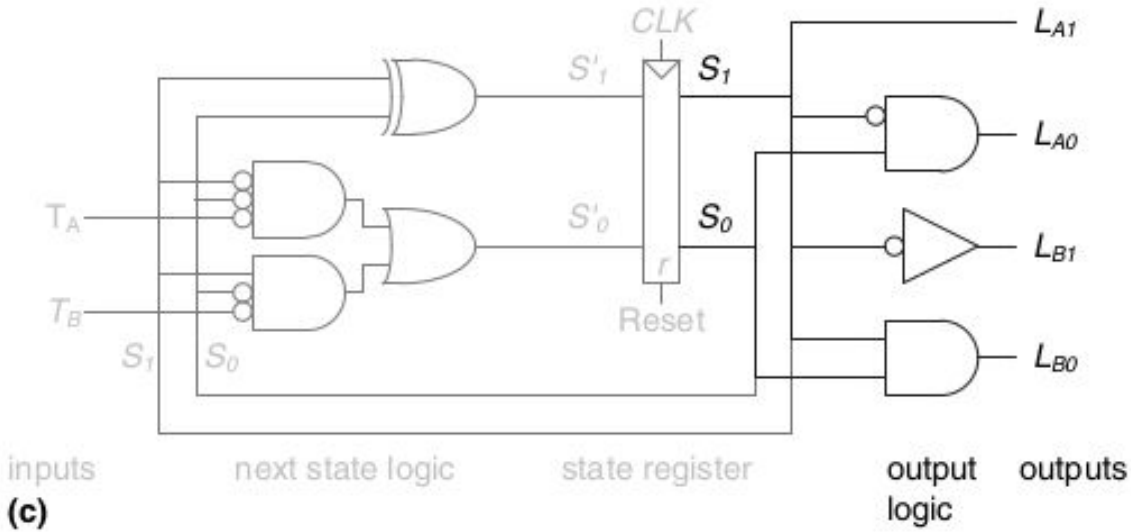
Donanım Tanımlama Dilleri

Suhap SAHIN

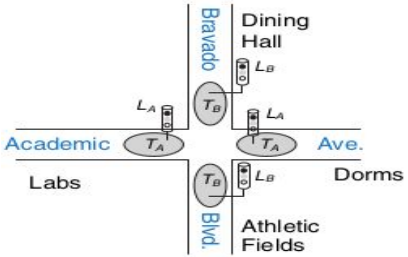
4.1 Giris

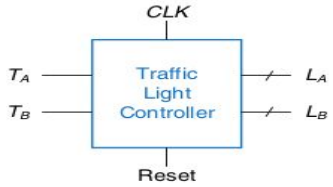
Sematik Tasarım



(c)

4.1 Giris





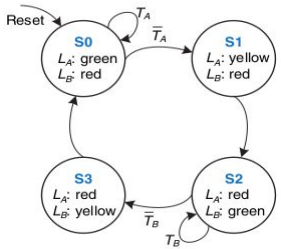
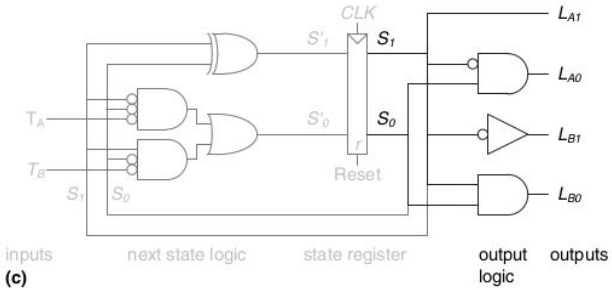


Table 3.1 State transition table

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



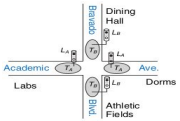
(c)

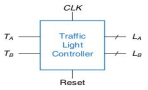
$$L_{A1} = S_1$$
$$L_{A0} = \overline{S}_1 S_0$$
$$L_{B1} = \overline{S}_1$$
$$L_{B0} = S_1 S_0$$

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

4.1 Giris





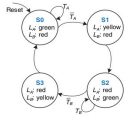
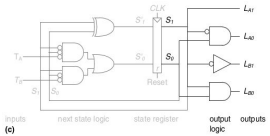


Table 3.1 State transition table

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



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

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

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
entity TLC is
  Port (
       Trafficlights: out STD_LOGIC_Vector (5 downto 0);
       Clck : in STD_LOGIC; Reset : in STD_LOGIC;
       P_B : in STD_LOGIC); end TLC;
architecture Behavioral of TLC is
    type state_type is (st0_R1_G2, st1_R1_A1_A2, st2_G1_R2,
st3_A1_R2_A2);
    signal state: state_type;
    signal count : std_logic_vector (3 downto 0);
    constant sec10 : std_logic_vector ( 3 downto 0) := "1010";
    constant sec2 : std_logic_vector (3 downto 0 ) := "0010";
    constant sec16: std_logic_vector (3 downto 0 ) := "1111";
begin
    process (Clck,Reset)
    begin
       if Reset='1' then
            state <= st0_R1_G2;
            count <= X"0";
       elsif Clck' event and Clck = '1' then
            case (state) is
...
...
...
```

4.1 Giris

SystemVerilog

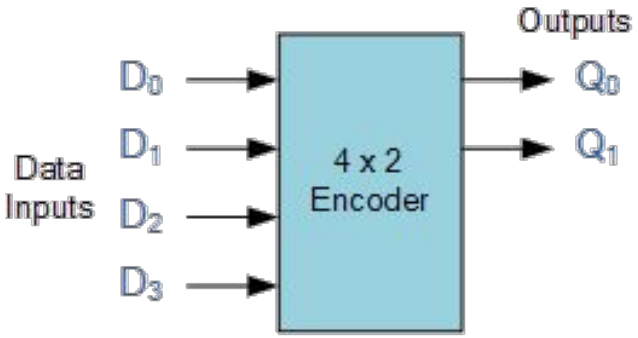
```
module sillyfunction(       input logic a, b, c,
                                    output logic y);
            assign y =      ~a & ~b & ~c |
                             a & ~b & ~c |
                             a & ~b & c;
endmodule
```

VHDL

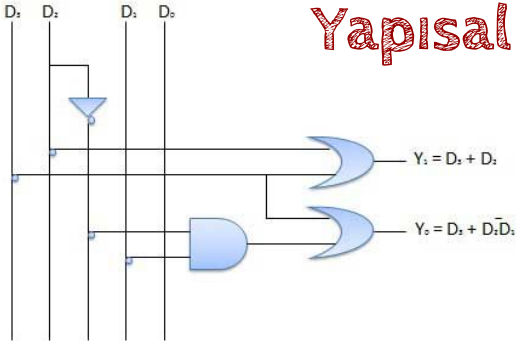
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sillyfunction is
        port( a, b, c: in STD_LOGIC;
                y: out STD_LOGIC);
end;

architecture synth of sillyfunction is
begin
        y <=    (not a and not b and not c) or
                (a and not b and not c) or
                (a and not b and c);
end;
```

4.1.1 Giris

Modul



Yapısal Modelleme



Davranıssal Modelleme

Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

Örnek 4.1. Birlesik Mantık

```
module sillyfunction(   input logic a, b, c,
                        output logic y);
    assign y = ~a & ~b & ~c |
               a & ~b & ~c |
               a & ~b & c;
endmodule
```

$y = a'b'c' + ab'c' + ab'c$

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sillyfunction is
    port( a, b, c: in STD_LOGIC;
          y: out STD_LOGIC);
end;

architecture synth of sillyfunction is
begin
    y <=  (not a and not b and not c) or
          (a and not b and not c) or
          (a and not b and c);
end;
```

Örnek 4.1. Birlesik Mantık

```
module sillyfunction(   input logic a, b, c,
                        output logic y);
        assign y = ~a & ~b & ~c |
                    a & ~b & ~c |
                    a & ~b & c;
endmodule
```

$$y = a'b'c' + ab'c' + ab'c$$

Örnek 4.1. Birlesik Mantık

```
module sillyfunction(   input logic a, b, c,
                        output logic y);
    assign y = ~a & ~b & ~c |
               a & ~b & ~c |
               a & ~b & c;
endmodule
```

$y = a'b'c' + ab'c' + ab'c$

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sillyfunction is
    port( a, b, c: in STD_LOGIC;
          y: out STD_LOGIC);
end;

architecture synth of sillyfunction is
begin
    y <=  (not a and not b and not c) or
          (a and not b and not c) or
          (a and not b and c);
end;
```

Örnek 4.1. Birlesik Mantık

```
module sillyfunction(   input logic a, b, c,
                        output logic y);
    assign y = ~a & ~b & ~c |
               a & ~b & ~c |
               a & ~b & c;
endmodule
```

$y = a'b'c' + ab'c' + ab'c$

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sillyfunction is
    port( a, b, c: in STD_LOGIC;
          y: out STD_LOGIC);
end;

architecture synth of sillyfunction is
begin
    y <=  (not a and not b and not c) or
          (a and not b and not c) or
          (a and not b and c);
end;
```

4.1.2 Dilin Kökeni

```
module sillyfunction(   input logic a, b, c,
                        output logic y);
        assign y = ~a & ~b & ~c |
                    a & ~b & ~c |
                    a & ~b & c;
endmodule
```

$y = a'b'c' + ab'c' + ab'c$

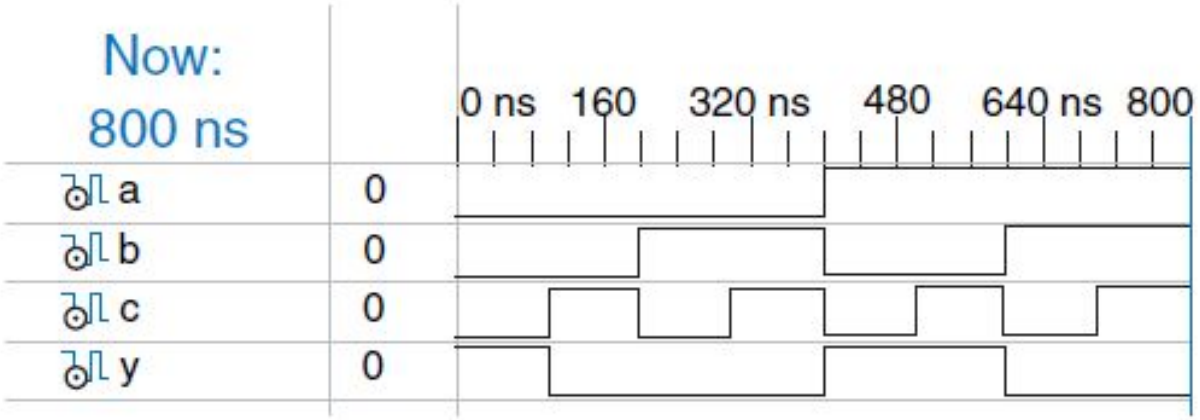
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sillyfunction is
        port( a, b, c: in STD_LOGIC;
              y: out STD_LOGIC);
end;

architecture synth of sillyfunction is
begin
        y <=  (not a and not b and not c) or
              (a and not b and not c) or
              (a and not b and c);
end;
```

4.1.2 Dilin Kökeni

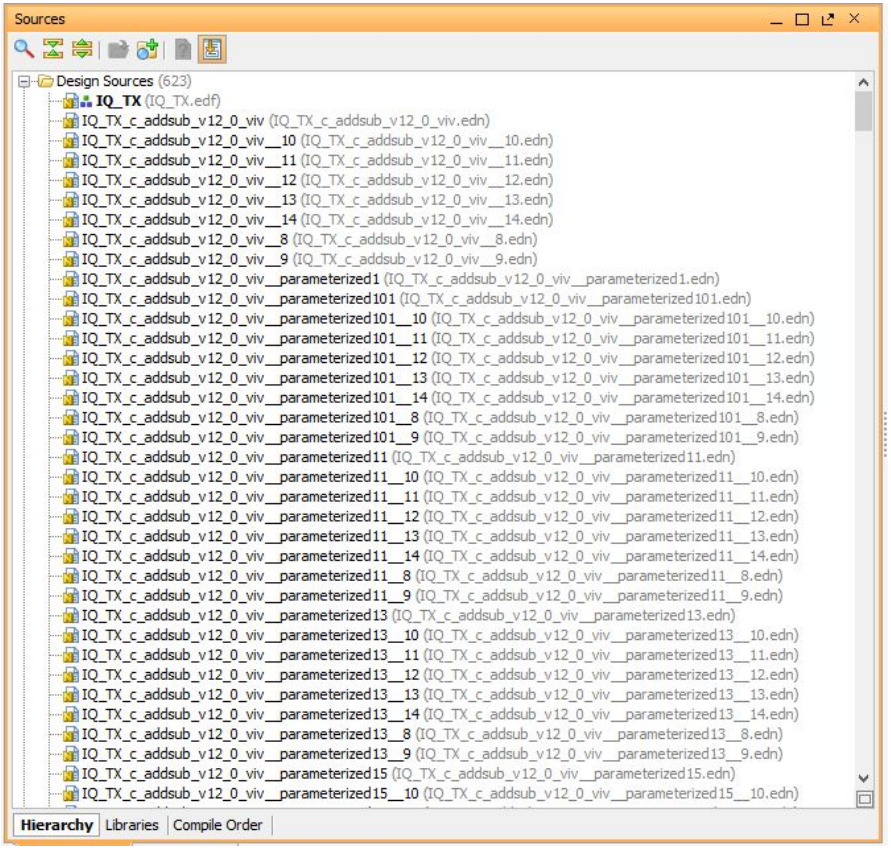
```
module sillyfunction(   input logic a, b, c,
                        output logic y);
    assign y = ~a & ~b & ~c |
               a & ~b & ~c |
               a & ~b & c;
endmodule
```

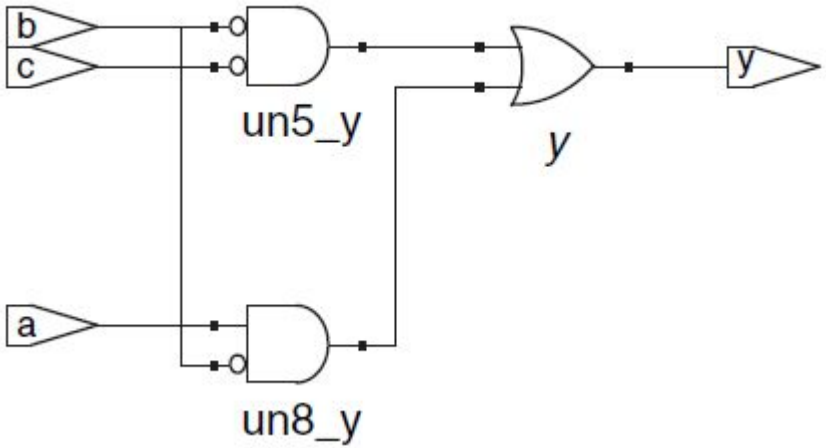
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sillyfunction is
    port( a, b, c: in STD_LOGIC;
          y: out STD_LOGIC);
end;

architecture synth of sillyfunction is
begin
    y <=  (not a and not b and not c) or
          (a and not b and not c) or
          (a and not b and c);
end;
```

4. 1. 3 Benzetim ve Sentez



4. 1. 3 Benzetim ve Sentez





4. 1. 3 Benzetim ve Sentez

testbech

```
   -- set clock to 20MHz and run
   InByte <= "0000"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 200 ns;
   -- set clock to 10MHz and run
   InByte <= "0001"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 200 ns;
   -- set clock to 4MHz and run
   InByte <= "0010"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 500 ns;
   -- set clock to 2MHz and run
   InByte <= "0011"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 1 us;
   -- set clock to 1MHz and run
   InByte <= "0100"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 2 us;
   -- set clock to 400KHz and run
   InByte <= "0101"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 5 us;

   wait; -- will wait forever
END PROCESS;
```

HDL

```
When "01" =>                                          -- 2 Way
   case SizeSel is
      when "00" =>                                    -- 64K
         if D_Set0_in = '1' and D_Set1_in = '1' then
               if lruin(0) = '0' then
                  D_Set0_out <= '1' ;
                  D_Set1_out <= '0' ;
                  D_Set2_out <= '0' ;
                  D_Set3_out <= '0' ;
                  D_Set4_out <= '0' ;
                  D_Set5_out <= '0' ;
                  D_Set6_out <= '0' ;
                  D_Set7_out <= '0' ;
                  Lruout <= "0111111" ;
               else
                  D_Set0_out <= '0' ;
                  D_Set1_out <= '1' ;
                  D_Set2_out <= '0' ;
                  D_Set3_out <= '0' ;
                  D_Set4_out <= '0' ;
                  D_Set5_out <= '0' ;
                  D_Set6_out <= '0' ;
                  D_Set7_out <= '0' ;
                  Lruout <= "0111111" ;
               end if;
```

4. 1. 3 Benzetim ve Sentez

testbech

```
   -- set clock to 20MHz and run
   InByte <= "0000"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 200 ns;
   -- set clock to 10MHz and run
   InByte <= "0001"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 200 ns;
   -- set clock to 4MHz and run
   InByte <= "0010"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 500 ns;
   -- set clock to 2MHz and run
   InByte <= "0011"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 1 us;
   -- set clock to 1MHz and run
   InByte <= "0100"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 2 us;
   -- set clock to 400KHz and run
   InByte <= "0101"; RegSel <= "01"; RegStrb <= '0';
   wait for 50 ns;
   RegStrb <= '1';
   wait for 5 us;

   wait; -- will wait forever
END PROCESS;
```

HDL

```
When "01" =>                                          -- 2 Way
   case SizeSel is
      when "00" =>                                    -- 64K
         if D_Set0_in = '1' and D_Set1_in = '1' then
               if lruin(0) = '0' then
                  D_Set0_out <= '1' ;
                  D_Set1_out <= '0' ;
                  D_Set2_out <= '0' ;
                  D_Set3_out <= '0' ;
                  D_Set4_out <= '0' ;
                  D_Set5_out <= '0' ;
                  D_Set6_out <= '0' ;
                  D_Set7_out <= '0' ;
                  Lruout <= "0111111" ;
               else
                  D_Set0_out <= '0' ;
                  D_Set1_out <= '1' ;
                  D_Set2_out <= '0' ;
                  D_Set3_out <= '0' ;
                  D_Set4_out <= '0' ;
                  D_Set5_out <= '0' ;
                  D_Set6_out <= '0' ;
                  D_Set7_out <= '0' ;
                  Lruout <= "0111111" ;
               end if;
```

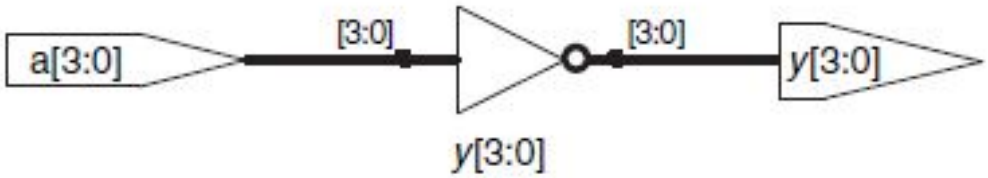
4. 2. Birlesimsel Mantık

Kombinasyonel mantık ve saklayıcılardan oluşan eşzamanlı sıralı devreler tasarlamak için kendimizi disipline ettiğimizi hatırlayın. Kombinasyonel mantığın çıktıları yalnızca akım girişlerine bağlıdır. Bu bölümde, HDL'lerle kombinasyonel mantığın davranışsal modellerinin nasıl yazılacağı açıklanmaktadır.

4.2.1. Bit islemleri

```
module inv(input logic [3:0] a,
                output logic [3:0] y);
        assign y = ~a;
endmodule
```

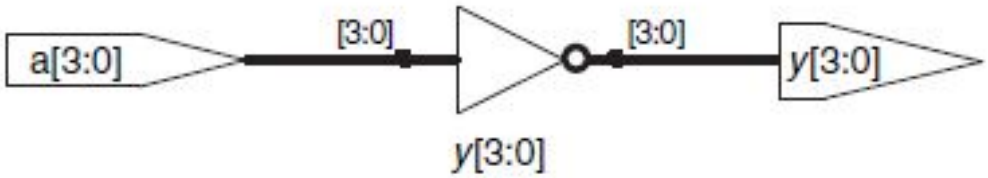
```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity inv is
        port(a: in STD_LOGIC_VECTOR(3 downto 0);
                y: out STD_LOGIC_VECTOR(3 downto 0));
end;
architecture synth of inv is
begin
        y <= not a;
end;
```



4.2.1. Bit islemleri

```
module inv(input logic [3:0] a,
                output logic [3:0] y);
        assign y = ~a;
endmodule
```

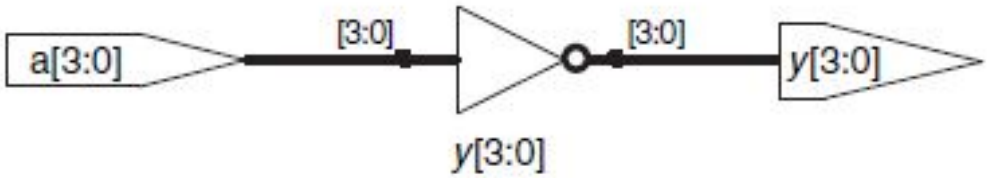
```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity inv is
        port(a: in STD_LOGIC_VECTOR(3 downto 0);
                y: out STD_LOGIC_VECTOR(3 downto 0));
end;
architecture synth of inv is
begin
        y <= not a;
end;
```



4.2.1. Bit islemleri

```
module inv(input logic [3:0] a,
                output logic [3:0] y);
        assign y = ~a;
endmodule
```

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity inv is
        port(a: in STD_LOGIC_VECTOR(3 downto 0);
                y: out STD_LOGIC_VECTOR(3 downto 0));
end;
architecture synth of inv is
begin
        y <= not a;
end;
```

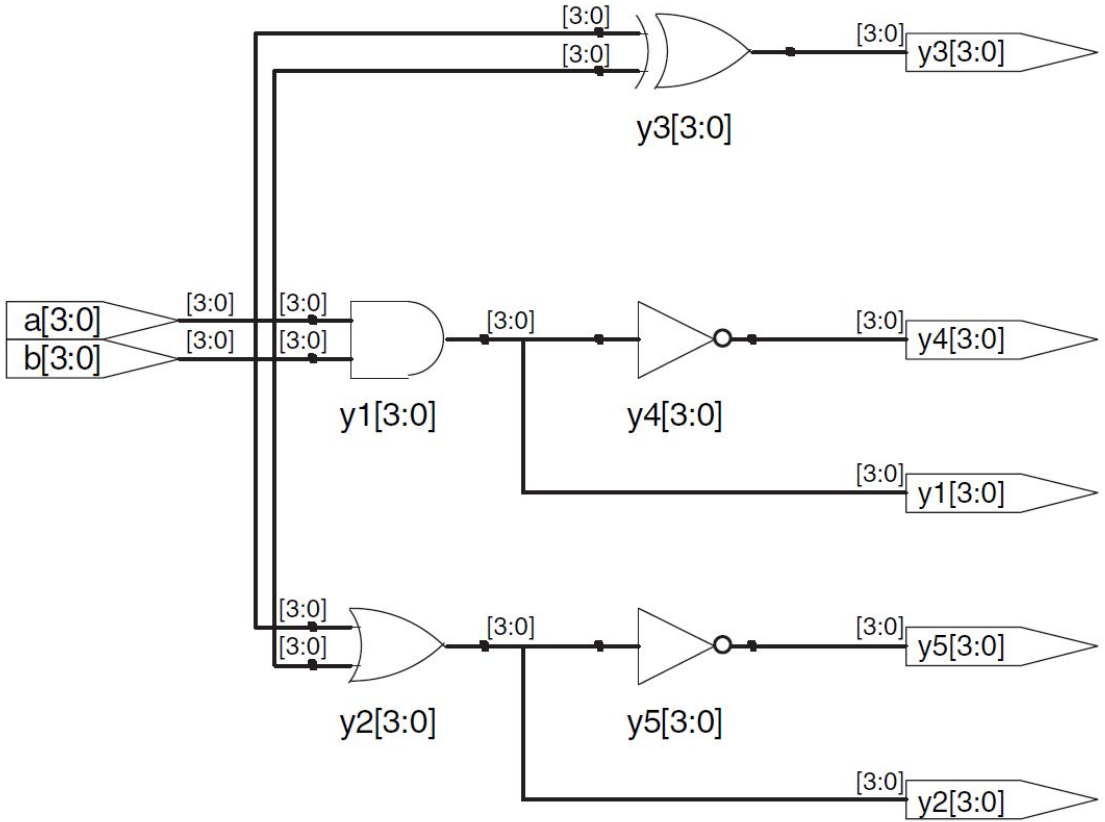


Örnek 4.3 Mantık Kapıları

```
module gates(input logic [3:0] a, b,
             output logic [3:0] y1, y2,
             y3, y4, y5);
  assign y1 = a & b;     // AND
  assign y2 = a | b;     // OR
  assign y3 = a ^ b;     // XOR
  assign y4 = ~(a & b); // NAND
  assign y5 = ~(a | b);  // NOR
endmodule
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity gates is
  port(  a, b: in STD_LOGIC_VECTOR(3 downto 0);
         y1, y2, y3, y4,
         y5: out STD_LOGIC_VECTOR(3 downto 0));
end;
architecture synth of gates is
begin
  y1 <= a and b;
  y2 <= a or b;
  y3 <= a xor b;
  y4 <= a nand b;
  y5 <= a nor b;
end;
```

Örnek 4.3 Mantık Kapıları



4. 2. 2 Yorum ve Beyaz Bosluk

WHITE SPACE

SystemVerilog / VHDL:
Beyaz bosluk (bosluklar, sekmeler ve satır sonları) kullanımı konusunda seçici degildir.

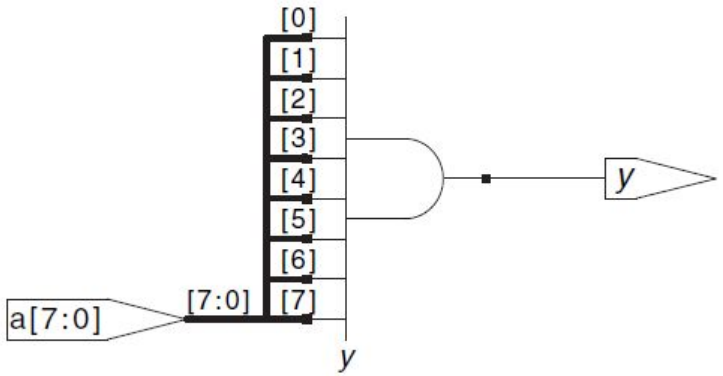
SystemVerilog

/ * Çoklu yorum satırları * /
// Tek satır yorumu
SystemVerilog büyük / küçük harfe duyarlıdır. y1 ve Y1, SystemVerilog'daki farklı sinyallerdir.

VHDL
/ * Çoklu yorum satırları * /
-- Tek satır yorumu
VHDL, büyük / küçük harfe duyarlı degildir. y1 ve Y1, VHDL'de aynı sinyaldir.

4. 2. 3 Indirgeme Operatörü

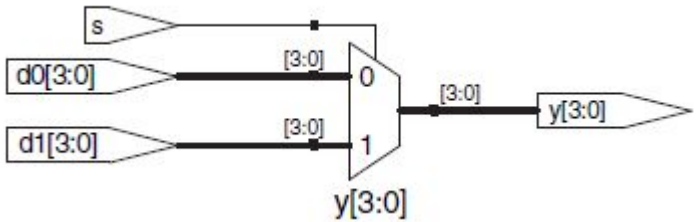
```
module and8(input logic [7:0] a,
                output logic y);
        assign y = &a;
        // &a is much easier to write than
        // assign y = a[7] & a[6] & a[5] & a[4] &
        // a[3] & a[2] & a[1] & a[0];
endmodule
```




```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity and8 is
port(a: in STD_LOGIC_VECTOR(7 downto 0);
        y: out STD_LOGIC);
end;

architecture synth of and8 is
begin
        y <= and a;
        -- and a is much easier to write than
        -- y <= a(7) and a(6) and a(5) and a(4) and
        -- a(3) and a(2) and a(1) and a(0);
end;
```

4. 2. 4 Kosullu Atama

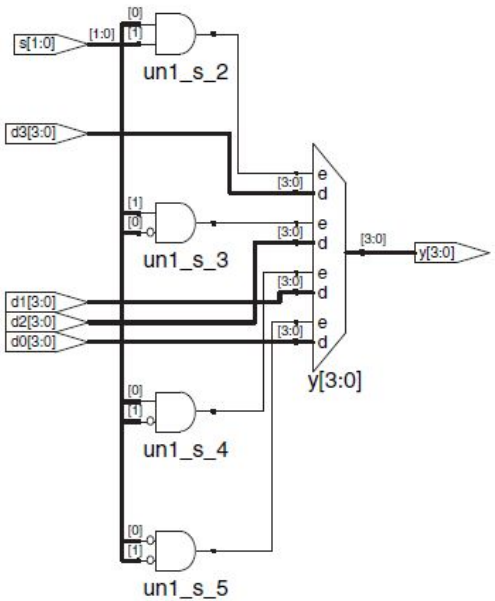
```
module mux2(    input logic [3:0] d0, d1,
                input logic s,
                output logic [3:0] y);
                assign y =s ? d1 : d0;
endmodule
```




```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux2 is
        port( d0, d1: in STD_LOGIC_VECTOR(3 downto 0);
                s: in STD_LOGIC;
                y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of mux2 is
begin
        y <= d1 when s else d0;
end;
```

4. 2. 4 Kosullu Atama

```
module mux4(    input logic [3:0] d0, d1, d2, d3,
                input logic [1:0] s,
                output logic [3:0] y);
      assign y = s[1]   ?      (s[0] ? d3 : d2)
                        :      (s[0] ? d1 : d0);
endmodule
```




```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity mux4 is
        port( d0, d1,
                d2, d3: in STD_LOGIC_VECTOR(3 downto 0);
                s: in STD_LOGIC_VECTOR(1 downto 0);
                y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth1 of mux4 is
begin
        y <=  d0 when s = "00" else
                d1 when s = "01" else
                d2 when s = "10" else
                d3;
end;
```

4. 2. 5 dahili degiskenler

```
module fulladder(input logic a, b, cin,
                 output logic s, cout);
    logic p, g;
    assign p =a ^ b;
    assign g =a & b;
    assign s = p ^ cin;
    assign cout = g | (p & cin);
endmodule
```

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity fulladder is
    port( a, b, cin: in STD_LOGIC;
          s, cout: out STD_LOGIC);
end;

architecture synth of fulladder is
    signal p, g: STD_LOGIC;
begin
    p <= a xor b;
    g <= a and b;

    s <= p xor cin;
    cout <= g or (p and cin);
end;
```

4.2.6 Öncelik

SystemVerilog

Table 4.1 SystemVerilog operator precedence

	Op	Meaning
Highest	~	NOT
	*, /, %	MUL, DIV, MOD
	+, -	PLUS, MINUS
	<<, >>	Logical Left/Right Shift
	<<<, >>>	Arithmetic Left/Right Shift
	<, <=, >, >=	Relative Comparison
	==, !=	Equality Comparison
	&, ~&	AND, NAND
	^, ~^	XOR, XNOR
	|, ~|	OR, NOR
Lowest	?:	Conditional

VHDL

Table 4.2 VHDL operator precedence

	Op	Meaning
Highest	not	NOT
	*, /, mod, rem	MUL, DIV, MOD, REM
	+, -	PLUS, MINUS
	rol, ror, srl, sll	Rotate, Shift logical
	<, <=, >, >=	Relative Comparison
	=, /=	Equality Comparison
Lowest	and, or, nand, nor, xor, xnor	Logical Operations

4.2.7 Numaralar

Table 4.3 SystemVerilog numbers

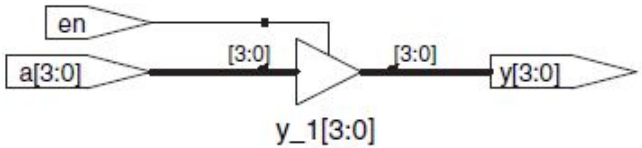
Numbers	Bits	Base	Val	Stored
3'b101	3	2	5	101
'b11	?	2	3	000 ... 0011
8'b11	8	2	3	00000011
8'b1010_1011	8	2	171	10101011
3'd6	3	10	6	110
6'o42	6	8	34	100010
8'hAB	8	16	171	10101011
42	?	10	42	00 ... 0101010

Table 4.4 VHDL numbers

Numbers	Bits	Base	Val	Stored
3B"101"	3	2	5	101
B"11"	2	2	3	11
8B"11"	8	2	3	00000011
8B"1010_1011"	8	2	171	10101011
3D"6"	3	10	6	110
6O"42"	6	8	34	100010
8X"AB"	8	16	171	10101011
"101"	3	2	5	101
B"101"	3	2	5	101
X"AB"	8	16	171	10101011

4. 2. 8 Z ve X degerleri

```
module tristate(  input logic [3:0] a,
                  input logic en,
                  output tri [3:0] y);

        assign y = en ? a : 4'bz;
endmodule
```




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity tristate is
        port( a: in STD_LOGIC_VECTOR(3 downto 0);
              en: in STD_LOGIC;
              y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of tristate is
begin
        y <= a when en else "ZZZZ";
end;
```

HDL Örnegi 4.11 TANIMLANMAMIS VE YÜZER GİRİSLERLE GERÇEK TABLOLAR

Table 4.5 SystemVerilog AND gate truth table with z and x

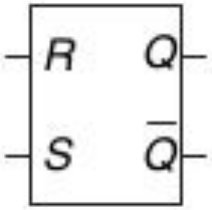
&		A			
		0	1	z	x
B	0	0	0	0	0
	1	0	1	x	x
	z	0	x	x	x
	x	0	x	x	x

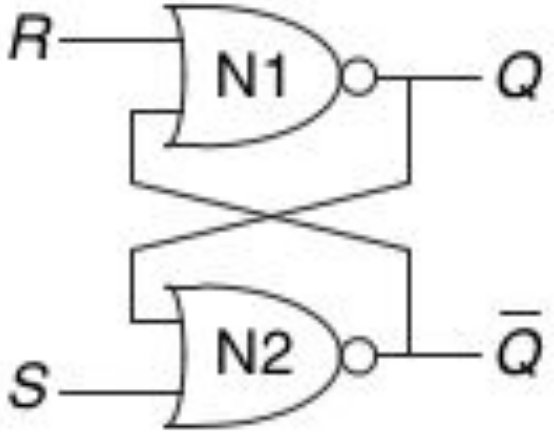
Table 4.6 VHDL AND gate truth table with z , x and u

AND		A				
		0	1	z	x	u
B	0	0	0	0	0	0
	1	0	1	x	x	u
	z	0	x	x	x	u
	x	0	x	x	x	u
	u	0	u	u	u	u

4 .2.9 Bit Swizzling

SR Latch





Case I: R=1, S=0

R=1 Q'=? --> N1 => FALSE

S=0 Q =0 --> N2 => TRUE

Case II: R=0, S=1

S=1 Q =? --> N2 => FALSE

R=0 Q'=0 --> N1 => TRUE

Case III: R=1, S=1

R=1 Q'=? --> N1 => FALSE

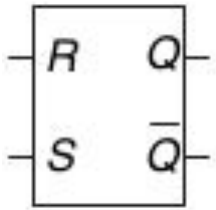
S=1 Q =? --> N2 => FALSE

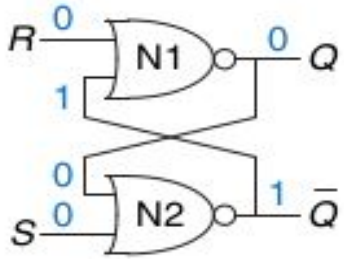
Case IV: R=0, S=0

R=0 Q'=? --> N1 => ?

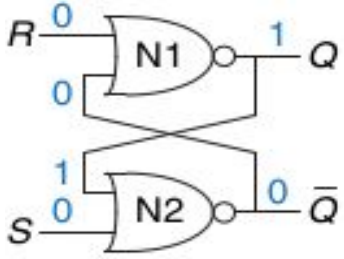
S=0 Q =? --> N2 => ?

SR Latch





(a)



(b)

Case IV: R=0, S=0

R=0 Q'=? --> N1 => ?

S=0 Q =? --> N2 => ?

Case IVa: Q=0

S=0 Q =0 --> N2 => TRUE

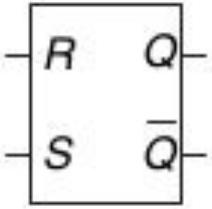
R=0 Q'=1 --> N1 => FALSE

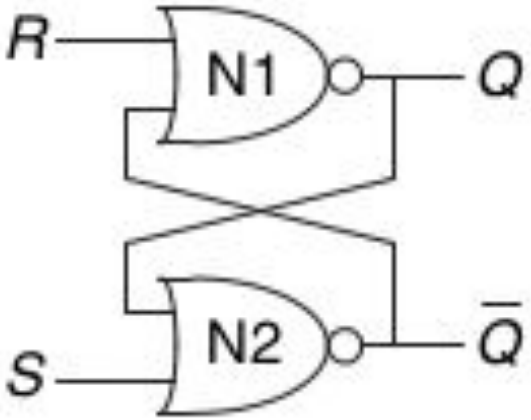
Case IVb: Q=1

S=0 Q =1 --> N2 => FALSE

R=0 Q'=0 --> N1 => TRUE

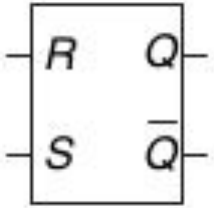
SR Latch



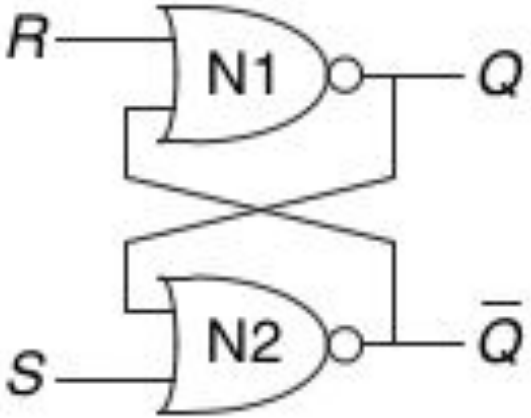


Case	S	R	Q	$\overline{Q}$
IV	0	0	Q_{prev}	$\overline{Q}_{prev}$
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0

SR Latch



R=1 ve S=1 oldugu durumda devre davranısı kestirilemez

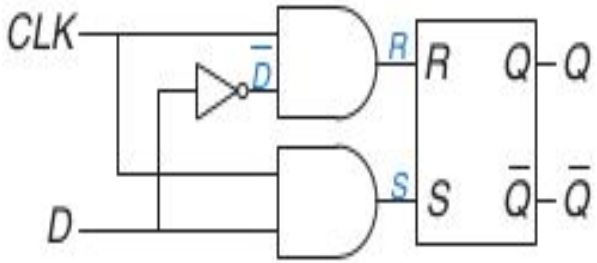


Case	S	R	Q	$\overline{Q}$
IV	0	0	Q_{prev}	$\overline{Q}_{prev}$
I	0	1	0	1
II	1	0	1	0
III	1	1	0	0

D Latch

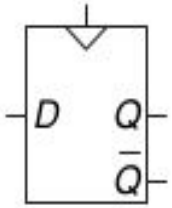
CLK = 1 ⇒ Latch is transparent

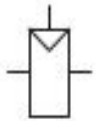
CLK = 0 ⇒ Latch is opaque



CLK	D	$\bar{D}$	S	R	Q	$\bar{Q}$
0	X	$\bar{X}$	0	0	Q_{prev}	$\bar{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

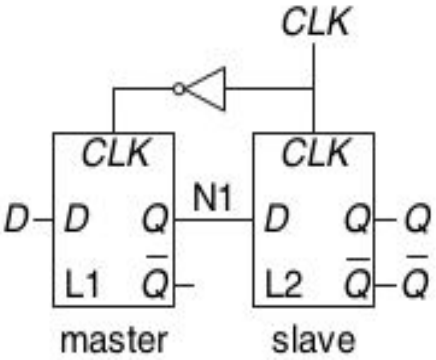
D Flip-Flop

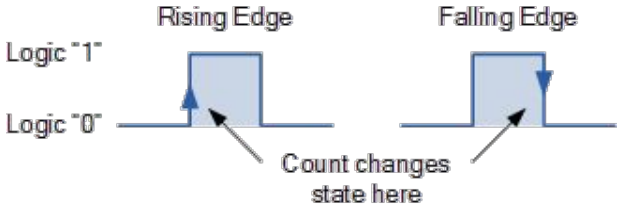




CLK = 0 ⇒ Master = transparent Slave = opaque

CLK = 1 ⇒ Master = opaque Slave = transparent





D tipi flip flop;

Yükselen kenarda D verisini çıkısa (Q) aktarır.

Aksi taktirde durumunu korur(hatırlar).

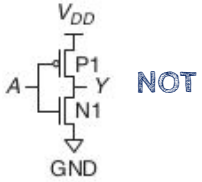
D tipi flip flop;

master-slave flip-flop

edge-triggered flip-flop

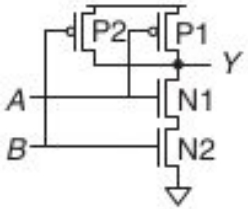
positive edge-triggered flip-flop

Örnek: D Flip-Flop Transistör sayısı

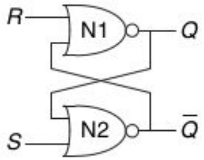


NOT

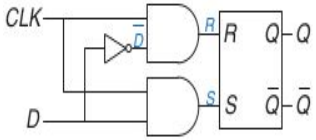
NAND



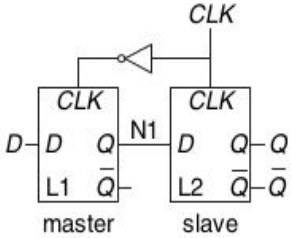
SR LATCH



D LATCH



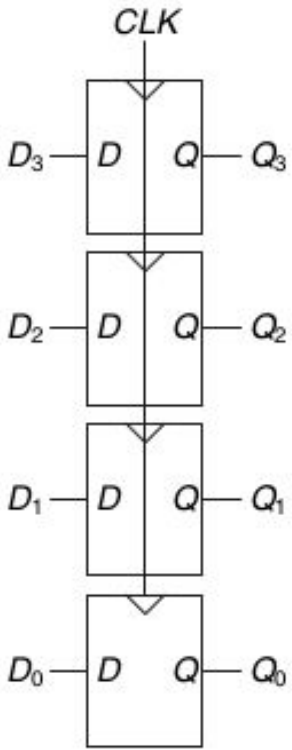
D Flip-Flop

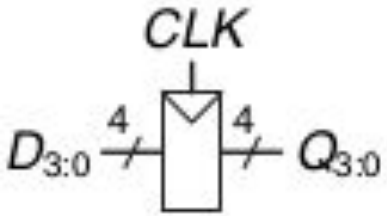


D tipi flip flop'taki transistör sayısı kaçtır?

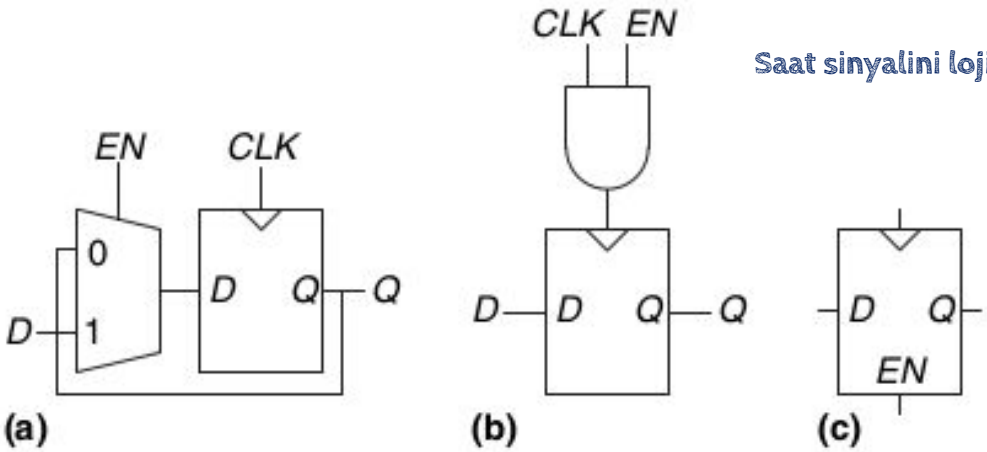
NAND / NOR	= 4 transistör		
NOT	= 2 transistör		
AND	= NAND + NOT	= 6 transistör	
SR LATCH	= 2xNOR	= 8 transistör	
D LATCH	= SR +2xAND + NOT	= 8+2x6+2	= 22 transistör
D flip-flop	= 2xD LATCH + NOT	= 2X22 + 2	= 46 transistör

Register / Saklayıcı





Enabled Flip-Flop



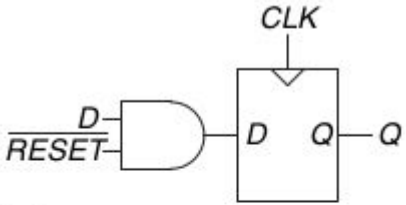
Saat sinyalini lojik bir islemden gecirmek kötü bir fikir

Verinin ne zaman kabul edilecegini belirleyen bir EN/ENABLE girisine sahiptir.

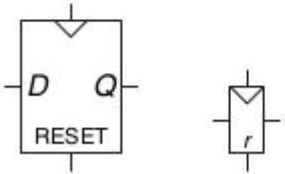
EN = TRUE ⇒ enable flip-flop = normal flip-flop

EN = FALSE ⇒ eski durum korunur

Resetlenebilir Flip-Flop



(a)



(b)

(c)

RESET = FALSE ⇒ enable flip-flop = normal flip-flop

RESET = TRUE ⇒ D = 0

Sistemde ilk calısmaya basladıgında;

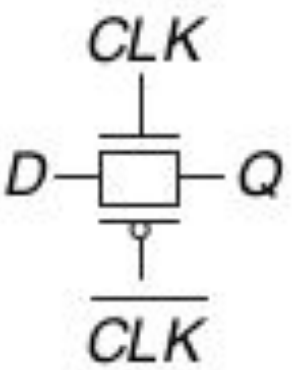
bütün flip-flop ları 0(baslangıc) durumuna getirmek için kullanılır.

Senkron resetlenebilir flip-flop: RESET girisi CLK sinyaline bagli

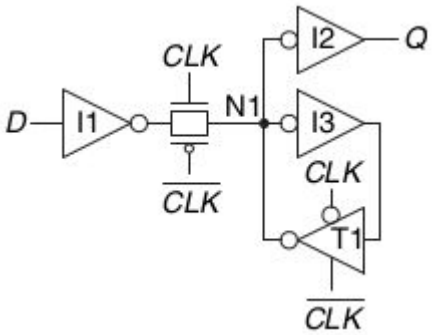
Asenkron resetlenebilir flip-flop: RESET girisi CLK sinyallinden bagımsız

Transistör Seviyesinde Latch ve Flip-Flop Tasarımı

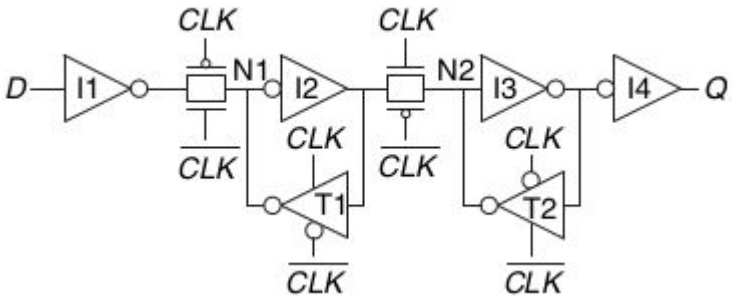
D latch



D latch

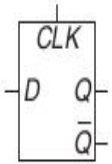


D flip flop



D Latch VS D Flip-Flop

D latch

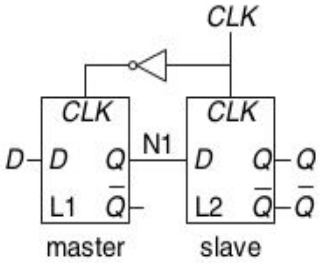


Level Triggering

CLK = 1 ⇒ Latch is transparent — D girisini Q çıkısına aktarır

CLK = 0 ⇒ Latch is opaque — Q önceki durumunu korur

D flip flop

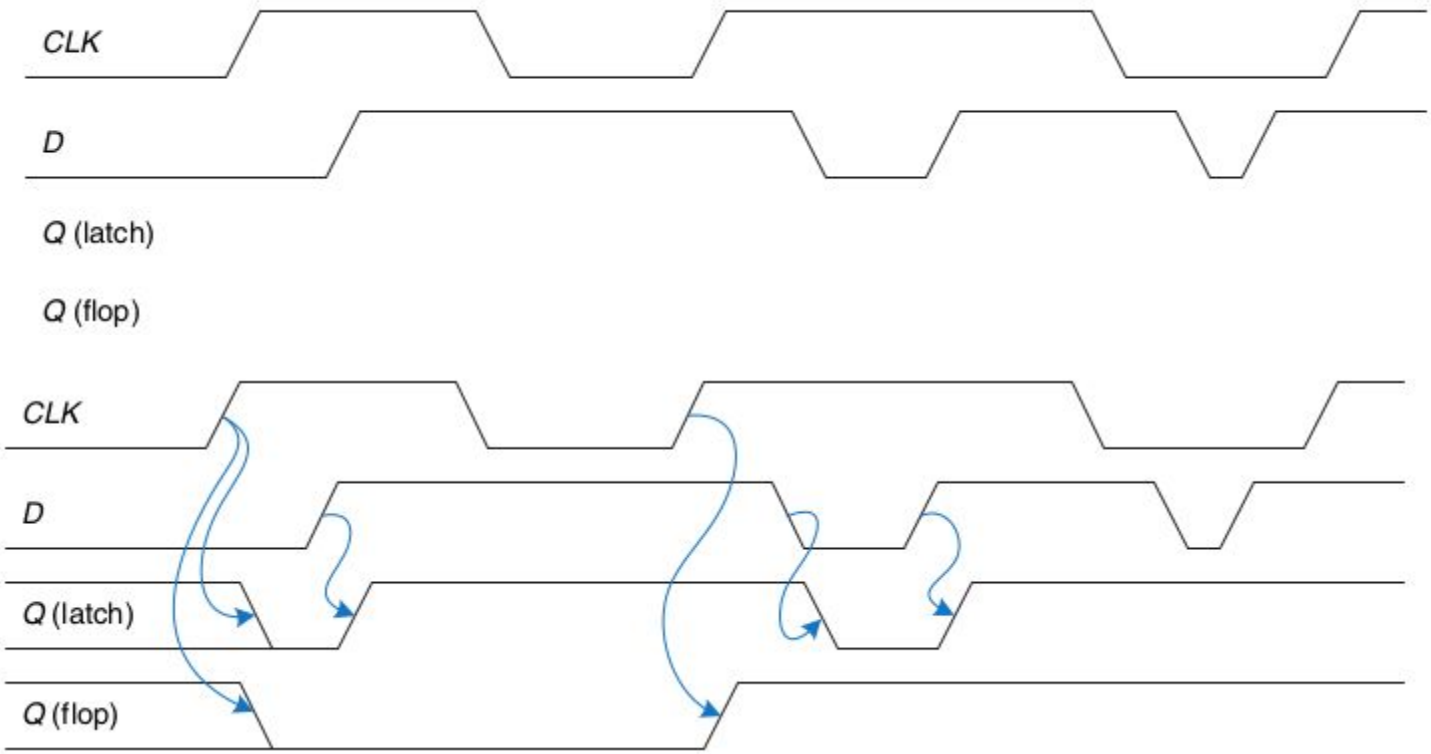


Positive Edge Triggering

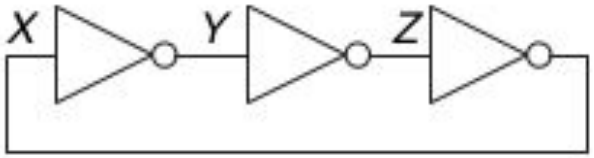
CLK = 0 ⇒ Master = transparent, Slave = opaque — D girisini Q çıkısına kopyalar

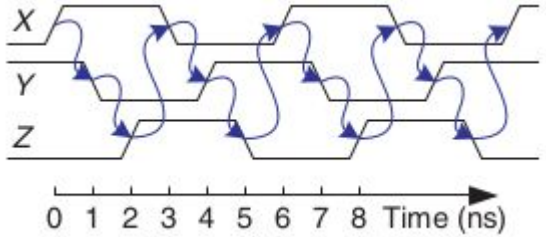
CLK = 1 ⇒ Master = opaque, Slave = transparent — Q önceki durumunu korur

D Latch VS D Flip-Flop



Senkron Lojik Tasarım - Halka Ösilatörü -





Inverter lerin yayılma gecikmesi 1ns

X=0 kabul edilirse;
Y=1 , Z=0 , X=1 (Kabul ile tutarsız olur)

0ns => X yükselir 1ns = > Y düser 2ns => Z yükselir ,
3ns => X düser 4ns => Y yükselir 5ns => Z düser
6ns => X yükselir

Yukarıdaki desen her 6 ns' de bir tekrar eder.
Bu devreye halka osilatörü denir.

Halka osilatörünün süresi = her invertörün yayılma gecikmesi
Gecikme = sürücünün üretimi, güç kaynagı voltajı

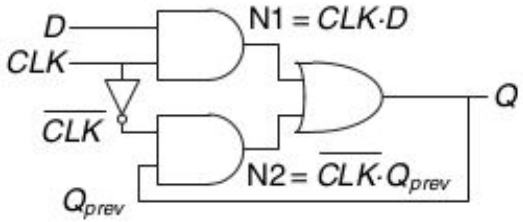
Halka osilatör periyodunun dogru bir sekilde tahmin edilmesi zordur.
Halka osilatörü, sıfır girisli ve periyodik olarak degisen bir çıkıslı sıralı bir devredir.

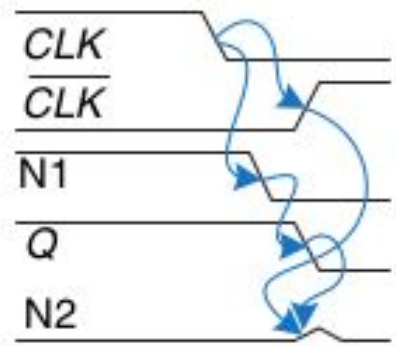
Senkron Lojik Tasarım - Yarış Durumu -

$$Q = CLK \cdot D + \overline{CLK} \cdot Q_{prev}$$

Ali az sayıda kapı kullanılarak, daha basarılı bir D Latch tasarladıgını idda ediyor.

CLK	D	Q_{prev}	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1





CLK = D = 1 => Q=1 (transparent)

CLK = düser => Q=1(eski degerini hatırlar)

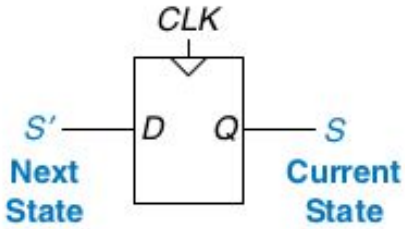
CLK dan CLK' gecis gecikmesinin göreceli olarak daha uzun oldugu varsayılsın

CLK' = yükselir => N1 ve Q; CLK' dan önce düser

Bu durumda;

N2 kesinlikle yükselemeyecektir.

Senkron Ardısık Devreler

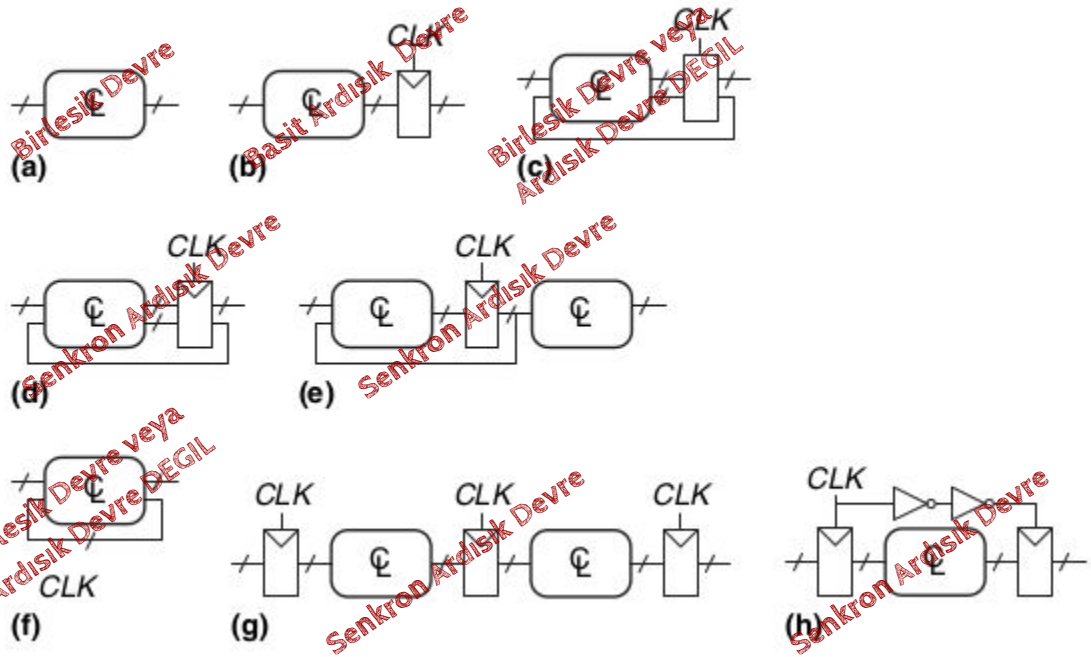


- Herbir elemanı saklayıcı veya birlesik devre
- En az bir elemanı saklayıcı
- Bütün saklayıcılar aynı saat sinyalini kullanılır
- Herbir döngüsel yol bir sakalayıcı(döngüyü bekletebilen) içerir

Bir girise(D) bir saat sinyaline(CLK), bir çıkısa (Q) ve iki duruma {0, 1} sahip; Bir flip-flop, en basit senkron ardısık devredir.
Mevcut durum S, sonraki durum ise S' ile ifade edilmistir.

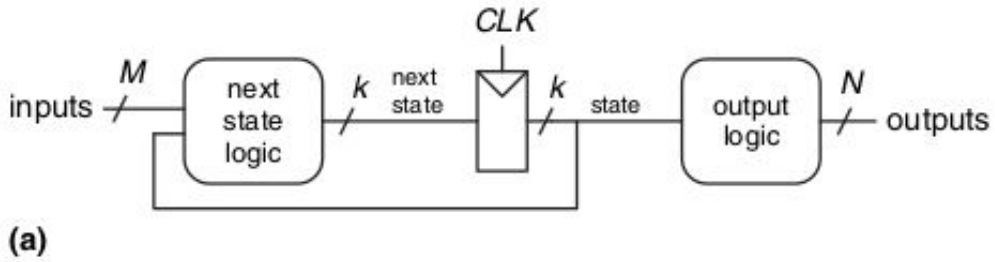
Senkron Ardısık Devreler

Asagıdalilerden hagisi senkron ardısık devredir.



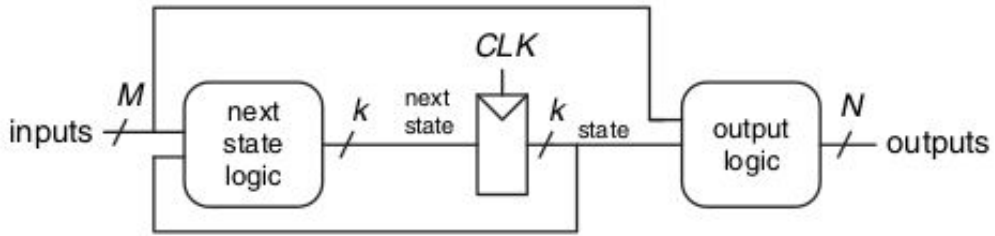
Sonlu Durum Makinaları

İstenilen fonksiyonu yerine getiren Senkron Ardısıl devrelerin tasarlanması için sematik bir yol sunar.



(a)

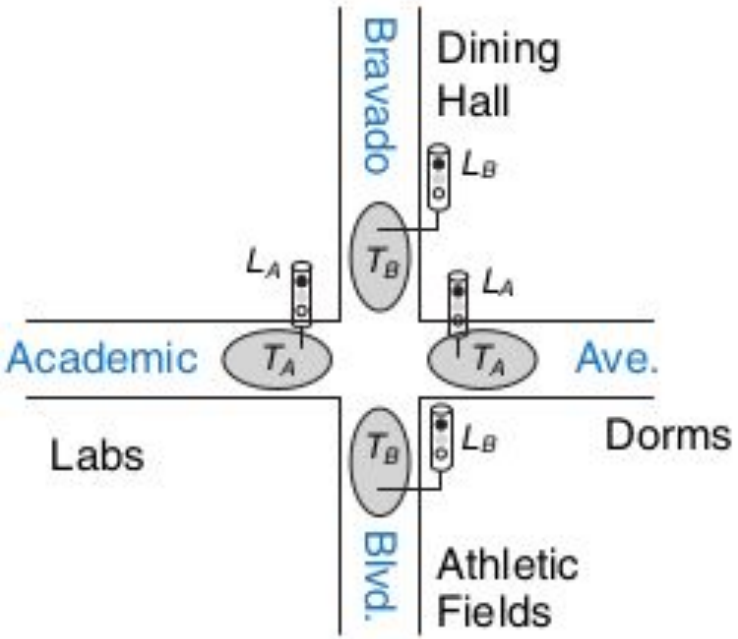
Moore machine

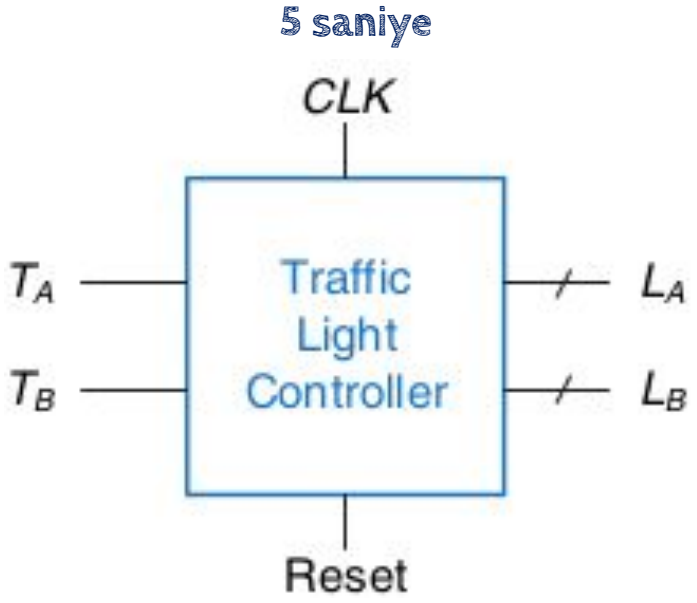


(b)

Mealy machine

Sonlu Durum Makinaları Örnegi





Sonlu Durum Makinaları Örnegi

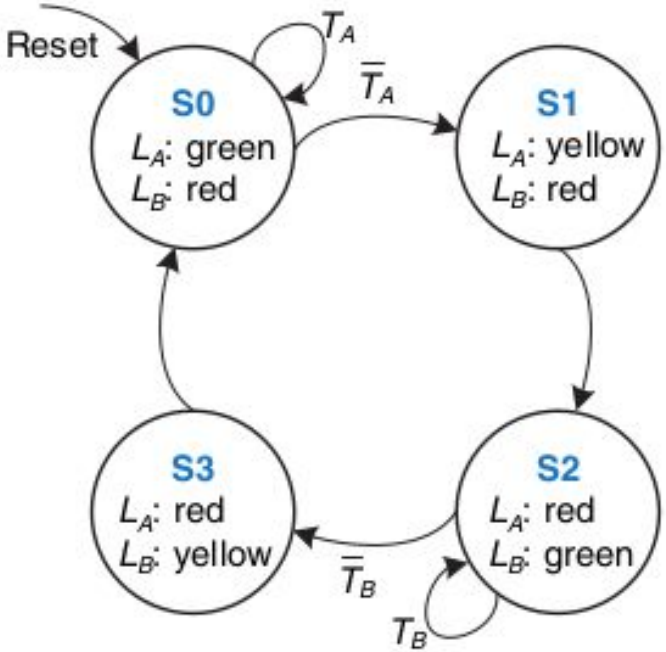


Table 3.1 State transition table

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

Sonlu Durum Makinaları Örnegi

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

Sonlu Durum Makinaları Örnegi

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

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

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

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

Sonlu Durum Makinaları Örnegi

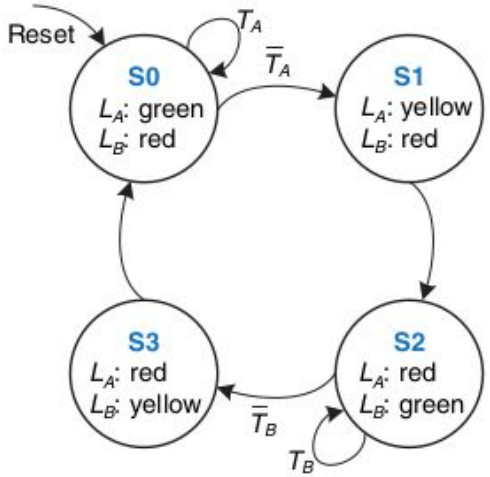


Table 3.5 Output table

Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	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

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

$$L_{A1} = S_1$$

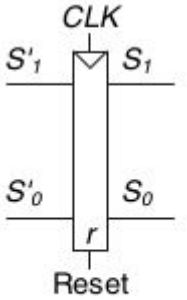
$$L_{A0} = \overline{S}_1 S_0$$

$$L_{B1} = \overline{S}_1$$

$$L_{B0} = S_1 S_0$$

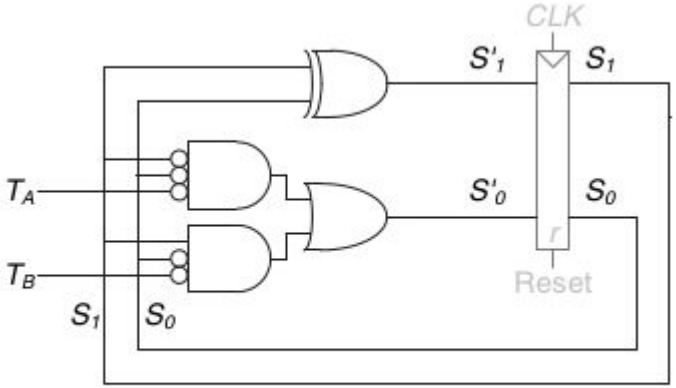
Sonlu Durum Makinaları Örnegi

Moore FSM



state register

(a)



inputs next state logic state register

(b)

$$S_1' = S_1 \oplus S_0$$

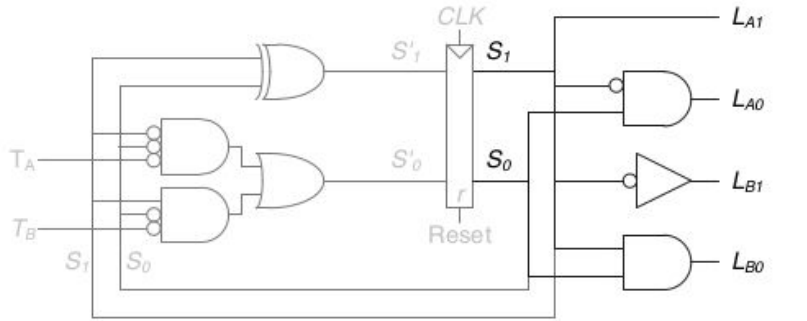
$$S_0' = \overline{S}_1 \overline{S}_0 \overline{T}_A + S_1 \overline{S}_0 \overline{T}_B$$

$$L_{A1} = S_1$$

$$L_{A0} = \overline{S}_1 S_0$$

$$L_{B1} = \overline{S}_1$$

$$L_{B0} = S_1 S_0$$



inputs next state logic state register output logic outputs

(c)

Sonlu Durum Makinaları Örnegi

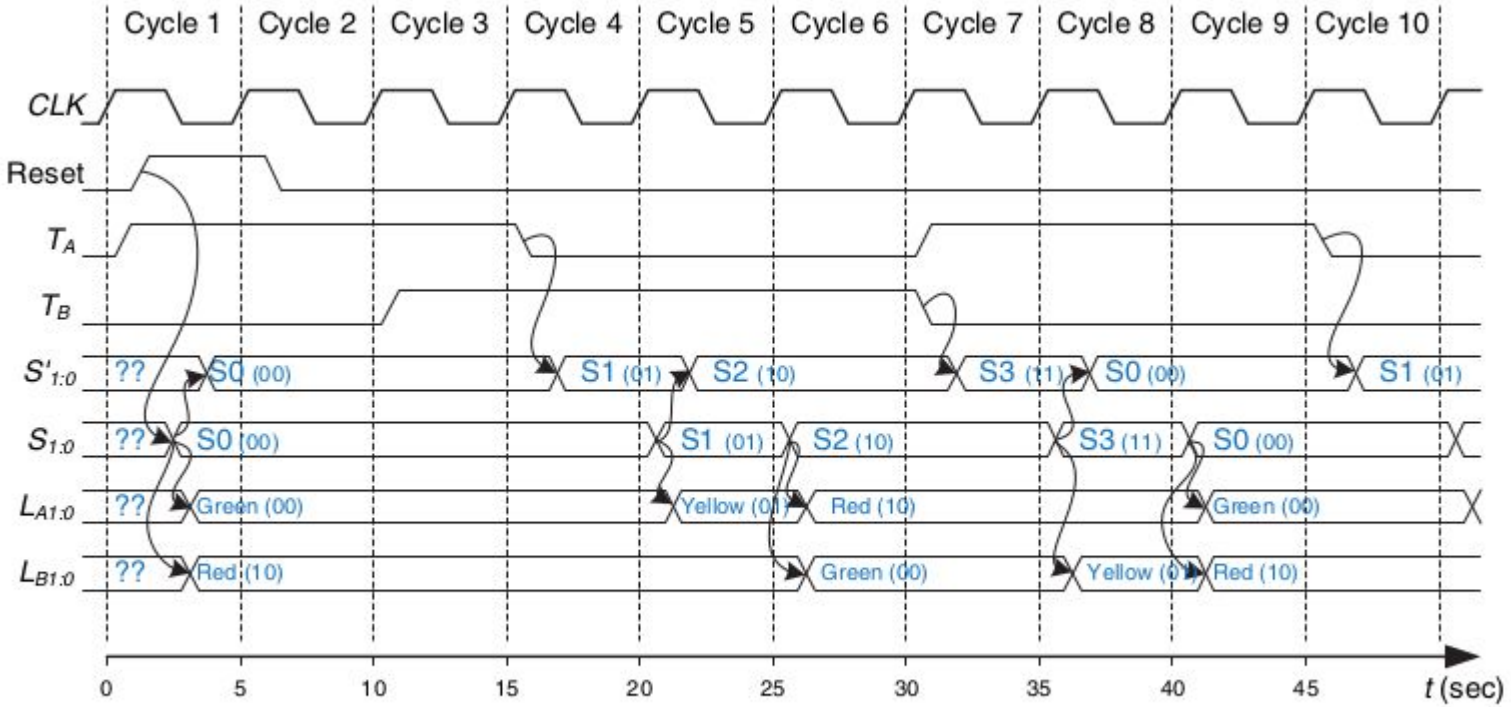


Figure 3.27 Timing diagram for traffic light controller

Durum Kodlama

ikili durum kodlama: Bir önceki örnekte oldugu gibi her durum ikili kodlama ile kodlanmaktadır. Daha fazla lojik kapı kullanılır.

tek-bit kodlama: Her durumda tek bir bit 1 degerini alan kodlamadır. Üç durum için 001, 010, 100 Daha fazla flip-flop kullanılır.

Durum Kodlama Örnek (Divide-by-N counter)

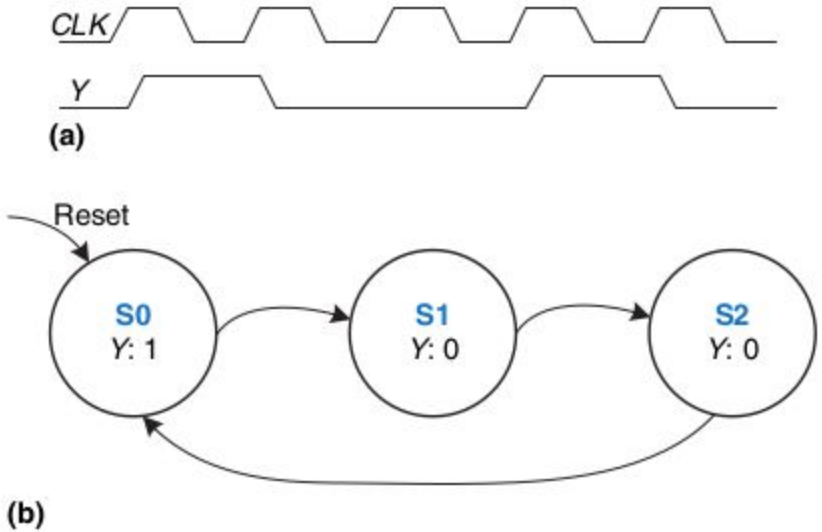


Figure 3.28 Divide-by-3 counter (a) waveform and (b) state transition diagram

Durum Kodlama Örnek (Divide-by-N counter)

Table 3.6 Divide-by-3 counter state transition table

Current State	Next State
S0	S1
S1	S2
S2	S0

Table 3.7 Divide-by-3 counter output table

Current State	Output
S0	1
S1	0
S2	0

Table 3.8 One-hot and binary encodings for divide-by-3 counter

State	One-Hot Encoding			Binary Encoding	
	S_2	S_1	S_0	S_1	S_0
S0	0	0	1	0	0
S1	0	1	0	0	1
S2	1	0	0	1	0

Durum Kodlama Örnek (Divide-by-N counter)

Table 3.6 Divide-by-3 counter state transition table

Current State	Next State
S0	S1
S1	S2
S2	S0

Table 3.7 Divide-by-3 counter output table

Current State	Output
S0	1
S1	0
S2	0

Table 3.9 State transition table with binary encoding

Current State		Next State	
S_1	S_0	S_1'	S_0'
0	0	0	1
0	1	1	0
1	0	0	0

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

$$Y = \bar{S}_1 \bar{S}_0$$

Durum Kodlama Örnek (Divide-by-N counter)

Table 3.6 Divide-by-3 counter state transition table

Current State	Next State
S0	S1
S1	S2
S2	S0

Table 3.7 Divide-by-3 counter output table

Current State	Output
S0	1
S1	0
S2	0

Table 3.10 State transition table with one-hot encoding

Current State			Next State		
S_2	S_1	S_0	S_2'	S_1'	S_0'
0	0	1	0	1	0
0	1	0	1	0	0
1	0	0	0	0	1

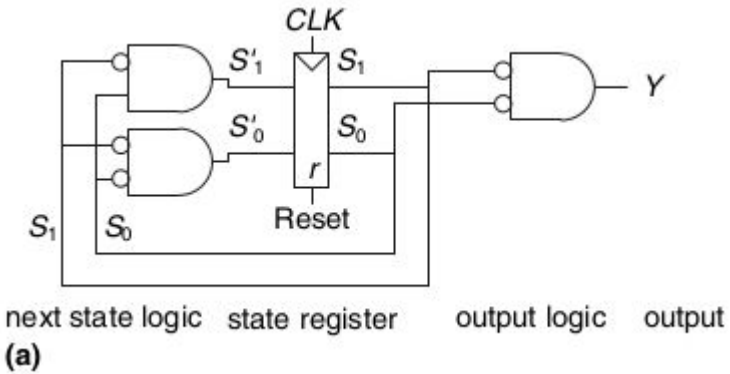
$$S_2' = S_1$$
$$S_1' = S_0$$
$$S_0' = S_2$$

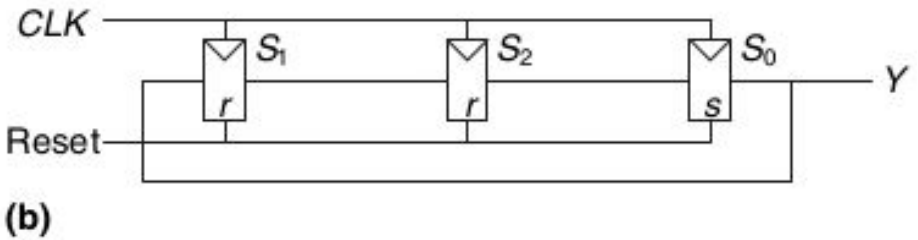
$$Y = S_0$$

Durum Kodlama Örnek (Divide-by-N counter)

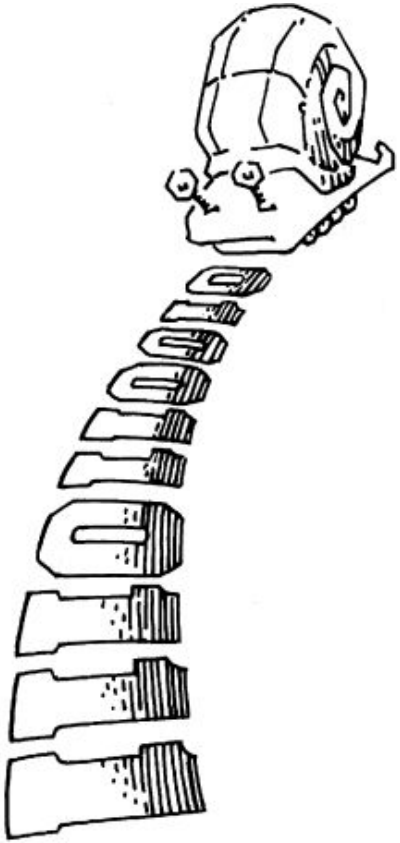
$$S_1' = \bar{S}_1 S_0$$
$$S_0' = \bar{S}_1 \bar{S}_0$$
$$Y = \bar{S}_1 \bar{S}_0$$

$$S_2' = S_1$$
$$S_1' = S_0$$
$$S_0' = S_2$$
$$Y = S_0$$



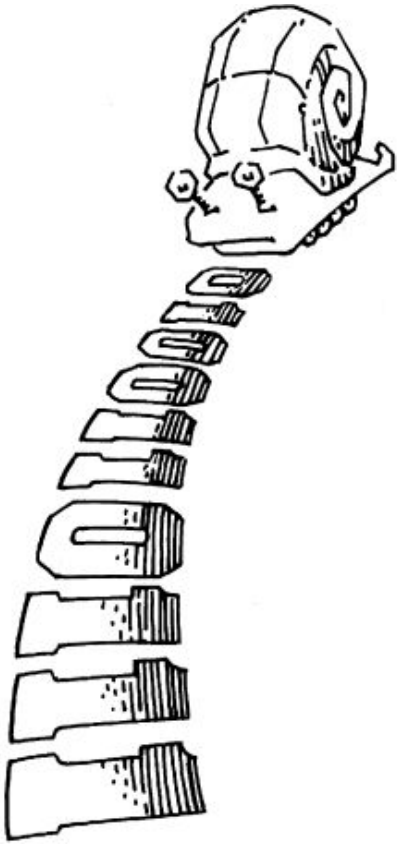


Moore and Mealy Makineleri

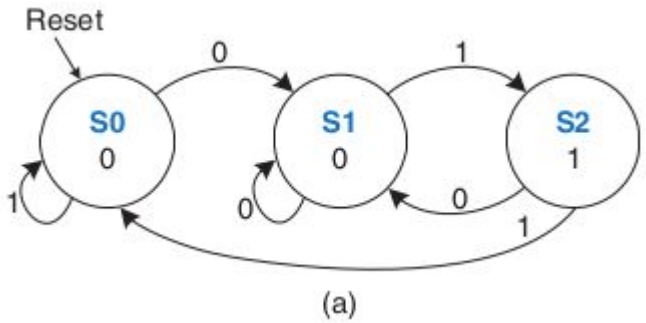


- Ali, FSM mantığı ile hareket eden robot bir salyangoz tasarladı.
- Salyangoz, 1 ve 0 dizisini içeren bir kağıt boyunca sağdan sola doğru ilerlemektedir.
- Salyangoz her saat dögüsünde bir bit geçmektedir.
- Geçtiği son iki bit 01 olursa salyangoz gülümsemektedir.
- Salyangozun ne zaman gülümseyeceğini hesaplayan FSM yi gelistiriniz.
- A girisi, salyangoz anteninin altındaki bittir.
- Salyangoz gülümsediğinde, Y çıkısı TRUE olur.
- Moore ve Mealy durum makinesi tasarımlarını karsılastırın.
- Alyssa'nın salyangozu 0100110111 dizisi boyunca gezinirken girisi, durumları ve çıktıyı gösteren her makine için bir zamanlama diyagramı çizin.

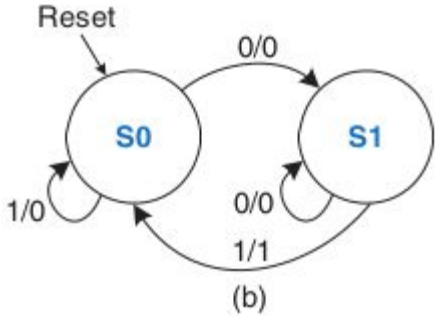
Moore and Mealy Makineleri



Moore Makinesi



Mealy Makinesi



A/Y

A: Gecisi tetikleyen girisler

Y: Gecise karsılık gelen cıkıs

Moore Makinesi

ikilik durum kodlama:
S0=00, S1=01, S2=10

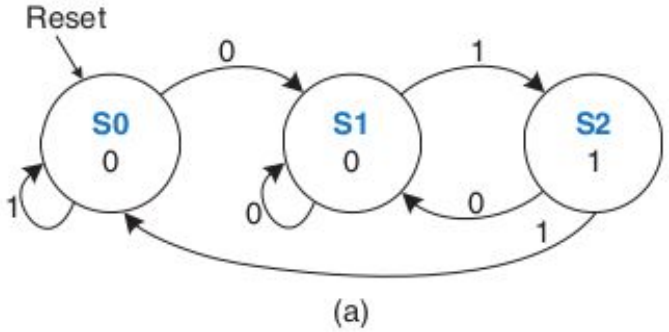


Table 3.11 Moore state transition table

Current State S	Input A	Next State S'
S0	0	S1
S0	1	S0
S1	0	S1
S1	1	S2
S2	0	S1
S2	1	S0

Table 3.12 Moore output table

Current State S	Output Y
S0	0
S1	0
S2	1

$$S_1' = S_0 A$$
$$S_0' = \overline{A}$$

$$Y = S_1$$

Mealy Makinesi

ikilik durum kodlama:
S0=0, S1=1

Table 3.15 Mealy state transition and output table

Current State S	Input A	Next State S'	Output Y
S0	0	S1	0
S0	1	S0	0
S1	0	S1	0
S1	1	S0	1

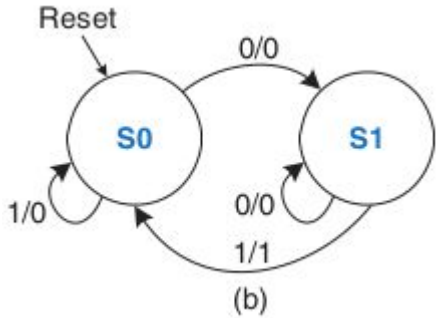


Table 3.16 Mealy state transition and output table with state encodings

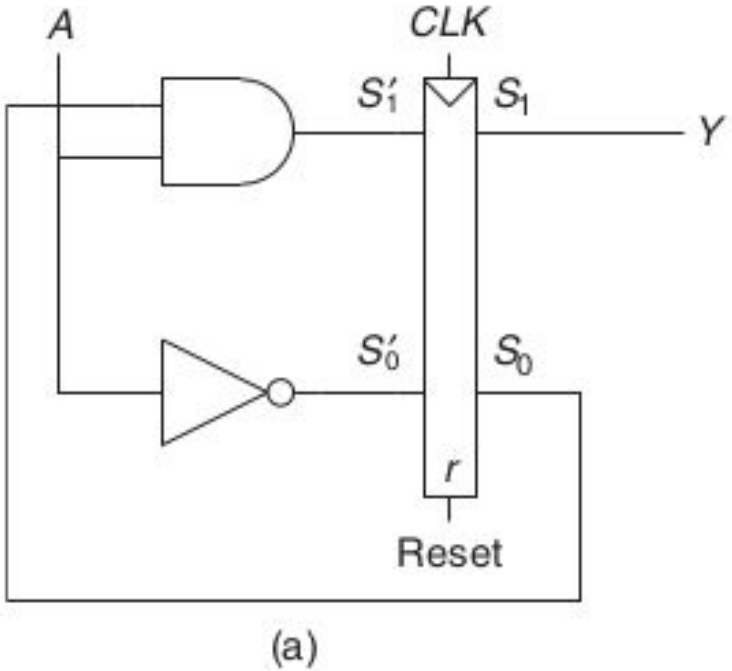
Current State S_0	Input A	Next State S'_0	Output Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

$$S'_0 = \overline{A}$$

$$Y = S_0 A$$

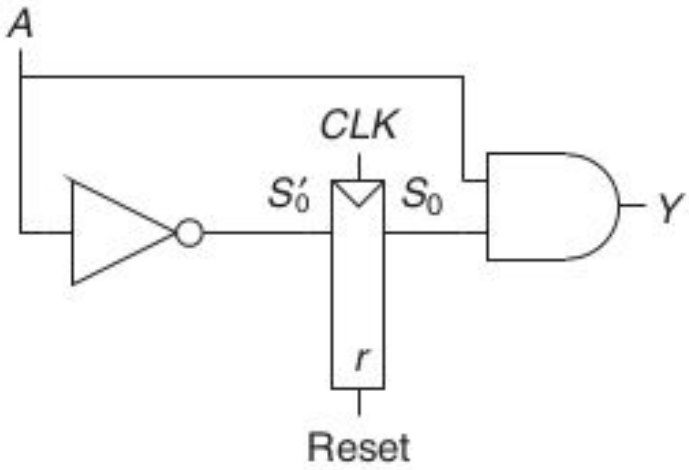
Moore and Mealy Makineleri

Moore Makinesi



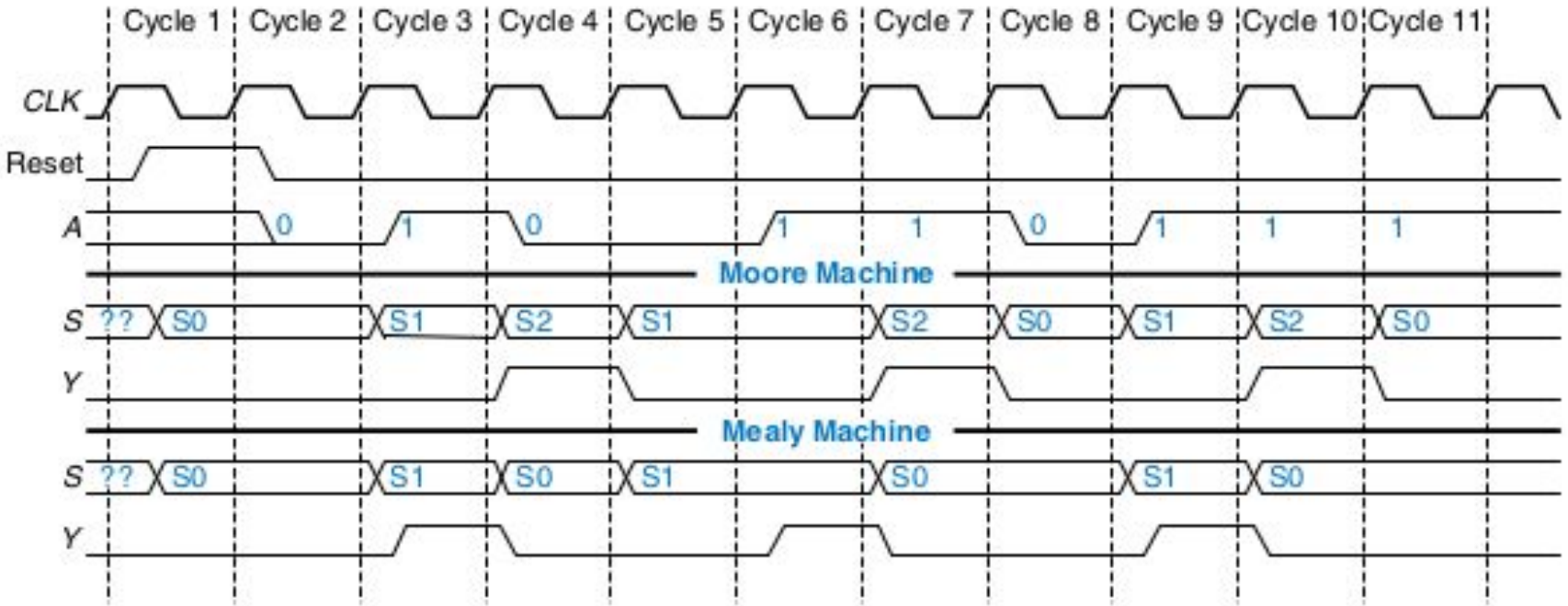
(a)

Mealy Makinesi

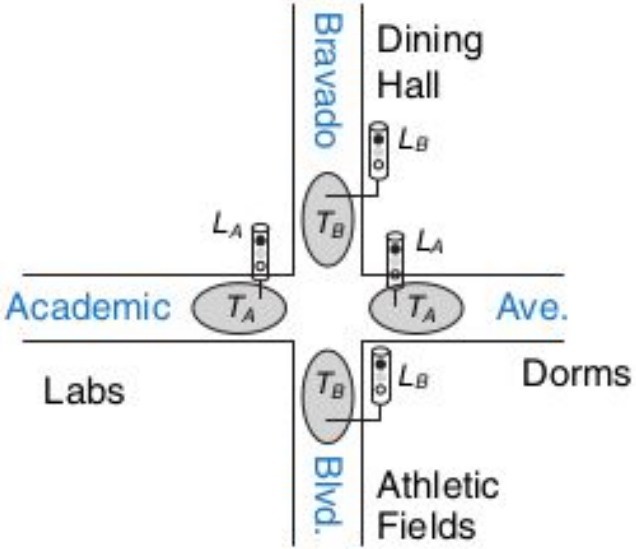


(b)

Moore and Mealy Makineleri

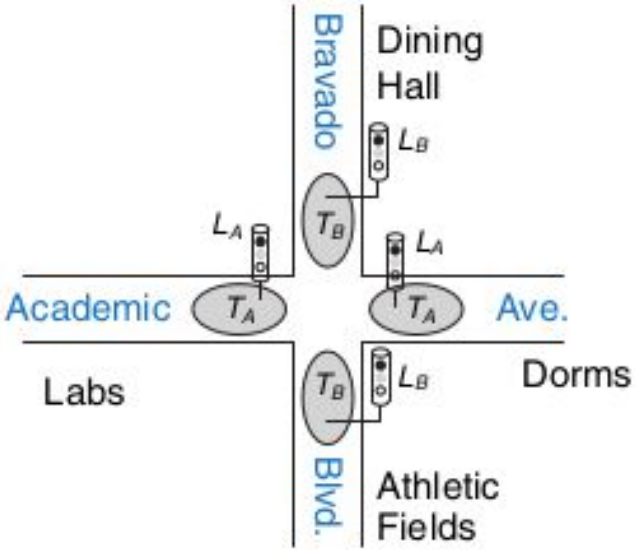


Durum Makinaları Üretimi

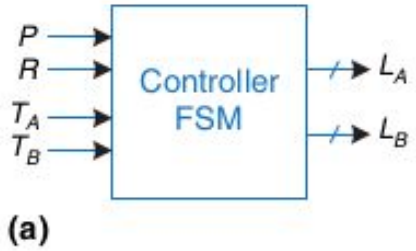


- Gecit Töreni Güncellemesi:
- Bravadı Bulvarında, futbol takımı ve bando takımı gecene kadar ısıgın yesilde kalması gerekmektedir.
- Dolayısıyla Kontrolör iki giris daha alır: P ve R
- Gecit töreninin baslaması için en az bir döngü: P
- Gecit töreninden cıkılması için en az bir döngü: R
- Geçit töreni modundayken, LB yesil olana kadar normal sıralamasını takip eder ve geçit modu sona erene kadar LB yesil durumduna kalır.

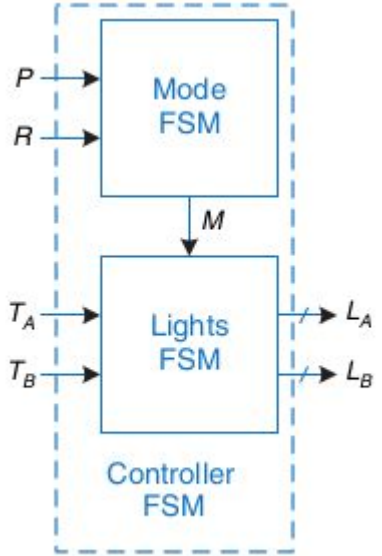
Durum Makinaları Üretimi



M: Gecit Töreni Modu

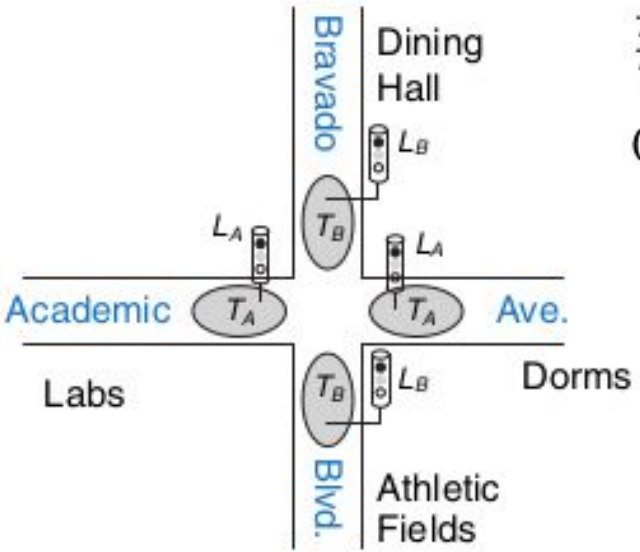


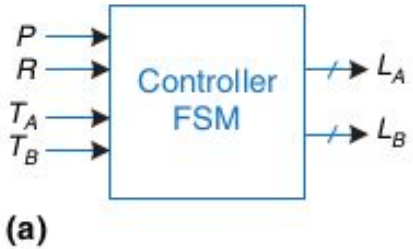
(a)



(b)

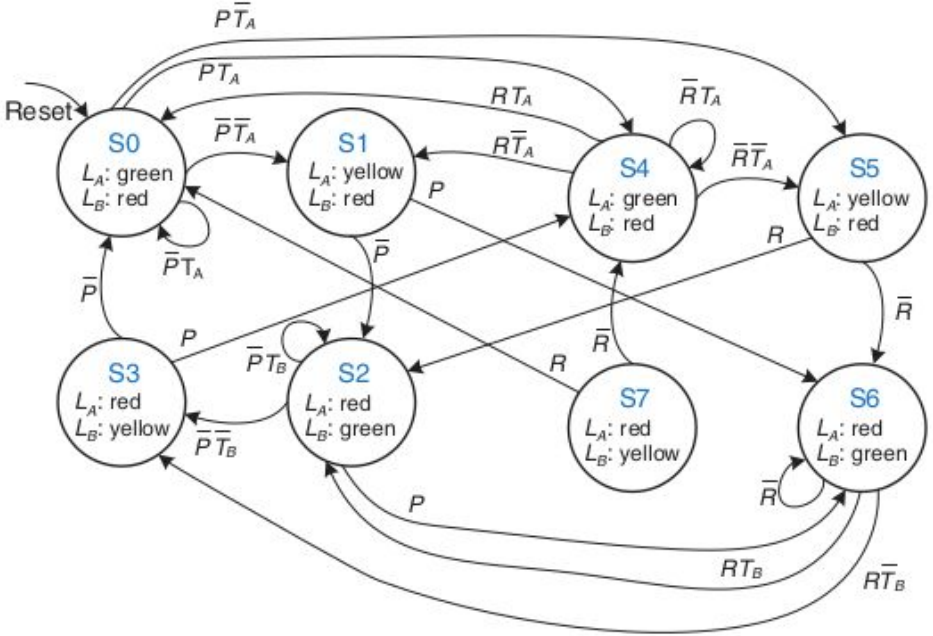
Durum Makinaları Üretimi





S0-S3: Normal Mod

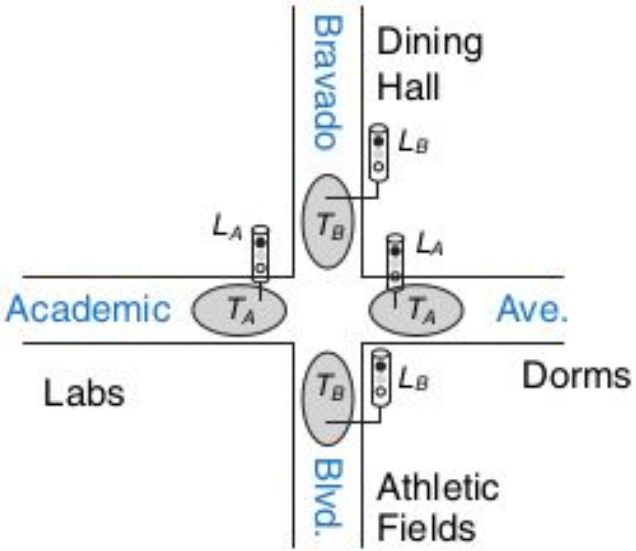
S4-S7: Gecit Modu

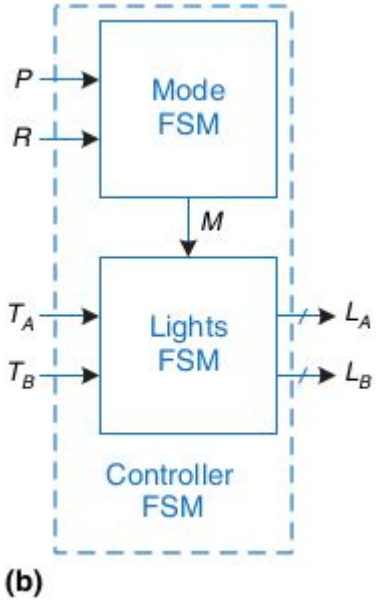


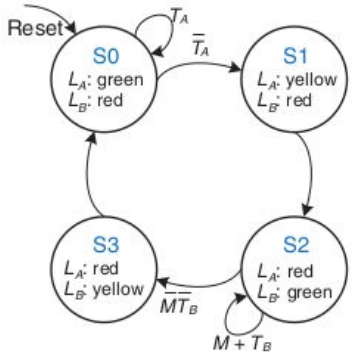
Durum Makinaları Üretimi

S0-S3: Normal Mod

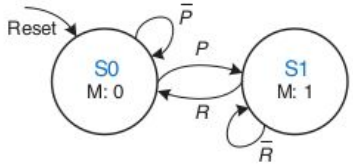
S4-S7: Gecit Modu





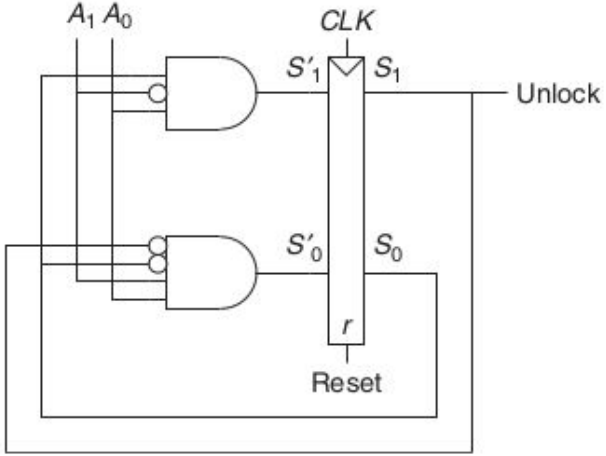


Lights FSM



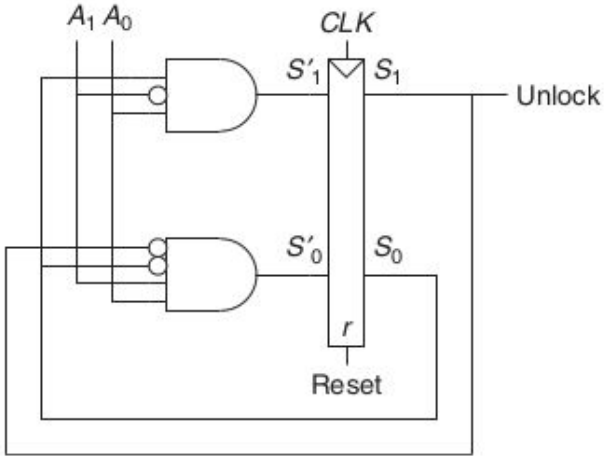
Mode FSM

FSM'yi Sematikten Çıkarma



- Ali eve geldi, ancak kapıdaki tus takımı kilidi yenilendigi icin eski kodu artık çalısmıyor.
- Ancak tus takımını yenileyen ustalar kapının üstünde yandaki devre seması bırakmıslardır.
- Ali, devrenin sonlu bir durum makinesi olabileceğini düsünüyor ve iceri girmek için durum geçis diyagramını çıkarmaya karar veriyor.

FSM'yi Sematikten Çıkarma



- Bu bir Moore makinesidir çünkü çıktı yalnızca durum bitlerine bağlıdır.
- Girisler : A1, A2
- Çıkıs: Unlock

$$S_1' = S_0\overline{A_1}A_0$$

$$S_0' = \overline{S_1}\,\overline{S_0}A_1A_0$$

$$Unlock = S_1$$

FSM'yi Sematikten Çıkarma

$$S_1' = S_0\overline{A_1}A_0$$

$$S_0' = \overline{S_1}\,\overline{S_0}A_1A_0$$

$$Unlock = S_1$$

Table 3.17 Next state table derived from circuit in Figure 3.35

Current State S_1	Current State S_0	Input A_1	Input A_0	Next State S_1'	Next State S_0'
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	1	0	0

Table 3.18 Output table derived from circuit in Figure 3.35

Current State S_1	Current State S_0	Output *Unlock*
0	0	0
0	1	0
1	0	1
1	1	1

FSM'yi Sematikten Çıkarma

$$S_1' = S_0 \overline{A_1} A_0$$

$$S_0' = \overline{S_1}\ \overline{S_0} A_1 A_0$$

$$Unlock = S_1$$

Table 3.19 Reduced next state table

Current State		Input		Next State	
S_1	S_0	A_1	A_0	S_1'	S_0'
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	X	X	0	0

Table 3.18 Output table derived from circuit in **Figure 3.35**

Current State		Output
S_1	S_0	*Unlock*
0	0	0
0	1	0
1	0	1
1	1	1

FSM'yi Sematikten Çıkarma

$$S_1' = S_0 \overline{A_1} A_0$$

$$S_0' = \overline{S_1}\ \overline{S_0} A_1 A_0$$

$$Unlock = S_1$$

Table 3.19 Reduced next state table

Current State		Input		Next State	
S_1	S_0	A_1	A_0	S_1'	S_0'
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	X	X	0	0

Table 3.18 Output table derived from circuit in **Figure 3.35**

Current State		Output
S_1	S_0	*Unlock*
0	0	0
0	1	0
1	0	1
1	1	1

FSM'yi Sematikten Çıkarma

$$S_1' = S_0 \overline{A_1} A_0$$

$$S_0' = \overline{S_1}\ \overline{S_0} A_1 A_0$$

$$Unlock = S_1$$

Table 3.21 Symbolic next state table

Current State S	Input A	Next State S'
S0	0	S0
S0	1	S0
S0	2	S0
S0	3	S1
S1	0	S0
S1	1	S2
S1	2	S0
S1	3	S0
S2	X	S0

Table 3.22 Symbolic output table

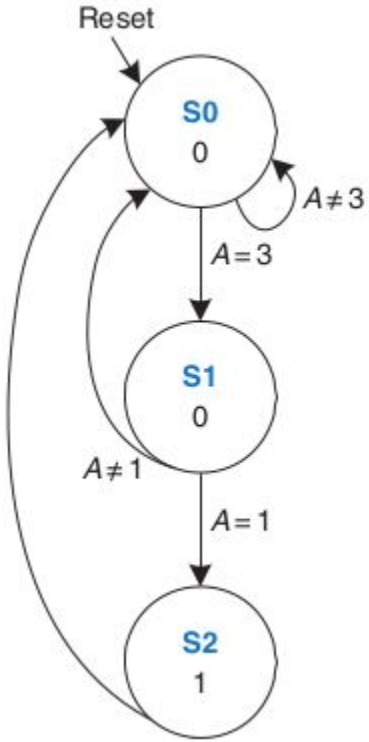
Current State S	Output *Unlock*
S0	0
S1	0
S2	1

FSM'yi Sematikten Çıkarma

$$S_1' = S_0\overline{A_1}A_0$$

$$S_0' = \overline{S_1}\,\overline{S_0}A_1A_0$$

$$Unlock = S_1$$



FSM Özeti

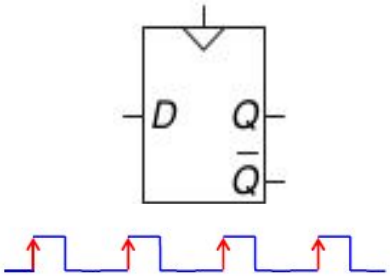
Sonlu durum makinaları, davranısı verilmis ardısıl devrelerin tasarımı için güclü bir yöntemdir.

Bir FSM tasarlamak için asagıdaki adımlar izlenmelidir:

- Devre giris ve çıkıslarını tanımlayın.
- Bir durum geçis diyagramı çizin.
- Moore makinesi için:
  - Bir durum geçis tablosu yazın.
  - Bir çıktı tablosu yazın.
- Mealy makinesi için:
  - Durum geçis ve çıktı tablosu beraber yazın.
- Kodlamayı secin-Kodlama tasarlanan donanımı etkiler-
- Sonraki durum ve çıkıs lojigi için boolean denklemlerini yazın
- Devre semasını çizin.

3.5 Ardısık Mantık Zamanlaması

Çıkan kenar süresince D giriside degisir ne olur?



Positive Edge Triggering

Statik Disipline göre; D sinyali çıkan kenar süresinin sonunda kararlı bir hal aldılgında kullanılabilir.

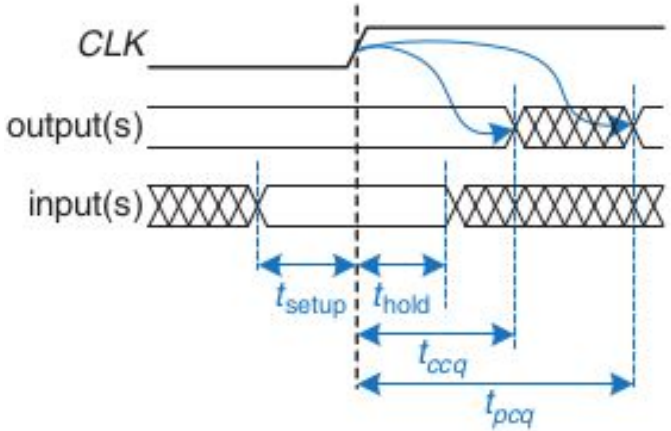
t dogal sayı ve n tam sayılı olmak kosulu ile;
t. saat sinyali sonunda A[t] yazmak yerine n.saat sinyali sonunda A[n] yazılabilir.

Dinamik Disiplin

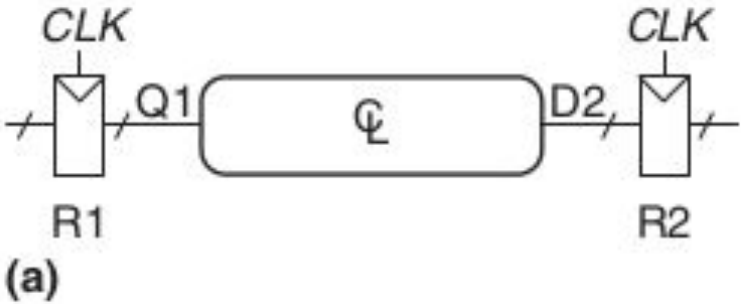
Cıkıs: t_{ccq} (contamination delay) gecikmesinden sonra, degismeye baslamalı ve t_{pcq} (propagation delay) içinde son degerine ulasmalıdır.

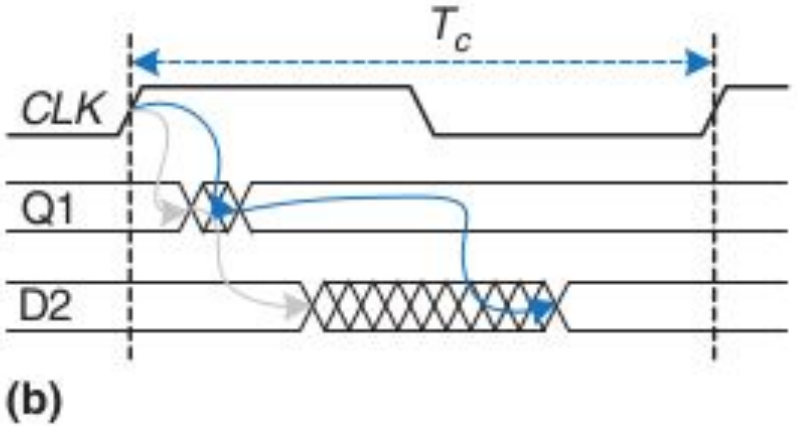
Giris yükselen kenarından önce kurulum süresi (t_{setup}) ve yükselen kenarından sonra tutma süresi(t_{hold}) toplamında sabit kalmalıdır.

Dinamik disiplin, bir senkron ardısık devrenin girislerinin açıklık süresince (aperture time = t_{setup} + t_{hold}) kararlı olmasını garanti eder.

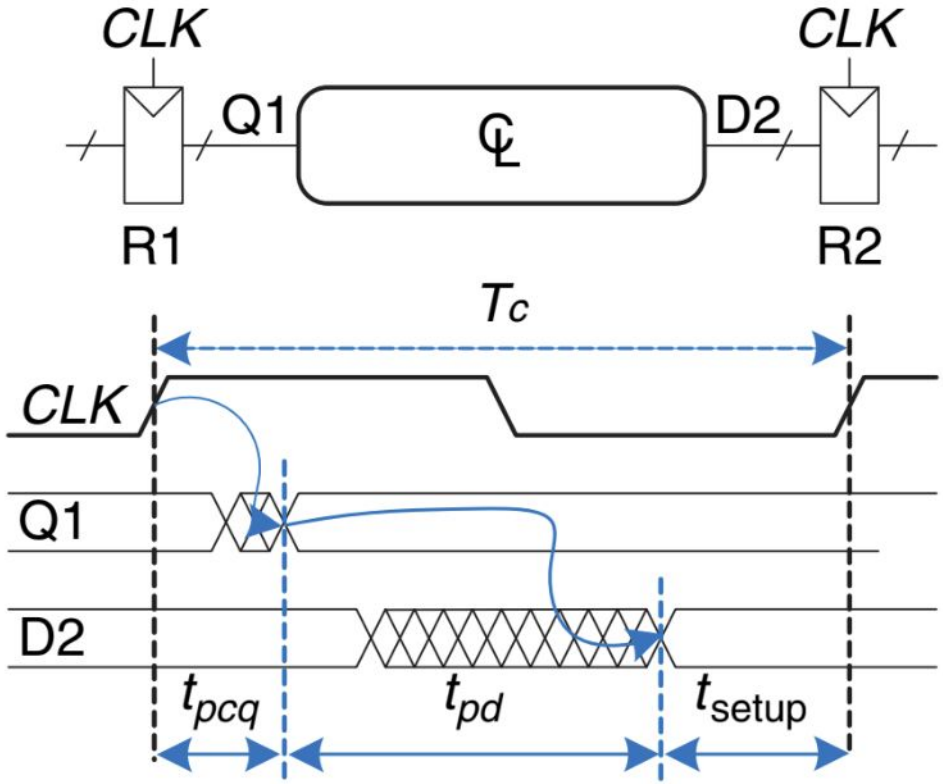


Sistem Zamanlaması

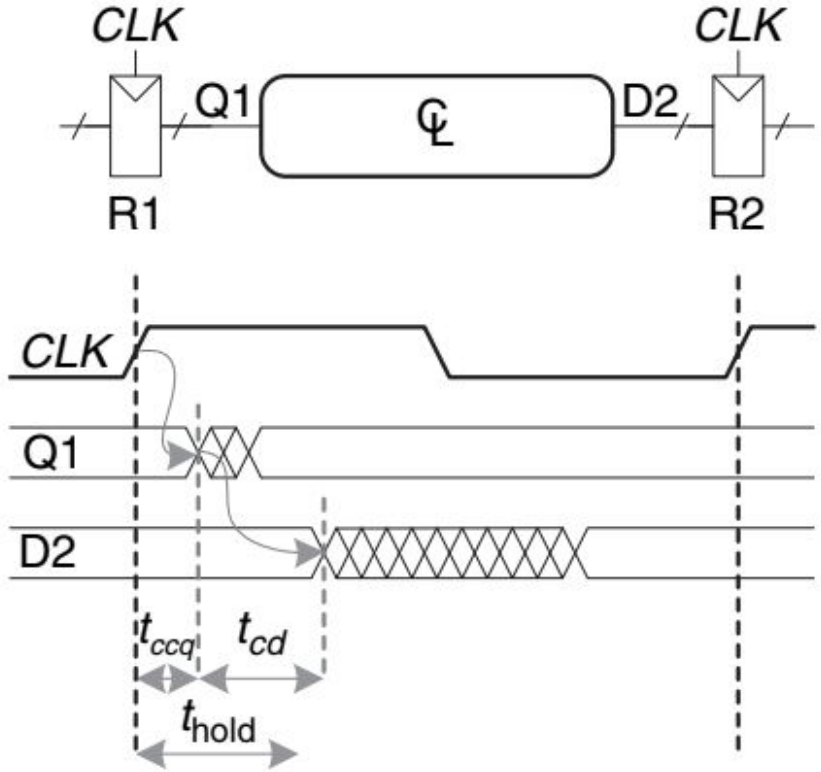




Kurulum Süresi Kısıtlaması



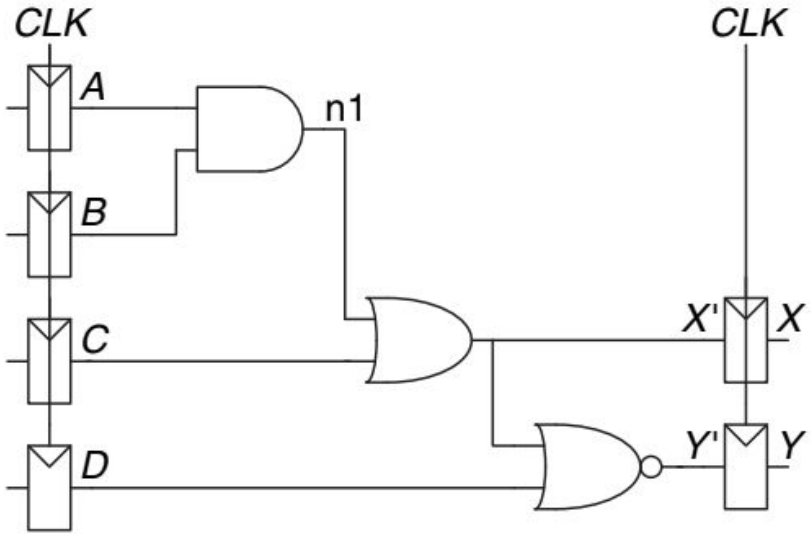
Tutuma Süresi Kısıtlaması



Hepsi Bir Arada

clock-to-Q kirlenme (contamination) gecikmesi: 30ps
yayılım gecikmesi(propagation delay): 80ps
Kurulum süresi: 50ps
Bekleme süresi: 60ps
Herbir lojik kapının yayılım gecikmesi, 40ps, kirleme gecikmesi: 25ps

Maksimum clock frekansı ?
Bekleme süresi ihlali ?

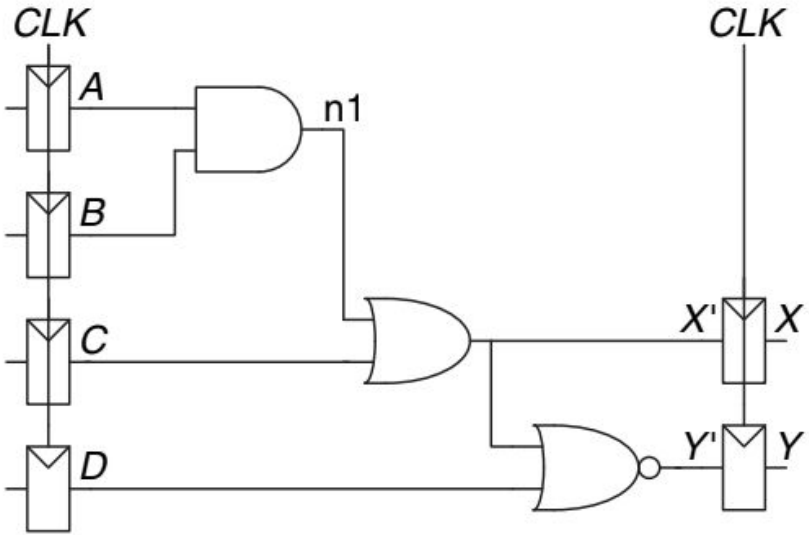


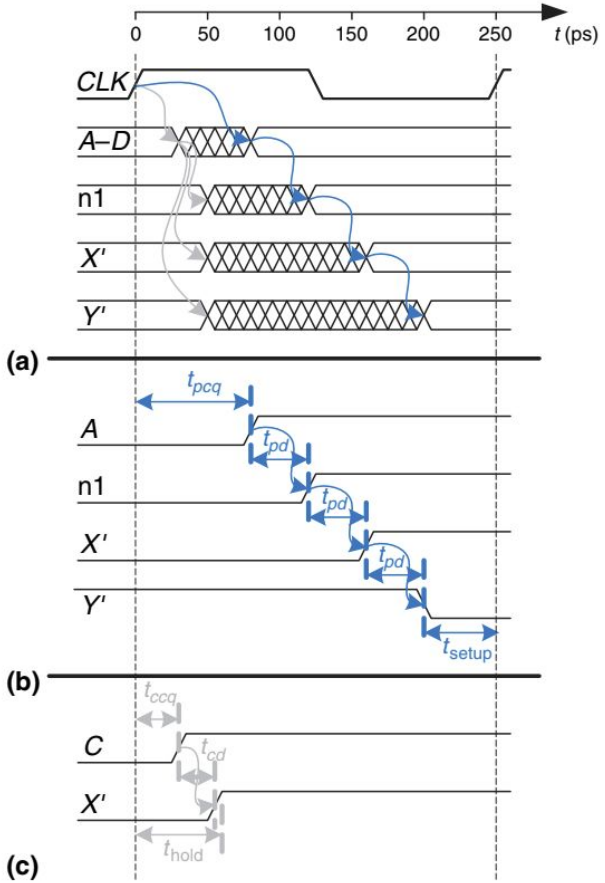
Hepsi Bir Arada

Zaman analizi:

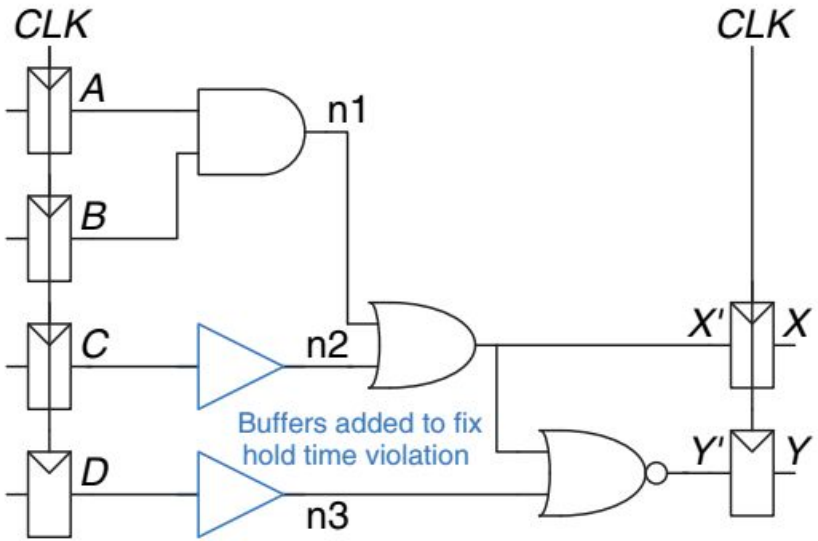
$$T_c \geq t_{pcq} + 3\,t_{pd} + t_{\text{setup}} = 80 + 3 \times 40 + 50 = 250\text{ps}$$

The maximum clock frequency is $f_c = 1/T_c = 4$ GHz.

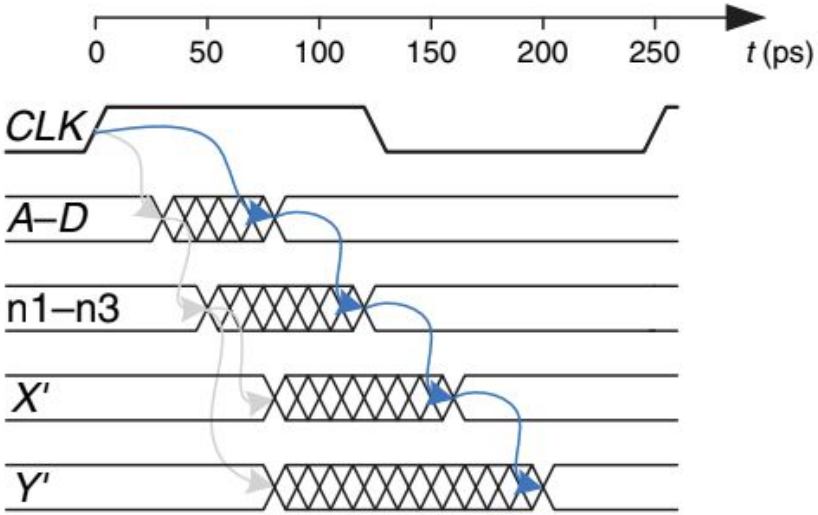




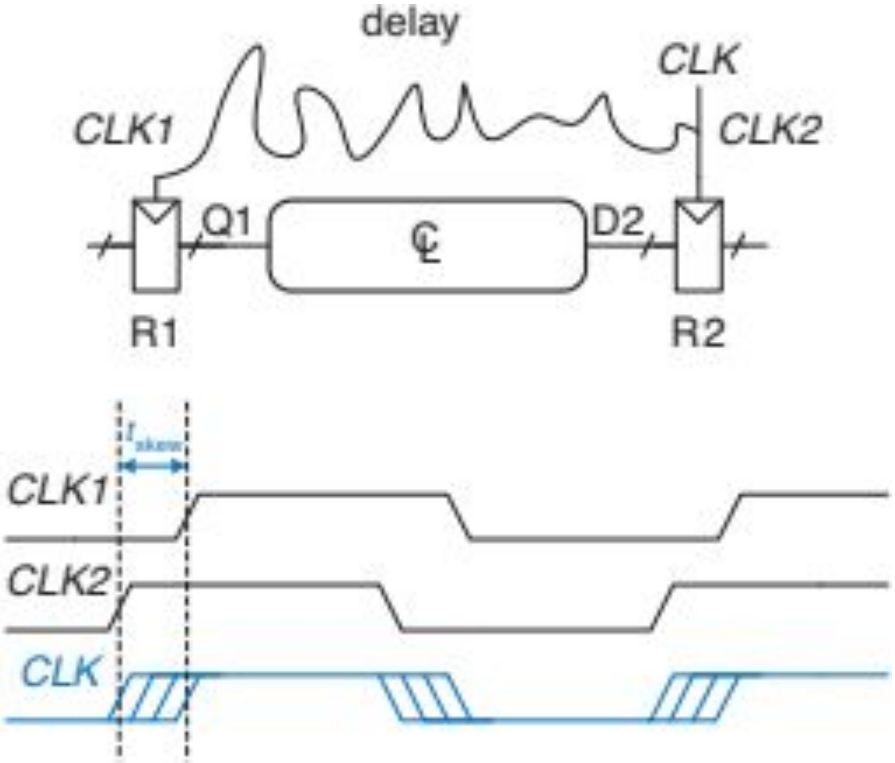
SABİT TUTMA SÜRESİ İHLALLERİ



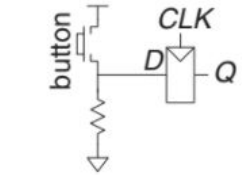
SABİT TUTMA SÜRESİ İHLALLERİ

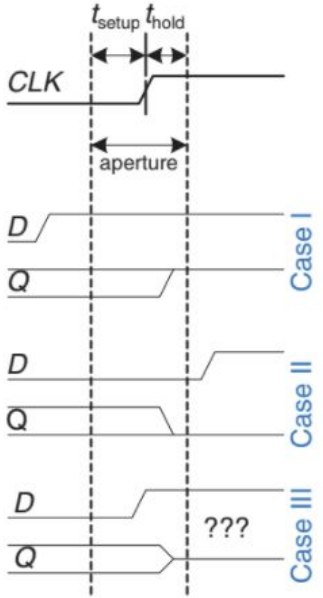


Saat Çarpıklığı*

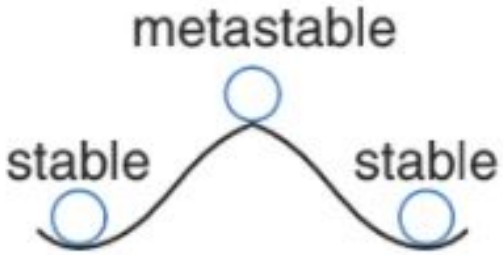


3.5.4 Meta kararlılık

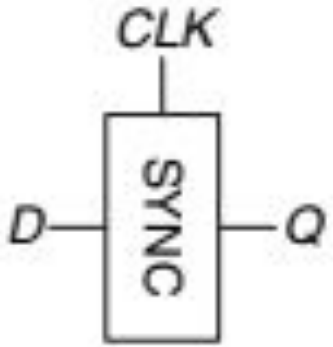




Meta kararlılık Durumu



Es Zamanlama



Paralellik

Paralellik

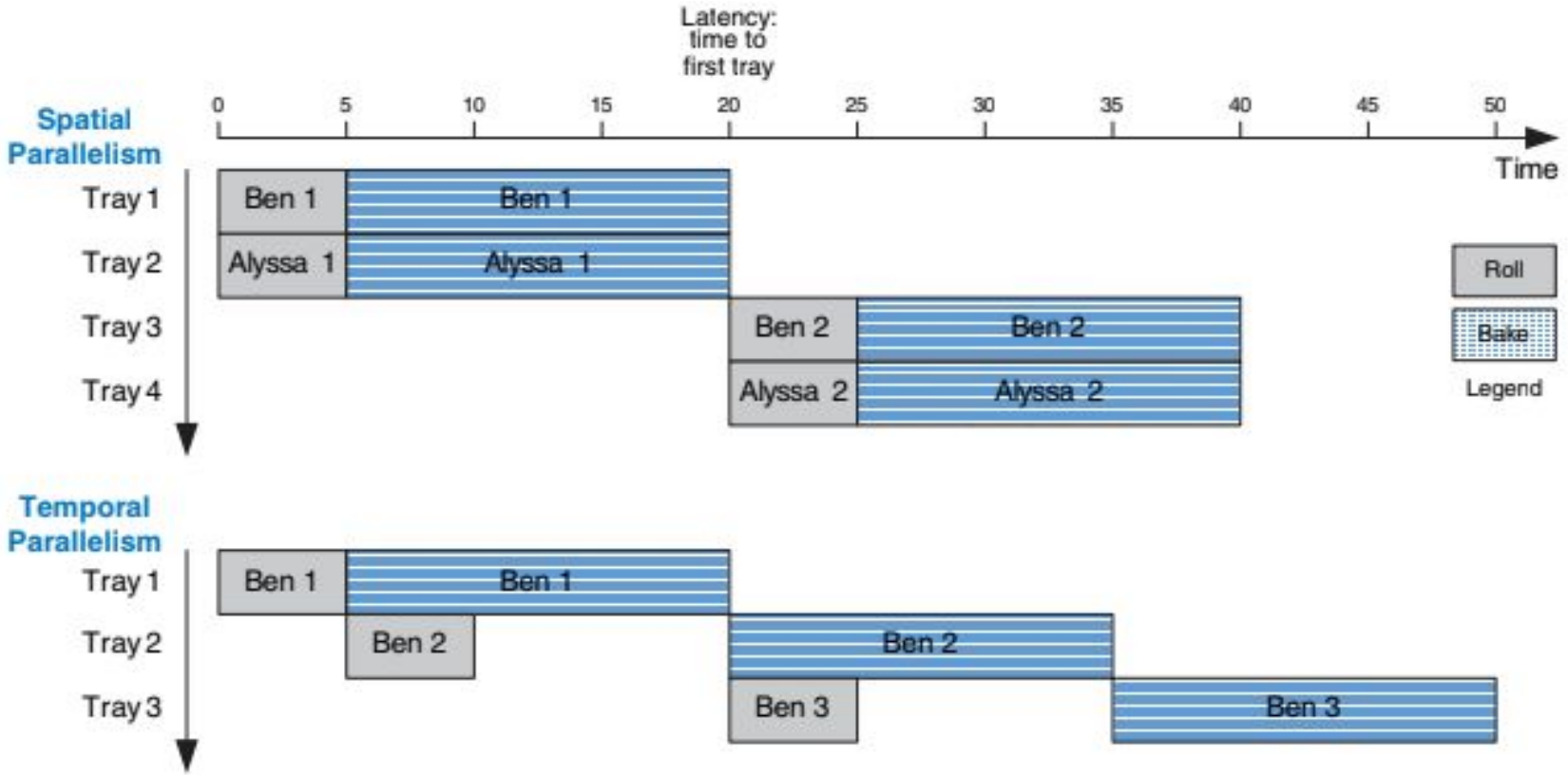
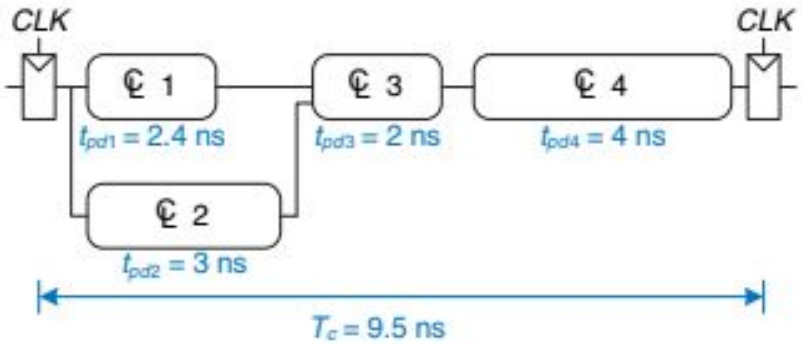
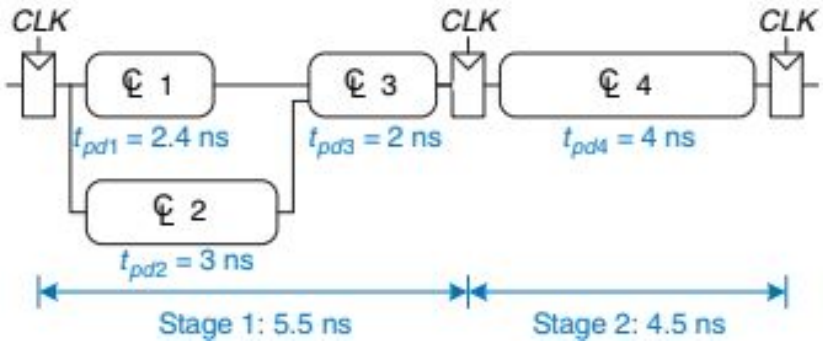


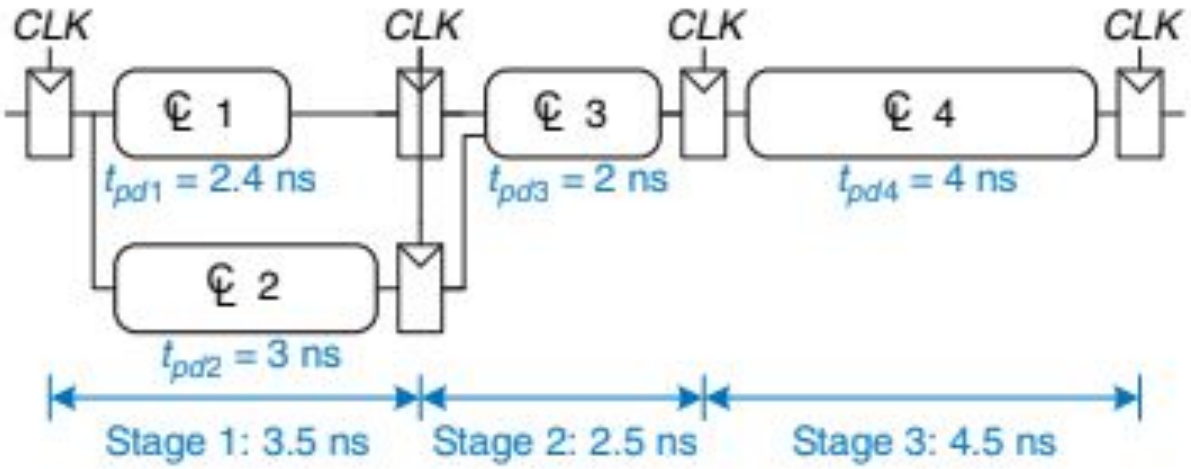
Figure 3.57 Spatial and temporal parallelism in the cookie kitchen

Paralellik


CLK
CLK
1
2
3
4
t_{pd1} = 2.4 ns
t_{pd2} = 3 ns
t_{pd3} = 2 ns
t_{pd4} = 4 ns
T_c = 9.5 ns


CLK
CLK
CLK
1
2
3
4
t_{pd1} = 2.4 ns
t_{pd2} = 3 ns
t_{pd3} = 2 ns
t_{pd4} = 4 ns
Stage 1: 5.5 ns
Stage 2: 4.5 ns

Paralellik


CLK
CLK
CLK
CLK
C 1
C 2
C 3
C 4
t_{pd1} = 2.4 ns
t_{pd2} = 3 ns
t_{pd3} = 2 ns
t_{pd4} = 4 ns
Stage 1: 3.5 ns
Stage 2: 2.5 ns
Stage 3: 4.5 ns

Sorular