VHDL

ENTITY = Modülün baglantilari ve baglanti yollarinin türü tanimlanir.

```
entity modülün ismi is
        [generik_bildirimi]
        [giris/çikis tanimlari]
        {entity-bildirim-elemanlari}
[begin
        devrenin çalisma sartlarinin kontrol edildigi bölüm]
end [modülün ismi];
```

Generic = Modülün yapisini veya davranisini kontrol etmek üzere kullanilan sabitlerin tanimlandigi alandir.

```
generic(
        sabit_ismi : tipi [:=ilk degeri]
        {;sabit_ismi : tipi [:= ilk degeri]};
```

→

```
port(
        port_ismi : [mod] tipi [:=ilk degeri]
        {;port_ismi : tipi [:= ilk degeri]});
```

→ 1 Bitlik Tam Toplayici

```
entity TAM_TOPLAYICI is
        port (A,B,EG : in STD_LOGIC;
              TOPLAM,EC : out STD_LOGIC);
end TAM_TOPLAYICI;
```

→ TOPLAYICI (4 Bitlik)

```
entity TOPLAYICI is
        generic (N: INTEGER := 4;      // Toplanacak sayilarin basamak sayisi
                 M: TIME := 10 ns);    // Entity zamanlama davranisi
        port( A,B : in STD_LOGIC_VECTOR (N-1 downto 0);
              T : out STD_LOGIC_VECTOR(N-1 downto 0);
              EG: in STD_LOGIC;
              EC: out STD_LOGIC);
end TOPLAYICI;
```

MİMARİ : Modülün girişleri ve çikişlari arasindaki ilişkiler tanimlanir. (Davranişsal - Veri Akişi - Yapisal)

```
architecture mimari_ismi of modülün_ismi is
        {mimari_bildirim_bölümü}
begin
        {eşzamanli_satirlar}
end [mimari_ismi];
```

Davranışsal Mimari ile Tam Toplayıcı

```
architecture BEHAVIOUR of TAM_TOPLAYICI is
begin
    process (A,B,EG)
    begin
        if (A='0' and B='0' and EG='0') then
            TOPLAM <='0';
            EG <='0';
        elsif (A='0' and B='0' and EG='1') or
              (A='0' and B='1' and EG='0') or
              (A='1' and B='0' and EG='0') then
              TOPLAM <='1';
              EG <='0';
        elsif (A='0' and B='1' and EG='1') or
              (A='1' and B='0' and EG='1') or
              (A='1' and B='1' and EG='0') then
              TOPLAM <='0';
              EG <='1';
        else
              TOPLAM <='1';
              EG <='1';
        endif;
    end process;
end BEHAVIOUR;
```

Veri Akışı Mimari ile Tam Toplayıcı

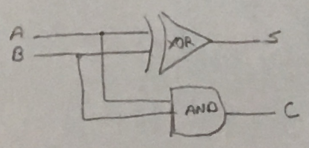
```
architecture DATAFLOW of TAM_TOPLAYICI is
        signal S: STD_LOGIC;
begin
    S <= A xor B
    TOPLAM <= S xor EG after 10 ns;
    EG <= (A and B) or (S and EG) after 5 ns;
end DATAFLOW;
```

```
entity TAM_TOPLAYICI is
    generic (N: TIME := 5 ns);
    port (A,B,EG: in STD_LOGIC;
          TOPLAM,EG: out STD_LOGIC);
end TAM_TOPLAYICI;

architecture DATAFLOW of TAM_TOPLAYICI is
        signal S: STD_LOGIC;
begin
    S <= A xor B
    TOPLAM <= S or EG after 2*N;
    EG <= (A and B) or (S and EG) after N;
end DATAFLOW;
```

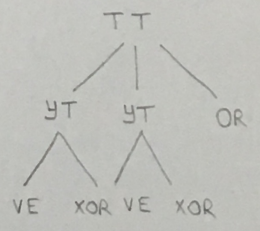
Yapısal Mimari ile TAM TOPLAYICI

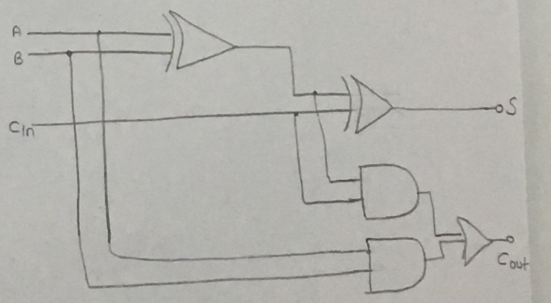
①
```
entity EXOR_KAPISI is
        port(G1,G2 : in STD_LOGIC;
             Cikis : out STD_LOGIC);
end EXOR_KAPISI;

architecture BEHAVIOUR of EXOR_KAPISI is
begin
      process(G1,G2)
      begin
           if (G1='0' and G2='0')or
              (G1='1' and G2='1') then
               Cikis <='0';
           else
               Cikis <='1';
           endif;
      end process;
end BEHAVIOUR;
```

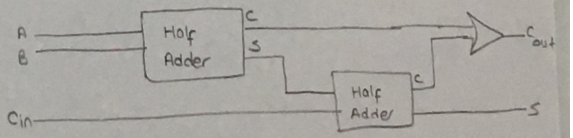
②
```
entity YARI_TOPLAYICI is
        port(G1,G2 : in STD_LOGIC;
             ELDE,TOPLAM :out STD_LOGIC);
end YARI_TOPLAYICI;

architecture STRUCTURE of YARI_TOPLAYICI is
          component EXOR_KAPISI
              port(G1,G2 : in STD_LOGIC;
                   Cikis :out STD_LOGIC);
          end component;
          component VE_KAPISI
              port(G1,G2 : in STD_LOGIC;
                   Cikis :out STD_LOGIC);
          end component;
        begin
            B1: EXOR_KAPISI port map (A,B,TOPLAM);
            B2: VE_KAPISI port map (A,B,ELDE);
end STRUCTURE;
```



③
```
architecture STRUCTURE of TAM_TOPLAYICI is
          component YARI_TOPLAYICI
              port(G1,G2 : in STD_LOGIC;
                   ELDE,TOPLAM : out STD_LOGIC);
          end component;
          component VEYA_KAPISI
              port(G1,G2 : in STD_LOGIC;
                   Cikis : out STD_LOGIC);
          end component;
          signal N1,N2,N3 : STD_LOGIC;
begin
      YT1: YARI_TOPLAYICI port map (A,B,N1,N2);
      YT2: YARI_TOPLAYICI port map (N2,EG,N3,TOPLAM);
      VEYA1: VEYA_KAPISI port map (N1,N3,EG);
end STRUCTURE;
```







```
entity VE_KAPISI is
      port(ve_g1,ve_g2 :in STD_LOGIC;
           ve_cikis :out STD_LOGIC);
end VE_KAPISI;
architecture veri_akisi of VE_KAPISI is
begin
     process(ve_g1,ve_g2)
     begin
          ve_cikis <= ve_g1 and ve_g2;
     end process;
end veri_akisi;
---
entity VEYA_KAPISI is
      port(veya_g1,veya_g2:in STD_LOGIC;
           veya_cikis:out STD_LOGIC);
end VEYA_KAPISI;
architecture davranissal of VEYA_KAPISI is
begin
     process(veya_g1,veya_g2)
     begin
          if(veya_g1='0' and veya_g2:'0') then
             veya_cikis<='0';
          else
             veya_cikis<='1';
          endif;
     end process;
end davranissal;
---
```

```
entity FF is
      port (D,Clock:in STD_LOGIC;
            Q,QT:inout STD_LOGIC);
end FF;
architecture davranissal of FF is
begin
     process(Clock)
     begin
          if Clock'EVENT and Clock='1' then
             Q<=D;
             QT<=not D;
          endif;
     end process;
end davranissal;
---
entity devre is
      port(x,saat:in STD_LOGIC;
           Z:out STD_LOGIC);
end devre;
architecture yapisal of devre is
componet VE_KAPISI is
      port(ve_g1,ve_g2:in STD_LOGIC;
           ve_cikis:out STD_LOGIC);
end component;
component VEYA_KAPISI is
      port(veya_g1,veya_g2:in STD_LOGIC;
           veya_cikis:out STD_LOGIC);
end component;
component FF is
      port(D,Clock:in STD_LOGIC;
           Q,QT:inout STD_LOGIC);
end component;
signal s1,s2,s3,s4,s5,s6 : STD_LOGIC;
```

```
begin
     blok1:
     blok2
     blok3:
     blok4:
```

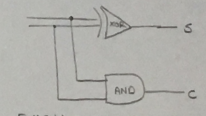
```
entity VE_KAPISI is
      port ( ve_g1,ve_g2 :in STD_LOGIC;
             ve_cikis:out STD_LOGIC);
end VE_KAPISI;
architecture veri_akisi of VE_KAPISI is
begin
     process (ve_g1,ve_g2)
     begin
          ve_cikis <=ve_g1 and ve_g2 ;
     end process;
end veri_akisi;
---
entity XOR_KAPISI is
      port ( xor_g1, xor_g2 : in STD_LOGIC;
             xor_cikis:out STD_LOGIC);
end XOR_KAPISI;
architecture veri_akisi of XOR_KAPISI is
begin
     process ( xor_g1 ,xor_g2)
     begin
          xor_cikis <= xor_g1 xor xor_g2 ;
     end process;
end veri_akisi;
---
entity yt is
      port ( yt_g1, yt_g2 :in STD_LOGIC;
             yt_toplam,yt_elde:out STD_LOGIC);
end yt;
architecture yapisal of yt is
      component XOR_KAPISI is
           port ( xor_g1,xor_g2 :in STD_LOGIC;
                  xor_cikis:out STD_LOGIC);
      end component ;
      component VE_KAPISI is
           port ( ve_g1,ve_g2 :in STD_LOGIC;
                  ve_cikis:out STD_LOGIC);
      end component;
begin
     blok1 : XOR_KAPISI port map( xor_g1 =>yt_g1, xor_g2 =>yt_g2 , xor_cikis =>yt_toplam);
     blok2 : VE_KAPISI port map (ve_g1 =>yt_g1, ve_g2 =>yt_g2 , ve_cikis => yt_elde);
end yapisal;
---
entity VEYA_KAPISI is
      port( veya_g1 ,veya_g2 : in STD_LOGIC;
            veya_cikis : out STD_LOGIC);
end VEYA_KAPISI;
architecture veri_akisi of VEYA_KAPISI is
begin
     process (veya_g1,veya_g2)
     begin
          veya_cikis <= veya_g1 or veya_g2 ;
     end process;
end veri_akisi;
---
entity TT is
      port(tt_g1,tt_g2,tt_eldegirisi : in STD_LOGIC;
           tt_toplam , tt_eldecikisi : out STD_LOGIC);
end TT;
architecture yapisal of TT is
      component yt is
           port ( yt_g1, yt_g2 :in STD_LOGIC;
                  yt_toplam,yt_elde:out STD_LOGIC);
      end component ;
      component VEYA_KAPISI is
           port ( veya_g1, veya_g2 : in STD_LOGIC;
                  veya_cikis :out STD_LOGIC );
      end component;
      signal arakablo1, arakablo2,arakablo3 : STD_LOGIC ;
begin
      blok1 : yt port map ( yt_g1 => tt_g1, yt_g2 => tt_g2, yt_elde =>arakablo1 ,yt_toplam =>arakablo2 );
      blok2 : yt port map ( yt_g1 =>arakablo2 ,yt_g2 =>tt_eldegirisi ,yt_elde =>arakablo3 ,yt_toplam =>tt_toplam );
      blok3 : VEYA_KAPISI port map (veya_g1 =>arakablo1 ,veya_g2 =>arakablo3, veya_cikis =>tt_eldecikisi );
end yapisal;
```

=>

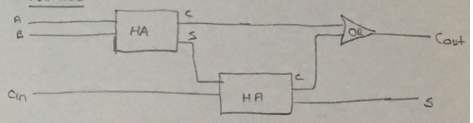
```
architecture davranissal of VEYA_KAPISI is
begin
     process (veya_g1 ,veya_g2)
     begin
          if (veya_g1 ='0' and veya_g2 ='0' ) then
             veya_cikis <='0' ;
          else
             veya_cikis <='1';
          endif ;
     end process;
end davranissal;
```

Design

Half Adder



Full Adder

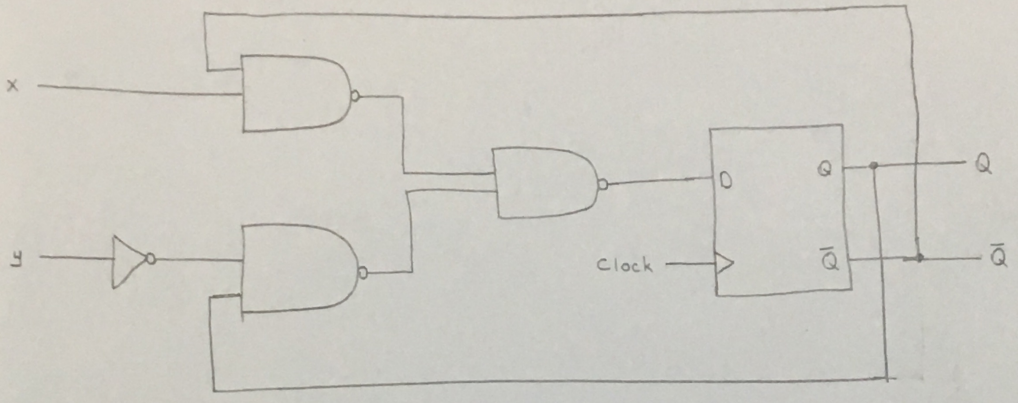


BOT-LAB-FİNAL

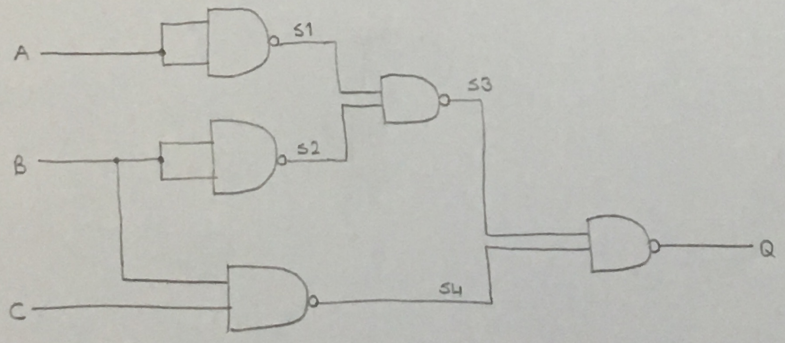
① Aşağıda D Türü flip-flop ve bazı kapılardan oluşturulan XY türü flip-flop'un şekli verilmiştir. Buna göre;

i) Not kapısı, NAND kapısı ve D flip-flop'un VHDL kodlarını davranışsal mimari ile yazınız.

ii) i şıkkında oluşturduğunuz bileşenleri kullanarak XY flip-flop'un yapısal tasarımına ait VHDL kodunu yazınız.



② 1. Sorudaki oluşturduğunuz NAND bileşeninden faydalanarak aşağıdaki devrenin VHDL kodunu yapısal mimari ile yazınız.



③ Yandaki tabloda bir ROM devresinin bellek gözlerinde yazılı veri görülmektedir. Adres $A = A_2A_1A_0$ vektörüyle, veri de $D = D_4D_3D_2D_1D_0$ vektörüyle gösterilmek üzere söz konusu ROM devresinin VHDL kodunu veri akışı mimarisi ile yazınız.

$A_2\ A_1\ A_0$	$D_4\ D_3\ D_2\ D_1\ D_0$
000	11010
001	10111
010	01101
011	11101
100	00011
101	01001
110	00010
111	01110

①

```
entity DFF is
    port(D,Clock:in STD_LOGIC;
         Q,Q' :out STD_LOGIC);
end DFF;
architecture Behavioural of DFF is
begin
    process(Clock,D)
    begin
        if(Clock'Event and Clock='1') then
            Q <= D;
            Q' <= not D;
        endif;
    end process;
end Behavioral;
---
entity NOT is
    port(not_g:in STD_LOGIC;
         not_cikis:out STD_LOGIC);
end NOT;
architecture Behavioral of NOT is
begin
    process(not_g)
    begin
        if(not_g='1') then
            not_cikis <='0';
        else
            not_cikis <='1';
        endif;
    end process;
end Behavioral;
```

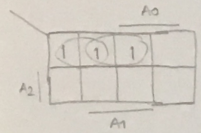
```
entity NAND_KAPI51 is
    port(nand_g1,nand_g2:in STD_LOGIC;
         nand_cikis:out STD_LOGIC);
end NAND_KAPI51;
architecture Behavioral of NAND_KAPI51 is
begin
    process(nand_g1,nand_g2)
    begin
        if(nand_g1='1' and nand_g2='1') then
            nand_cikis <='0';
        else
            nand_cikis <='1';
        endif;
    end process;
end Behavioral;
---
entity DEVRE is
    port(x,y:in STD_LOGIC;   -clock
         Q,Q':inout STD_LOGIC);
end DEVRE;
architecture Yapisal of DEVRE is
component DFF is
    port(D,Clock:in STD_LOGIC;
         Q,Q':out STD_LOGIC);
end component;
component NOT is
    port(not_g:in STD_LOGIC;
         not_cikis:out STD_LOGIC);
end component;
component NAND_KAPI51 is
    port(nand_g1,nand_g2:in STD_LOGIC;
         nand_cikis:out STD_LOGIC);
end component;
signal s1,s2,s3,s4: STD_LOGIC;
```

```
begin
    blok1: NOT port map(not_g=>y,not_cikis=>s1);
    blok2: NAND_KAPI51 port map(nand_g1=>Q',nand_g2=>x,nand_cikis=>s2);
    blok3: NAND_KAPI51 port map(nand_g1=>s1,nand_g2=>Q,nand_cikis=>s3);
    blok4: NAND_KAPI51 port map(nand_g1=>s2,nand_g2=>s3,nand_cikis=>s4);
    blok5: DFF port map(D=>s4,Clock=>d.Clock,Q=>Q,Q'=>Q');
end Yapisal;
```

②

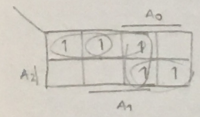
```
architecture Structural of DEVRE1 is
component NAND_KAPI51 is
    port(nand_g1,nand_g2:in STD_LOGIC;
         nand_cikis:out STD_LOGIC);
end component;
signal s1,s2,s3,s4 : STD_LOGIC;
begin
    blok1:NAND_KAPI51 port map(nand_g1=>A,nand_g2=>A,nand_cikis=>s1);
    blok2:NAND_KAPI51 port map(nand_g1=>B,nand_g2=>B,nand_cikis=>s2);
    blok3:NAND_KAPI51 port map(nand_g1=>s1,nand_g2=>s2,nand_cikis=>s3);
    blok4:NAND_KAPI51 port map(nand_g1=>B,nand_g2=>C,nand_cikis=>s4);
    blok5:NAND_KAPI51 port map(nand_g1=>s3,nand_g2=>s4,nand_cikis=>Q);
end Structural;
```

D_4 : (000-001-011)



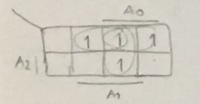
$D_4 = A_0'A_2' + A_1A_2' = A_2'(A_0' + A_1)$

D_3 : (000-011-101-111-010)



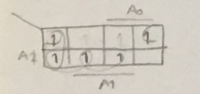
$D_3 = A_2'A_0' + A_2'A_1 + A_1A_0 + A_2A_0 = A_2'(A_0' + A_1) + A_0(A_1 + A_2)$

D_2 : (001-010-111-011)



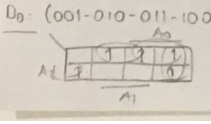
$D_2 = A_2'A_1 + A_1A_0 + A_2'A_0 = A_2'(A_1 + A_0) + A_1A_0$

D_1: (000,001,100,110,111)



$D_1 = A_2'A_1' + A_2A_0' + A_2A_1 + A_1'A_0' = A_2(A_0' + A_1) + A_1'(A_2' + A_0')$

D_0: (001-010-011-100-101)



$D_0 = A_2A_1' + A_2'A_1 + A_2'A_0 + A_1'A_0 = A_2'(A_1 + A_0) + A_1'(A_2 + A_0)$

```
entity devre is
    port(A:in STD_LOGIC_VECTOR(2 downto 0);
         D:out STD_LOGIC_VECTOR(4 downto 0));
end devre;
architecture DataFlow of devre is
begin
    D(0) <= (not A(2) and (A(1) or A(0))) or (not A(1) and (A(2) or A(0)));
    D(1) <= (A(2) and (not A(0) or A(1))) or (not A(1) and (not A(2) or not A(0)));
    D(2) <= (not A(2) and (A(1) or A(0))) or (A(1) and A(0));
    D(3) <= (not A(2) and (not A(0) + A(1))) or (A(0) and (A(1) or A(2)));
    D(4) <= (not A(2) and (not A(0) or A(1)))
end DataFlow;
```

```
entity TFF is
    port(t,Clock:in STD_LOGIC;
         Q,Q':inout STD_LOGIC);
end TFF;
architecture Behavioral of TFF is
signal tmp: STD_LOGIC;
begin
    process(Clock)
    begin
        if(Clock'Event and Clock='1') then
            tmp<='0';
        if(t='0') then
            tmp<=tmp;
        elsif(t='1') then
            tmp<=not(tmp);
        end if;
        Q<=tmp;
        Q'<=not(tmp);
```