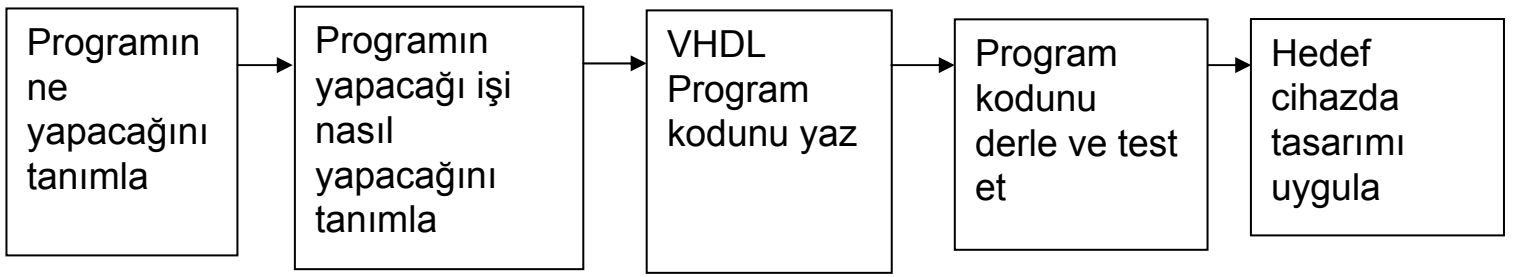
VHDL

Ece Olcay Güneş & S. Berna Örs

Giriş

- VHDL “VHSIC Hardware Description Language” ‘in kısaltmasıdır.
- VHSIC “Very High Speed Integrated Circuit” ‘in kısaltmasıdır.
- VHDL dışında da pekçok donanım tasarlama dilleri mevcuttur. Verilog, AHDL, ABEL and CUPL diğer donanım tasarlama dillerine örnek olarak gösterilebilir.
- VHDL ilk olarak 1981 yılında tasarım yöntemlerinden ve var olan teknolojiden bağımsız olarak tasarım için ABD savunma bakanlığı tarafından geliştirilmiştir. Daha sonra IEEE'ye yönlendirilmiş ilk standart hali 1983 yılında yayımlanmıştır. 1993 yılında geliştirilerek yine IEEE standart dilleri arasında yeralmıştır.

Sayısal lojik tasarımda VHDL kullanımı

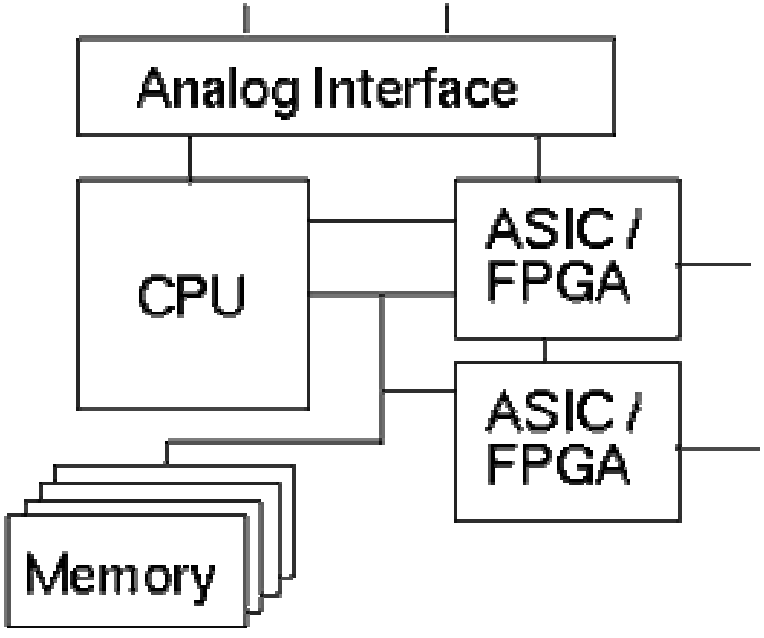


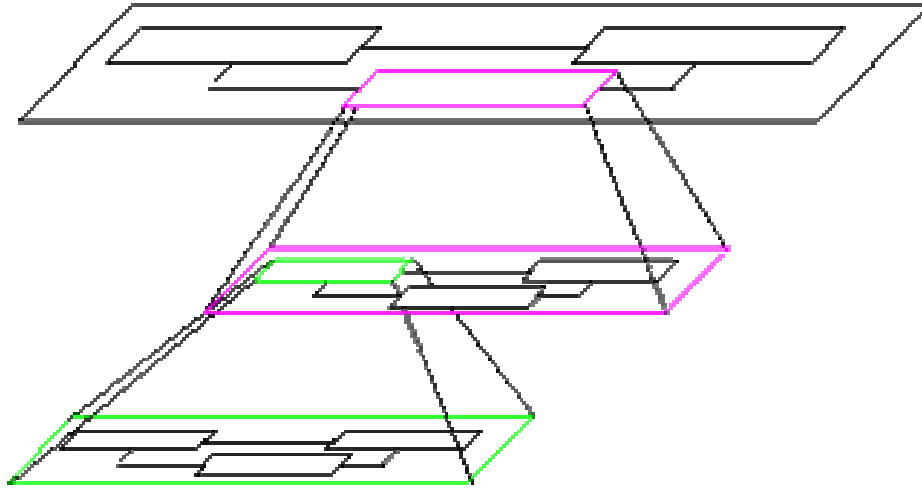
3 farklı devre tanımlama şekli kullanılır

- Yapısal (Structural): Lojik fonksiyon basit kapılar ve bunların bağlantılarıyla tanımlanır.
- Veri Akışı (Data Flow):Kapılardan işaret geçişinin nasıl olacağı tanımlanır. Yapısal ve veri akışı tanımının ikisi de devrenin iç-yapısının ayrıntılarıyla ilgilidir.
- Davranışsal (Behavioral): Kapıların tipi ve bağlantılarının nasıl olduğu ya da bunlar arasında veri akışının nasıl olacağı önemli değildir. Devrenin giriş ve o andaki durumuna göre çıkışlarının ne olacağı davranışsal tanımı oluşturur.
- Lojik devre çok basitse yapısal ve veri akışı tanımı, devre karmaşıksa, davranışsal tanımı, çoğu zaman da 3'ü bir arada kullanılır

Alt Bloklar

- Bir tasarımı daha anlaşılabilir ve değişikliklere açık hale getirebilmek için devreler alt bloklara ayrılır.
- Daha sonra bu bloklar bütün devreyi oluşturmak üzere uygun şekilde bağlanır.





VHDL tasarım birimleri

Entity

Entity: VHDL'de her blok kendi başına bir devre olarak düşünülür ve **entity** olarak adlandırılır.

Verilen bir lojik fonksiyon için bütün giriş ve çıkışları tanımlar. Yani lojik fonksiyonun dış dünyayla bağlantısını tanımlar.

entity entity tanımlayıcı **is**

port (işaret tanımlama);

end entity tanımlayıcı;

Port: port giriş ve çıkış işaretidir. Port ifadesi, her işaret için, port tanımlayıcı, port yönü ve port veri tipini belirlemelidir. Port da 3 yön kullanılır. Giriş için **in**, çıkış için **out**, çift yönlü portlar için **inout** kullanılır. Pek çok değişik veri tipi kullanılabilir.

Veri tipleri:

bit: 0 ve 1 değerini alabilir

bit-vector: aynı isim altında bir dizi 0 ve 1'i göstermek için kullanılır. Bir dizideki herbir elemanın ortak bir tanımlayıcı adı vardır. Mesela dizinin adı A ise, ilk eleman A(0)'a, ikinci eleman A(1)'e karşı düşer.

integer: pozitif ya da negatif tamsayılara karşı düşer

natural: 0'dan başlayıp istenen bir limit değere kadar tamsayılar için kullanılır.

positive: 1'den başlayıp istenen bir limit değere kadar tamsayılar için kullanılır.

Boolean: Doğru ve yanlış olmak üzere 2 değerli bir veri tipidir.

Entity Örneği

```
entity Ornek is
    port (A,B: in bit; Z: out naturel range 0 to 31);
end Ornek;
```

- İlk satır yeni tanımlanacak devrenin ismini bildirir: Ornek
- Son satır tanımlamanın bittiğini gösterir.
- Aradaki satırlar devrenin giriş cıkışlarını tanımlar.
- Port tanımında her satır bir giriş-çıkış listesi, modunu ve veri tipini gösterir.
- İlk satırın dışındaki her satır ; ile bitirilir. Port tanımının tamamı da ; ile bitirilir.

Architecture

Lojik fonksiyonun işlevi **architecture** tarafından belirlenir. Her **entity** için bir **architecture** tanımlanır.

architecture mimari ismi **of** tanımlayıcı ismi **is**

begin

lojik fonksiyonun yapacağı iş burada tanımlanır;

end mimari ismi;

Architecture Örneği

```
entity AND_Gate is
   port(A, B: in bit; X:out bit);
end entity AND_Gate;
Architecture dataflow of AND_Gate is
begin
  X <= A and B;
end dataflow;
```

- İlk satır dataflow isimli mimarinin AND_Gate isimli devreye ait olduğunu gösteriyor.
- **begin** ve **end** arasındaki satırlar AND_Gate in nasıl bir işlem gerçekleyeceğini belirtir.

Yapısal Tanımlamalar

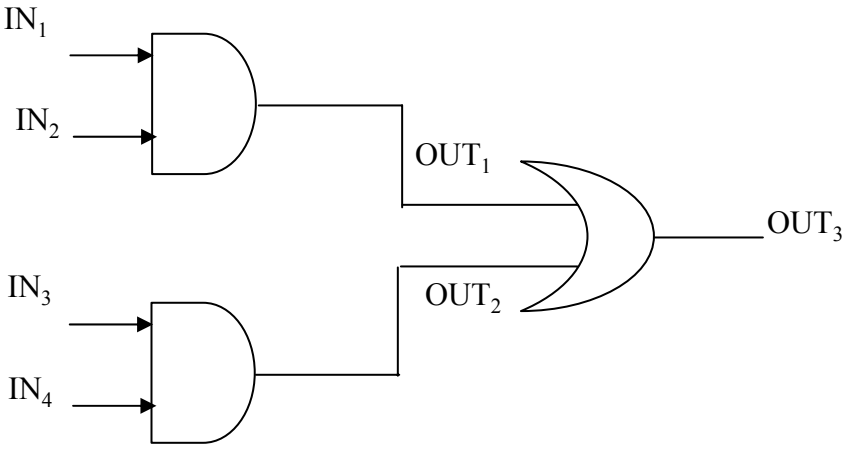
- Tasarımımızdaki temel bloklar **entity** ve karşılık gelen **architecture** kullanılarak tanımlandıktan sonra diğer tasarımlarda kullanılmak üzere birleştirilebilirler. Bir program içerisinde bir eleman çok kere kullanılıyorsa, buna ait entity ve architecture'i defalarca yazmak yerine component kullanılır.
- Bu işlem yapılırken alt bloklar üst bloğun içinde **component** olarak tanımlanırlar.

component elemanın-adı **is**

port (uç tanımları)

end component;

Component Örneği



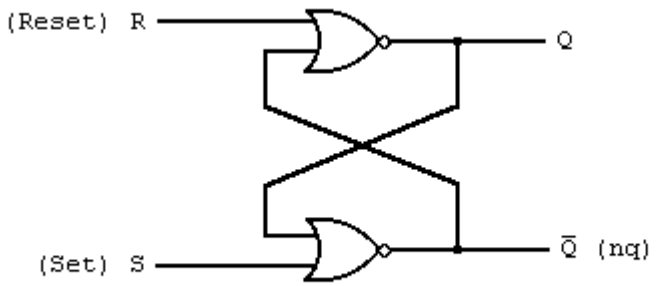
Ara bağlantılara karşı düşer. Entegre içinde dış bağlantısı olmayan elemanlar arası bağlantılardır.

```
entity AND-OR-logic is
    port(IN1, IN2, IN3, IN4: in bit; OUT3: out
      bit);
end AND-OR-logic;
architecture LogicOperation of AND-OR-logic
      is
    component AND_Gate is
      port(A, B:in bit; X:out bit);
    end component;
    component OR_Gate is
      port(A,B:in bit; X:out bit);
    end component;
    signal OUT1, OUT2:bit;
    begin
        G1: AND_Gate
              port map (A⇒IN1, B⇒IN2,
      X⇒OUT1);
        G2: AND_Gate
             port map (A⇒IN3, B⇒IN4,
      X⇒OUT2);
        G3: OR_Gate port map (A⇒OUT1,
      B⇒OUT2, X⇒OUT3);
end LogicOperation;
```

Veri Akışı Tanımlama

- Veri akışı tanımlamada temel blokların (örneğin *and* kapısı) girişlerinin ve çıkışlarının devre içinde nasıl bağlanacağı tanımlanır.
- İşaretlerin devre içinde nasıl akacağı tanımlanır.
- Çıkışların Boole fonksiyonu ifadelerine ihtiyaç vardır.

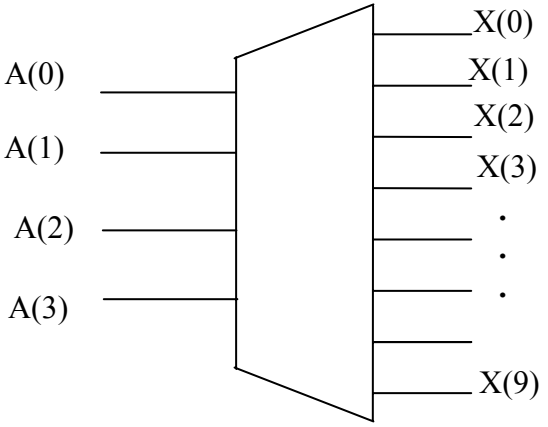
Veri Akışı Tanımlama Örneği 1



SR Latch

```
entity latch is
  port (s,r : in bit; q,nq :
inout bit);
end latch;
architecture dataflow of
latch is
begin
  q<=r nor nq;
  nq<=s nor q;
end dataflow;
```

Sayıları BCD'den Decimal'e çeviren decoder tasarımı (Örnek 2)




```
entity decoder is
    port (A: in bit-vector(0 to3); X:out bit-vector (0 to 9));
end decoder;

architecture BCD-to-decimal of decoder is
begin
   X(0)⇐not A(3) and not A(2) and not A(1) and not A(0);
   X(1)⇐not A(3) and not A(2) and not A(1) and A(0);
   X(2)⇐not A(3) and not A(2) and A(1) and not A(0);
   X(3)⇐not A(3) and not A(2) and A(1) and A(0);
   X(4)⇐not A(3) and A(2) and not A(1) and not A(0);
   X(5)⇐not A(3) and A(2) and not A(1) and A(0);
   X(6)⇐not A(3) and A(2) and A(1) and not A(0);
   X(7)⇐not A(3) and A(2) and A(1) and A(0);
   X(8)⇐A(3) and not A(2) and not A(1) and not A(0);
   X(9)⇐A(3) and not A(2) and not A(1) and A(0);
end BCD-to-decimal;
```

Not

- Bu örnekte birden fazla giriş çıkış söz konusu olduğu için bunlar VHDL'de bir dizi (bit-vector) veri tipi ile gösterilebilir.
- Genel olarak bit dizilerinin büyüklüğü 0'dan n'ye seçilir.
- 0 her zaman dizinin ilk elemanına karşı düşer.
- Bu nedenle n dizideki eleman sayısından 1 eksik olarak seçilmelidir.
- n+1 giriş veya çıkış dizi olarak bit-vector (0 to n) olarak gösterilir.
- Örnekte giriş bit-vector (0 to 3)
- Çıkış bit-vector (0 to 9) biçimindedir.

Durumsal ifadeler

- **IF**

  Bir sayısal sistem belirli durumlar içinden en uygununu seçme durumunda kalabilir. "If " bir Boole fonksiyonunu doğru veya yanlışa götüren ifadedir. Her zaman bir durumsal ifade ile birlikte kullanılır. Eğer ifade doğru ise VHDL ifadesi yerine getirilir. En sonunda mutlaka "end if" olmalıdır**.**

If durumsal ifade **then**

VHDL ifadesi

end if

Örnek: 2 girişli 3 çıkışlı bir devre tasarlanmak istenmektedir. A girişi B girişine eşitse V1 değişkeni (yani çıkış) 1 arttırılır. (Çıkış 3 ile sınırlandırıldığı için natural veri tipi kullanılması uygundur)

```
entity ornek1 is
    port (A,B: in bit; X:out natural range 0 to 7);
end entity ornek;
architecture islem1 of ornek1 is
begin
   process (A,B)
   variable V1: natural range 0 to 7;
   begin
      V1:=0;
      if (A=B) then
         V1:=V1+1;
      end if;
    X⇐V1;
   end process;
end architecture islem1;
```

- **If Then Else**

Else, if içerisinde alternatif bir yol önerir. Eğer VHDL ifadesi yanlışa gidiyorsa, else ifadesini izler.

ÖRNEK:

```
entity örnek2 is

        port (A,B: in bit; X:out natural range 0 to 7);
end entity ornek2;
architecture islem2 of ornek2 is
begin

    process (A,B)
  variable V1: natural range 0 to 7;
  begin
  V1:=6;
    if (A=B) then
       V1:=V1+1;
      else
        V1:=0;
     end if
  X⇐V1;
  end process;
end architecture islem2;
```

- **Elsif**

Bazen giriş koşullarına göre birden fazla seçenek tanımlanabilir. Bu durumda "elsif" ifadesi kullanılır. İçiçe geçmiş if olarak tanımlanabilir.

Örnek: 2 kontrol girişli MUX tasarımı

```
entity MUX2 is
    port(S:in bit_vector (0 to 1); D:in bit_vector (0 to 3); X:out bit);
end entity MUX2;
architecture islem of MUX2 is
begin
  process (S,D)
    begin
      if (S="00") then
        X⇐D(0);
      elsif (S="01") then
        X⇐D(1);
      elsif (S="10") then
        X⇐D(2);
      else
        X⇐D(3);
      end if;
    end process;
end architecture islem;
```

•Case

Case ifadesi içiçe geçmiş if (elsif) ifadesinin bir diğer seçeneğidir. Bir farkı case ifadesi, case bloğuna uygulanan tüm mümkün değerlere aynı kıymeti verir. VHDL keywordlerinden "others" kıymet verilmeyecek girişlerin cevaplarını belirlemeye yarar. Eğer tüm girişler belirliyse others'a gerek yoktur.

```
case ifade is
    when seçenek 1 ⇒
        VHDL ifadesi;
    when seçenek 2 ⇒
        VHDL ifadesi;
    when seçenek 3 ⇒
        VHDL ifadesi;
    when others ⇒
        VHDL ifadesi;
end case
```

Örnek: 1 kontrol girişli MUX

```
entity MUX is
    port(S,D0,D1:in bit; X:out bit);
end entity MUX;
architecture islem for MUX is
begin
  process (S, D0, D1)
  begin
    case S is
       when '0' ⇒
         X⇐D0;
       when '1' ⇒
         X⇐D1;
      end case;
   end process;
end architecture islem;
```

For Çevrimi

Davranışsal yaklaşımda belirli bir iterasyon sayısına sahip ardışıl bir ifadedir. **For,** çevrimin nerede başladığı nerede bittiği belli ise kullanılır. **For** çevrimi 4 bölümden oluşur.

- Giriş noktası: çevrimin başladığı nokta
- İterasyon:
- Terminal test: çevrimin bittiğini belirleyerek çıkış koşullarını test eder
- Çıkış noktası: Çevrimin son bulduğu nokta

for tanımlayıcı (kullanıcı tarafından belirlenir) **in** başlangıç noktası **to** durma noktası **loop**

    VHDL ifadeleri

end loop;

to durma noktasını izler. Başlangıç noktası terminal noktasından küçük ise kullanılır.

downto başlangıç noktası terminal noktasından büyükse ve çevrim terminal noktasına doğru

azalarak hareket ediyorsa kullanılır.

Örnek: 7 girişi olan ve girişine gelen sayı içindeki 1'lerin sayısını sayıp sonucu çıkışa aktaran bir devre tasarlanacaktır. (A girişi için 8 bitlik std_logic_vector kullanılacaktır. X çıkışı için naturel veri tipi kullanılacak hesaplanan değer 0 ile 7 arasında olacaktır. V1 sayılan 1'lerin sayısını tutacaktır.

```
library ieee;
use ieee.std_logic_1164.all;
entity Birsayici is
    port (A: in std_logic_vector (0 to 7); X:out natural range 0 to 7);
end entity Birsayici;
architecture benimsayicim of Birsayici is
begin
    process(A)
    variable V1:natural range 0 to 7;
    begin
       V1:=0;
         for i in 0 to A'length-1 loop
            if (A(i)='1') then
               V1:=V1+1;
           end if;
        end loop;
        X⇐V1;
    end process;
ed architecture benimsayicim;
```

NOT

- Bu örnekte çevrim ve iterasyon sayısı bilindiği için **for** çevrimi kullanılmıştır.
- Sayıcı **std_logic_vector** (std_logic_1164 IEEE tarafından belirlenen bir standart kütüphanedir), 0 başlangıç pozisyonu ile başlamakta ve en son bit okunana kadar devam etmektedir. 1'lerin sayıldığı programda **std_logic_vector** ile belirlenen sayının bit sayısının belirlenmesi gerekir. Burada görüldüğü gibi bu tanımlayıcı ismi (vektörün ismi) **'** ve **length** ile belirlenmiştir. **'length** vektörün bit sayısını verir. Böylece değişik uzunluklu vektörler için program yeniden düzenlenebilir. Bit sayısı bilinmeden program tarafından otomatik olarak dizinin bit sayısı belirlenebilir.

While çevrimi

İterasyon sayısının bilinemediği durumlarda kullanılır. While çevrimi herhangi bir Boole ifadesi sağlanıp çevrimi durduruncaya kadar devam eder. Ancak kilitlenen türden bir kod yazılmamasına dikkat edilmelidir. Bazen Boole ifadesi sağlanamadığı için çevrim son bulamaz. While çevrimi 3 bölümden oluşur.

- Giriş noktası: çevrimin başladığı nokta
- Terminal test: çevrimin bittiğini belirleyerek çıkış koşullarını test eder
- Çıkış noktası: Çevrimin son bulduğu nokta

while boole ifadesi **loop**

VHDL ifadeleri

end loop;

1. Bölümün sonu