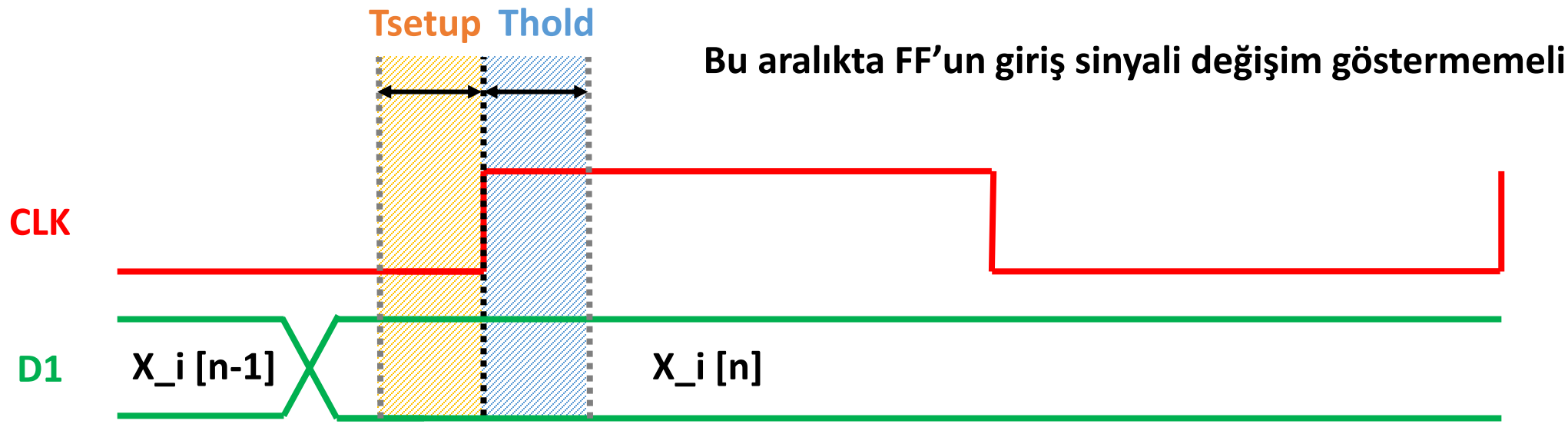
Wire Delay (Routing Delay): Sinyalin bağlantı kablosu, yolu üzerinde ilerlerken meydana gelen gecikme


X_i
D1
Q1
FF1
wire delay
Combinational
Delay
wire delay
D2
Q2
Y_o
FF2
CLK
Tcq
Tc + Tr: logic (comb) and routing delay
Tcq
D1
X_i [n-1]
X_i [n]
X_i [n+1]
Q1
X_i [n-1]
X_i [n]
X_i [n+1]
D2
X_i [n-1]
X_i [n]
Q2
X_i [n-2]
X_i [n-1]
X_i [n]



Setup Time (Tsetup): Flip-Flop çıkış sinyalinin tutarlı olması için, Flip-Flop giriş sinyalinin (D1) saat sinyalinin yükselen kenarından <u>önce</u> beklemesi gereken süre.

Hold Time (Thold): Flip-Flop çıkış sinyalinin tutarlı olması için, Flip-Flop giriş sinyalinin (D1) saat sinyalinin yükselen kenarından <u>sonra</u> beklemesi gereken süre.



STATIC TIMING ANALYSIS

SETUP TIME ANALİZİ - MAX FREKANS FORMÜLÜ

Tperiod – Tsetup > Tcq + Tcomb
Tperiod > Tcq + Tcomb + Tsetup
Period (min) = Tcq + Tcomb + Tsetup
Frequency (max) = 1 / (Tcq + Tcomb + Tsetup)

Slack = Tperiod – (Tcq + Tcomb + Tsetup)

Positive Slack 

Negative Slack 

HOLD TIME ANALİZİ

Tcq + Tcomb > Thold
if (Tcomb == 0) “Back to back 2 FF durumu”
Tcq > Thold

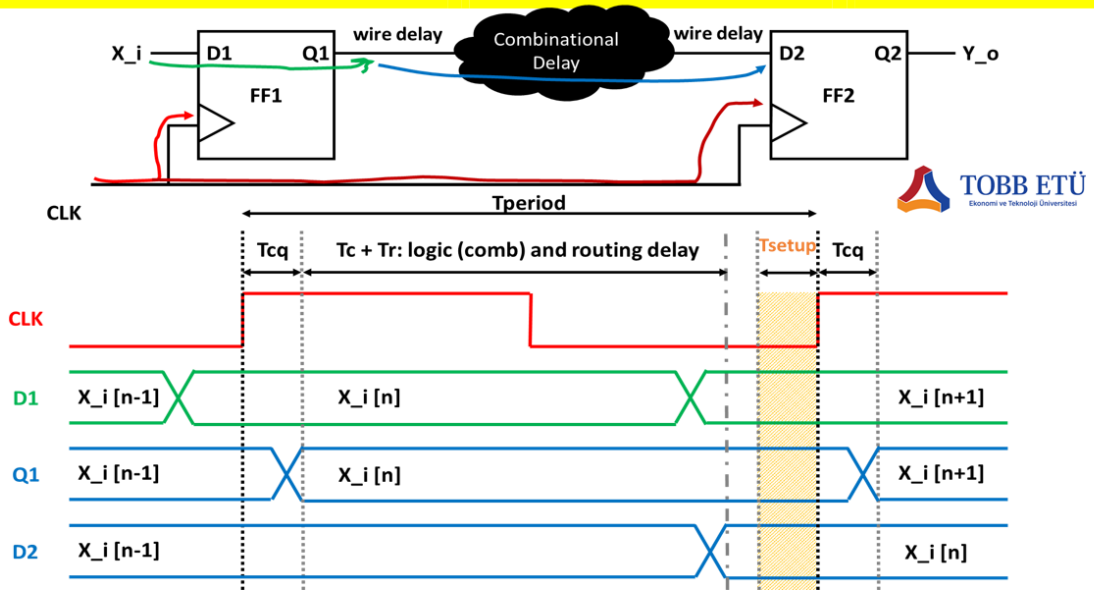
Slack = Tcq + Tcomb - Thold

Positive Slack 

Negative Slack 

MANTIKSAL DEVRE TASARIMI (LOGIC DESIGN)

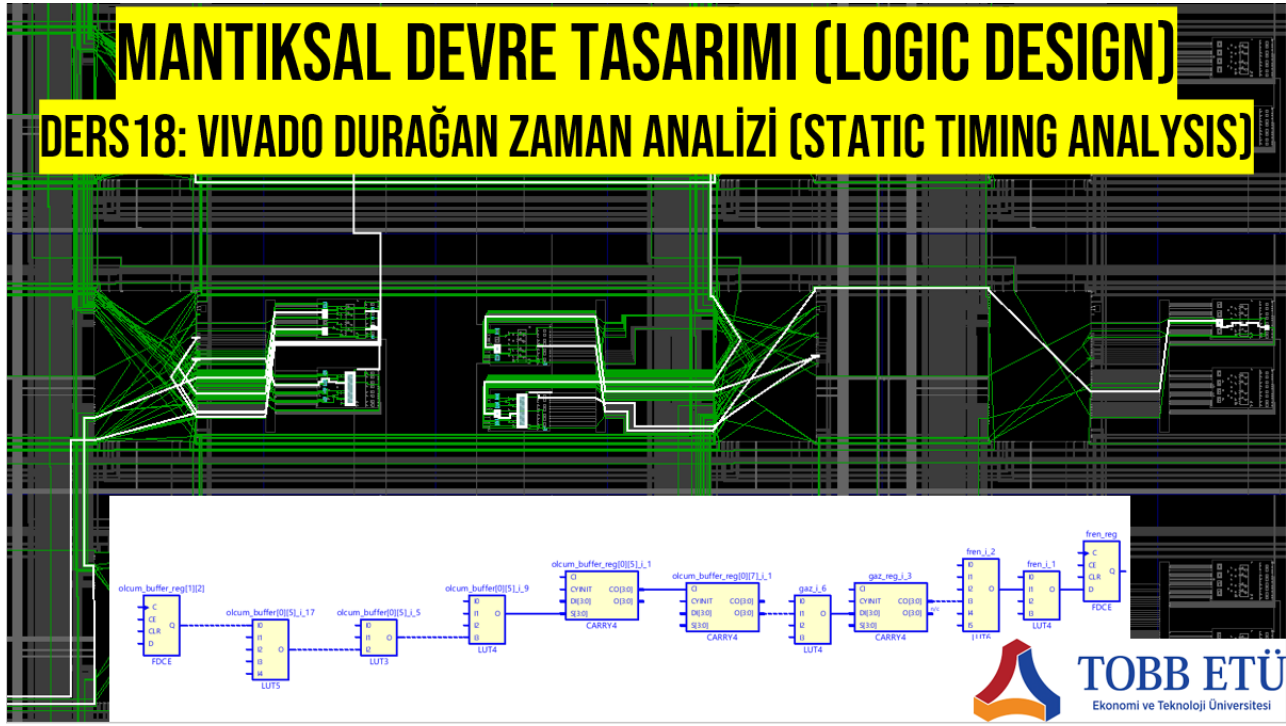
DERS17: DURAĞAN ZAMAN ANALİZİ – STATIC TIMING ANALYSIS – SETUP & HOLD TIME



TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

MANTIKSAL DEVRE TASARIMI (LOGIC DESIGN)

DERS18: VIVADO DURAĞAN ZAMAN ANALİZİ (STATIC TIMING ANALYSIS)



TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi