6. Alt Devreler, Alt Programlar ve Paketler

VHDL dili tasarımcılar için pek çok kolaylık sağlamaktadır. Özellikle tasarımcıların sıklıkla kullandığı bileşenleri tekrar tekrar yazmasını engelleyen, genel tasarım akışını kolaylaştırıp tasarımda modülerlik sağlayan alt devreler (**component**) son derece faydalı bir özelliktir.

Tasarımda kolaylık sağlayan bir diğer önemli özellik ise **generic** tanımlamasıdır. Bu sayede tasarımdaki **generic** olarak tanımlanan özellikler kolayca değiştirmek mümkün olmaktadır. Bu bölümde anlatılacak diğer tasarım bileşenleri ise **function**, **procedure** ve **package** kullanımı olacaktır. Bu tasarım bileşenleri kullanılarak yapılan çalışmalar hem modüler hem de istendiği anda ihtiyaçlara uygun olarak güncellenebilir şekilde tasarlanabilecektir.

6.1. VHDL'de Alt Devre Tanımlama – PORT MAP Kullanımı

VHDL varlığında, bir kaynak kod dosyası başka bir kaynak dosyası altında alt devre olarak kullanılabilir. VHDL dilinde alt devre **component** olarak adlandırılır. Giriş çıkış portlarından oluşan bileşenlerin tanımı, **component** tanımlamaya uygun olarak yapılmalıdır. Bu tanımlama, ana kod içerisinde tanımlama bölgesinde (**architecture Behavioral of** varlik_adi **is** ile **begin** arasında) veya **package** içerisinde tanımlanabilmektedir. Ana kod içerisinde alt devre aşağıdaki gibi tanımlanabilmektedir.

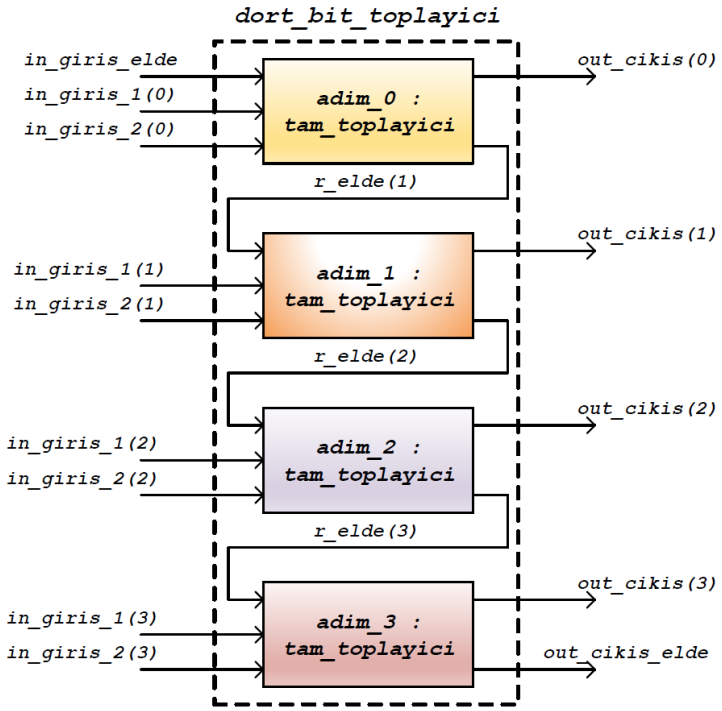
```
durum_adi : component_adi port map (
   asil_ad => mevcut_ad
{, asil_ad => mevcut_ad} ) ;
```

asil_ad, alt devre varlığında tanımlı portların isimleridir. **mevcut_ad** ise sinyal veya ana kod portlarıdır. **port map** tanımlama içerisinde **asil_ad** kullanmak zorunlu değildir. Aşağıda **component** tanımlama şekli gösterilmiştir.

```
architecture Behavioral of varlik_adi is
      ..
      ..
component bilesen_adi
    generic (parametre_adi : integer := varsayilan_deger{;
         parametre_adi : integer := varsayilan_deger} ) ;
    port( port_adi {, port_adi} : [port_modu] tip_adi {;
      port_adi {, port_adi} : [port_modu] type_adi} ) ;
end component;
..
```

```
..
begin
```

Şekil 6-1'de 4 bitlik toplayıcı devresinin, **tam_toplayici** alt devresinin kullanımı ile tasarlanmasına ilişkin blok şema gösterilmiştir. Şekilden de görüleceği üzere **dort_bit_toplayici** 4 bitlik **in_giris_1** ve **in_giris_2** giriş portlarına, 1 bitlik **in_giris_elde** giriş portuna, 4 bitlik **out_cikis** çıkış portuna ve 1 bitlik **out_cikis_elde** çıkış portuna sahiptir. **tam_toplayici** varlığı ise 1 bitlik 3 giriş portuna ve 1 bitlik 2 çıkış portuna sahiptir.



Şekil 6-1 4 bitlik tam toplayıcı devresinin alt devre kullanılarak gösterimi

in_giris_elde girişi ile birlikte **in_giris_1** ve **in_giris_2** girişlerinin en anlamsız bitleri (**in_giris_1(0)** ve **in_giris_2(0)**) **adim_0** ile tanımlanan **tam_toplayici** alt devresine giriş olarak verilmektedir. **adim_0** alt devrenin sonucunda elde edilen ilk çıkış biti **out_cikis** çıkış değerinin en anlamsız biti (**out_cikis(0)**) olmaktadır. Diğer çıkış parametresi olan elde değeri **r_elde** sinyalinin 1. bitine atanmakta ve bu bit değeri aynı zamanda **adim_1** ile tanımlanan **tam_toplayici** alt devresine giriş olarak verilmektedir.

r_elde(1) girişi ile birlikte **in_giris_1(1)** ve **in_giris_2(1)** bitleri **adim_1** ile tanımlanan **tam_toplayici** alt devresine giriş olarak verilmektedir. Bu alt devrenin sonucunda elde edilen ilk çıkış biti **out_cikis(1)** çıkış değerine atanmaktadır. Diğer çıkış parametresi olan elde değeri **r_elde** sinyalinin 2. bitine atanmakta ve bu bit değeri aynı zamanda **adim_2** ile tanımlanan **tam_toplayici** alt devresine giriş olarak verilmektedir.

r_elde(2) girişi ile birlikte **in_giris_1(2)** ve **in_giris_2(2)** bitleri **adim_2** ile tanımlanan **tam_toplayici** alt devresine giriş olarak verilmektedir. Bu alt devrenin sonucunda elde edilen ilk çıkış biti **out_cikis(2)** çıkış değerine atanmaktadır. Diğer çıkış parametresi olan elde değeri **r_elde** sinyalinin 3. bitine atanmakta ve bu bit değeri aynı zamanda **adim_3** ile tanımlanan **tam_toplayici** alt devresine giriş olarak verilmektedir.

r_elde(3) girişi ile birlikte **in_giris_1(3)** ve **in_giris_2(3)** bitleri **adim_3** ile tanımlanan **tam_toplayici** alt devresine giriş olarak verilmektedir. Bu alt devrenin sonucunda elde edilen ilk çıkış biti

out_cikis(3) çıkış değerine atanmaktadır. Diğer çıkış parametresi olan elde değeri ise **out_cikis_elde** çıkışına atanarak toplama işleminin sonucu elde edilmektedir.

Uygulamada dikkat edilmesi gereken husus, **adım_1** ile tanımlanan alt devrenin doğru sonuç üretebilmesi için **adım_0** ile tanımlanan alt devrenin sonucunu bekleyeceğidir. Aynı şekilde **adım_2** ile tanımlanan alt devre **adim_1** ile tanımlanan alt devrenin, **adim_3** ile tanımlanan alt devre ise **adim_2** ile tanımlanan alt devrenin sonucunu beklemektedir.

Örnek 6:1: Şekil 5.1'de verilen blok şemaya ait **dort_bit_toplayici.vhd** VHDL kodu aşağıda verilmiştir. **dort_bit_toplayici** varlığının port tanımlamaları 5-11. satırlar arasında yapılmıştır. **in_giris_elde** portu **in** modunda **std_logic** tipindedir. **in_giris_1** ve **in_giris_2** portları **in** modunda 4 bitlik **std_logic_vector** tipindedir. **out_cikis** portu **out** modunda 4 bitlik **std_logic_vector** tipindedir. **out_cikis_elde** portu **out** modunda **std_logic** tipindedir. **tam_toplayici** alt devresinin tanımlama işlemleri 17-25. satırlar arasında yapılmıştır.

30-31. satırlarda **adim_0** isimli **tam_toplayici** alt devresin port atamaları yapılmıştır. Görüleceği üzere atam işlemleri yapılırken **asil_ad** kullanılmamıştır. Şekil 5.1'de gösterildiği gibi **adim_0** alt devresinde **in_giris_elde**, **in_giris_1(0)** ve **in_giris_2(0)** giriş portları alt devreye giriş olarak verilmiştir. **adim_0** alt devresi çıkışlarından sonuç değerini tutan port direk olarak **out_cikis(0)** çıkış portuna bağlanmıştır. Elde değerini tutan çıkış portu **r_elde** sinyalinin 1. bitine atanarak **adim_1** alt devresine elde girişi olarak verilmektedir.

32-33. satırlarda **adim_1** isimli **tam_toplayici** alt devresin port atamaları yapılmıştır. Görüleceği üzere atam işlemleri yapılırken **asil_ad** kullanılmamıştır. **adim_1** alt devresinde **r_elde(1)** sinyali ile birlikte **in_giris_1(1)** ve **in_giris_2(1)** giriş portları alt devreye giriş olarak verilmiştir. **adim_1** alt devresi çıkışlarından sonuç değerini tutan port direk olarak **out_cikis(1)** çıkış portuna bağlanmıştır. Elde değerini tutan çıkış portu **r_elde** sinyalinin 2. bitine atanarak **adim_2** alt devresine elde girişi olarak verilmektedir.

34-35. satırlarda **adim_2** isimli **tam_toplayici** alt devresin port atamaları yapılmıştır. Görüleceği üzere atam işlemleri yapılırken **asil_ad** kullanılmamıştır. **adim_2** alt devresinde **r_elde(2)** sinyali ile birlikte **in_giris_1(2)** ve **in_giris_2(2)** giriş portları alt devreye giriş olarak verilmiştir. **adim_2** alt devresi çıkışlarından sonuç değerini tutan port direk olarak **out_cikis(2)** çıkış portuna bağlanmıştır. Elde değerini tutan çıkış portu **r_elde** sinyalinin 3. bitine atanarak **adim_3** alt devresine elde girişi olarak verilmektedir.

36-41. satırlarda **adim_3** isimli **tam_toplayici** alt devresin port atamaları yapılmıştır. Görüleceği üzere atam işlemleri yapılırken **asil_ad** kullanılmıştır. Kullanımdan da görüleceği üzere önce **component** içerisinde tanımlanan giriş çıkış değerlerine karşılık gelen atamalarının yapıldığı görülmektedir. Örneğin **r_elde(3)** sinyali **tam_toplayici** alt devresinin **in_giris_elde** giriş portuna bağlanmıştır. **adim_3** alt devresinde **r_elde(3)** sinyali ile birlikte **in_giris_1(3)** ve **in_giris_2(3)** giriş portları alt devreye giriş olarak verilmiştir. **adim_3** alt devresi çıkışlarından sonuç değerini tutan port direk olarak **out_cikis(3)** çıkış portuna bağlanmıştır. Elde değerini tutan çıkış portu **out_cikis_elde** çıkış portuna bağlanmıştır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity dort_bit_toplayici is
5.   Port(
6.     in_giris_elde : in std_logic;
7.     in_giris_1 : in std_logic_vector(3 downto 0);
8.     in_giris_2 : in std_logic_vector(3 downto 0);
9.     out_cikis : out std_logic_vector(3 downto 0);
10.    out_cikis_elde : out std_logic
```

```
11.  );
12.end dort_bit_toplayici;
13.
14.architecture Behavioral of dort_bit_toplayici is
15.
16.  component tam_toplayici
17.  Port (
18.    in_giris_elde : in std_logic;
19.    in_giris_1 : in std_logic;
20.    in_giris_2 : in std_logic;
21.    out_cikis : out std_logic;
22.    out_cikis_elde : out std_logic
23.  );
24.  end component;
25.
26.  signal r_elde : std_logic_vector(1 to 3) ;
27.
28.begin
29.
30.  adim_0: tam_toplayici port map (in_giris_elde, in_giris_1(0),
31.          in_giris_2(0), out_cikis(0), r_elde(1) ) ;
32.  adim_1: tam_toplayici port map (r_elde(1), in_giris_1(1),
33.          in_giris_2(1), out_cikis(1), r_elde(2) ) ;
34.  adim_2: tam_toplayici port map (r_elde(2), in_giris_1(2),
35.          in_giris_2(2), out_cikis(2), r_elde(3) ) ;
36.  adim_3: tam_toplayici port map (
37.          in_giris_elde => r_elde(3),
38.          in_giris_1 => in_giris_1(3),
39.          in_giris_2 => in_giris_2(3),
40.          out_cikis => out_cikis(3),
41.          out_cikis_elde => out_cikis_elde) ;
42.
43.end Behavioral;
```

Örnekte alt devre olarak tam toplayıcı devresine ait **tam_toplayici.vhd** VHDL kodu aşağıda verilmiştir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity tam_toplayici is
```

```
5.    Port (
6.      in_giris_elde : in std_logic;
7.      in_giris_1 : in std_logic;
8.      in_giris_2 : in std_logic;
9.      out_cikis : out std_logic;
10.     out_cikis_elde : out std_logic
11.   );
12.end tam_toplayici;
13.
14.architecture Behavioral of tam_toplayici is
15.
16.begin
17.
18.   out_cikis <= in_giris_elde xor in_giris_1 xor in_giris_2;
19.   out_cikis_elde <= (in_giris_elde and in_giris_1) or
20.                     (in_giris_elde and in_giris_2) or
21.                     (in_giris_1 and in_giris_2);
22.
23.end Behavioral;
```

6.2. GENERIC Kullanarak ENTITY Tanımlama

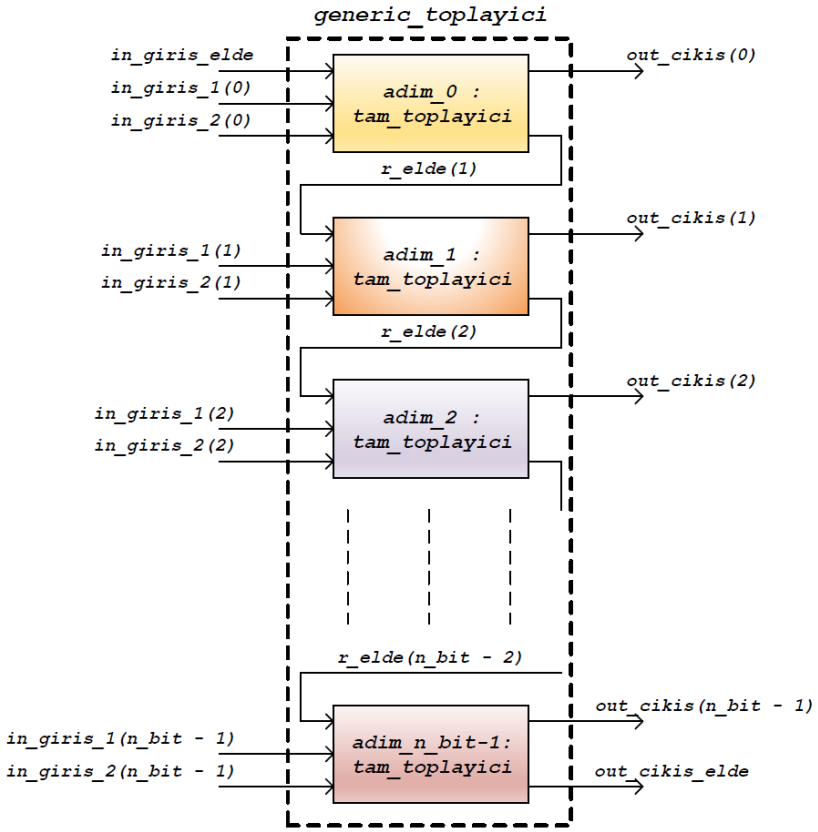
4 bitlik bir toplayıcı için art arda 4 adet tam toplayıcı devresinin birbirine bağlanması gerekmektedir. Eğer tasarlanan sistem 4 bitlik yerine artık 8 bitlik bir toplayıcı olarak kullanılacaksa bu sefer 8 adet tam toplayıcı devresinin birbirine bağlanması gerekmektedir. Bu sayı arttıkça kod içerisinde yapılacak değişikliklerin sayısı da artmaktadır (Şekil 6-2). Bu durumu kontrol altına alabilmek amacı ile VHDL dilinde parametrik tasarım yapılabilmesi için **generic** tanımlama mevcuttur.

generic tanımlaması yapılarak tasarlanan devre parametrik özelliğe sahip olmaktadır. Özellikle büyük tasarımlarda **generic** kullanımıyla yapılmak istenen değişiklikler kolaylıkla yapılabilmektedir.

generic olarak tanımlanan ifadeler yapılacak değişikliklerle tasarlanan tasarımın tamamı değiştirilmektedir. Bu nedenle kullanıcı tasarım içerisinde yapacağı değişiklikleri tek tek yapmak yerine **generic** ile daha kolay yapma imkânına sahip olmaktadır.

Örnek 6:2: Şekil 6-2'de verilen blok şemaya ait **generic_toplayici.vhd** VHDL kodu aşağıda verilmiştir. Kodda parametrik yapıda tam toplayıcı işlemi gerçekleştirilmiştir. Parametrik işlem kodda 5-7. satırlar arasında tanımlanan **n_bit** parametresi ile sağlanmaktadır.

generic_toplayici varlığımızın port tanımlama işlemleri 8-14. satırlar arasında yapılmaktadır. **in_giris_elde** portu **in** modunda **std_logic** tipindedir. **in_giris_1** ve **in_giris_2** portları **in** modunda **n_bit** bitlik **std_logic_vector** tipindedir. **out_cikis** portu **out** modunda **n_bit** bitlik **std_logic_vector** tipindedir. **out_cikis_elde** portu **out** modunda **std_logic** tipindedir.



Şekil 6-2 n bitlik tam toplayıcı devresinin alt devre kullanılarak gösterimi

6. satırda tanımlanan **n_bit : integer := 8** söz dizimi ile tasarlanacak sistemimizde kullanılacak olan **n_bit** değişkeni 8 değerini alacaktır. **n_bit** değerini belirlenmesi ile birlikte **generic_toplayici** varlığımıza ait **in_giris_1**, **in_giris_2** ve **out_cikis** portları 8 bit olacaktır.

Mimari içerisinde 35. satırda tanımlanan **for n_i in 0 to n_bit - 1 generate** ifadesi ile döngü içerisinde tanımlanacak olan ifade ile oluşturulacak devreden **n_bit** adedinin birbirine bağlanacağı ifade edilmektedir. 36. satırda **adim: tam_toplayici port map** ifadesi ile **n_bit** adet tam toplayici devresinin bir biri ardına bağlanacağı anlaşılmaktadır.

Tasarımımızda toplayıcının bit uzunluğunu değiştirmek istediğimizde kod içersinde **n_bit** ve **n_bit - 1** gördüğümüz yerlerdeki tüm değerleri ayrı ayrı değiştirmemiz gerekmektedir. Tasarlanın devrenin de büyüklüğü düşünüldüğü zaman yapılacak değişiklikler daha da karmaşık hale gelebilmektedir. Bu nedenle **generic** kullanımı ile tasarımda değişikliklerin kolayca yapılabildiği görülmektedir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity generic_toplayici is
5.   Generic(
6.     n_bit : integer := 8
7.   );
8.   Port(
9.     in_giris_elde : in std_logic;
```

```
    in_giris_1 : in std_logic_vector(n_bit - 1 downto 0);
    in_giris_2 : in std_logic_vector(n_bit - 1 downto 0);
    out_cikis : out std_logic_vector(n_bit - 1 downto 0);
    out_cikis_elde : out std_logic
  );
end generic_toplayici;

architecture Behavioral of generic_toplayici is

  component tam_toplayici
  Port (
    in_giris_elde : in std_logic;
    in_giris_1 : in std_logic;
    in_giris_2 : in std_logic;
    out_cikis : out std_logic;
    out_cikis_elde : out std_logic
  );
  end component;

  signal r_elde : std_logic_vector(0 to n_bit) ;

begin

  r_elde(0) <= in_giris_elde;

  for_kontrol: for n_i in 0 to n_bit - 1 generate
    adim: tam_toplayici port map (
          in_giris_elde => r_elde(n_i),
        in_giris_1 => in_giris_1(n_i),
        in_giris_2 => in_giris_2(n_i),
        out_cikis => out_cikis(n_i),
        out_cikis_elde => r_elde(n_i + 1)
  );
  end generate for_kontrol;

  out_cikis_elde <= r_elde(n_bit);

end Behavioral;
```

Aşağıda verilen **tb_toplayici.vhd** VHDL kodunda farklı bit uzunluklarında **generic_toplayici** alt devreleri oluşturulmaktadır. **generic_toplayici_4_bit** etiketli alt devre de görüleceği üzere **generic** içerisinde **n_bit => 4** atama işlemi yapılmıştır. Bu nedenle bu alt devre 4 bitlik olacaktır. **generic_toplayici_8_bit** etiketli alt devre de ise **generic** atama işlemi yapılmamıştır. Bu durumlarda ise **component** içerisinde tanımlanan **n_bit** değeri varsayılan değer olarak atanmaktadır. Bu tasarım için 8 bitlik bir toplayıcı oluşturacaktır. **port map** içerisinde çıkış değerleri kullanılmayacak ise uygulamada görüldüğü gibi **open** ile açık hale getirilebilmektedir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity tb_toplayici is
5. end tb_toplayici;
6.
7. architecture Behavioral of tb_toplayici is
8.
9.   component generic_toplayici
10.  Generic(
11.    n_bit : integer := 8
12.  );
13.  Port (
14.    in_giris_elde : in std_logic;
15.    in_giris_1 : in std_logic_vector(n_bit - 1 downto 0);
16.    in_giris_2 : in std_logic_vector(n_bit - 1 downto 0);
17.    out_cikis : out std_logic_vector(n_bit - 1 downto 0);
18.    out_cikis_elde : out std_logic
19.  );
20.  end component;
21.
22.  signal in_giris4_1 : std_logic_vector(3 downto 0) := X"2";
23.  signal in_giris4_2 : std_logic_vector(3 downto 0) := X"1";
24.  signal out_cikis4 : std_logic_vector(3 downto 0);
25.  signal out_cikis_elde4 : std_logic;
26.
27.  signal in_giris8_1 : std_logic_vector(7 downto 0) := X"12";
28.  signal in_giris8_2 : std_logic_vector(7 downto 0) := X"22";
29.  signal out_cikis8 : std_logic_vector(7 downto 0);
30.
31.begin
32.
33.  generic_toplayici_4_bit : generic_toplayici
```

```
34.  Generic map( n_bit => 4 )
35.  Port map (
36.    in_giris_elde => '0',
37.    in_giris_1 => in_giris4_1,
38.    in_giris_2 => in_giris4_2,
39.    out_cikis => out_cikis4,
40.    out_cikis_elde => out_cikis_elde4
41.  );
42.
43.  generic_toplayici_8_bit : generic_toplayici
44.  Port map (
45.    in_giris_elde => '0',
46.    in_giris_1 => in_giris8_1,
47.    in_giris_2 => in_giris8_2,
48.    out_cikis => out_cikis8,
49.    out_cikis_elde => open
50.  );
51.
52.end Behavioral;
```

6.3. Function

Fonksiyon hesaplama değerleri veya davranış tanımlaması için bir algoritmanın tanımlı olduğu alt programdır. Fonksiyonun önemli özelliği belirli bir tipe ait değerde dönüş sağlamasıdır. Fonksiyonun bu özelliği diğer alt program tiplerinden en önemli farkıdır. Aşağıda VHDL dilinde fonksiyon tanımlama sözdizimi verilmiştir.

```
function fonksiyon_adi (fonksiyon_girisleri : giris_tipleri) return
donus_tipi is
      tanımlamalar
begin
      Sıralı Söz dizimleri
end fonksiyon_adi;
```

Aşağıda **ayni_giris** isimli fonksiyonda 4 bitlik giriş sinyallerinin bir biri ile aynı olması durumunda 1 aksi durumda 0 döndürülmektedir. **ayni_giris** fonksiyonun **in_giris_1** ve **in_giris_2** bitleri **std_logic_vector** tipinde 4 bit olarak tanımlanmıştır. Fonksiyonun döndürdüğü değer ise **std_logic** tipindedir.

```
function  ayni_giris(in_giris_1,  in_giris_2  :  std_logic_vector(3
downto 0)) return std_logic is
begin
  if in_giris_1 = in_giris_2 then
    return '1';
```

```
    else
      return '0';
    end if;
  end ayni_giris;
```

Bir başka örnekte ise **toplayici_4_bit** fonksiyonu ile 4 bitlik toplayıcı işlemi gerçekleştirilmektedir. 1. satırda fonksiyon tanımlama işlemi yapılmıştır. Tanımlama işleminde **in_giris_1** ve **in_giris_2** ifadeleri 4 bitlik **std_logic_vector** tipinde ve **in_giris_elde** ifadesi bir bitlik **std_logic** tipindedir. Fonksiyon **std_logic_vector** tipinde değer döndürmektedir. 19. satırda ise toplama sonucun tutulduğu 5 bitlik **std_logic_vector** tipindeki **v_toplam** değişkeni döndürülmektedir.

```
1. function toplayici_4_bit (in_giris_1, in_giris_2 : std_logic_vector(3
   downto 0); in_giris_elde: std_logic) return std_logic_vector is
2.
3.   variable v_elde : std_logic;
4.   variable v_toplam : std_logic_vector(4 downto 0);
5.
6. begin
7.
8.   v_elde:= in_giris_elde;
9.   v_toplam := (others => '0');
10.
11.  for n_i in 0 to 3 loop
12.    v_toplam(n_i) := in_giris_1(n_i) xor in_giris_2(n_i) xor v_elde;
13.    v_elde := (in_giris_1(n_i) and in_giris_2(n_i)) or
14.              (in_giris_1(n_i) and v_elde) or
15.              (in_giris_2(n_i) and v_elde);
16.  end loop;
17.
18.  v_toplam (4) := v_elde;
19.  return v_toplam;
20.
21.end toplayici_4_bit;
```

Örnek 6.3: Yukarıda tanımlanan **toplayici_4_bit** fonksiyonunun kullanıldığı **function_ornek.vhd** VHDL kodu aşağıda verilmiştir. Kodda fonksiyon tanımlama işlemleri 16-35. satırlar arasında yapılmaktadır. 41. satırda **toplayici_4_bit** fonksiyonu çağrılmaktadır. Fonksiyon girişleri **in_giris_1**, **in_giris_2** ve **in_giris_elde** giriş portlarıdır. Fonksiyonun dönderdiği değer **r_toplama_sonuc** sinyaline atanmaktadır. 43. satırda toplama sonucu **out_cikis** çıkış portuna **r_toplama_sonuc** sinyalinin sağdan 4 biti atanmaktadır. 44. satırda elde sonuç değeri **out_cikis_elde** portuna **r_toplama_sonuc** sinyalinin en anlamlı biti atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity function_ornek is
5.   Port (
6.     in_giris_elde : in std_logic;
7.     in_giris_1 : in std_logic_vector(3 downto 0);
8.     in_giris_2 : in std_logic_vector(3 downto 0);
9.     out_cikis : out std_logic_vector(3 downto 0);
10.    out_cikis_elde : out std_logic
11.  );
12.end function_ornek;
13.
14.architecture Behavioral of function_ornek is
15.
16.  function     toplayici_4_bit     (in_giris_1,     in_giris_2     :
   std_logic_vector(3 downto 0); in_giris_elde: std_logic)
17.  return std_logic_vector is
18.    variable v_elde : std_logic;
19.    variable v_toplam : std_logic_vector(4 downto 0);
20.
21.  begin
22.
23.    v_elde := in_giris_elde;
24.    v_toplam := (others => '0');
25.
26.    for n_i in 0 to 3 loop
27.      v_toplam(n_i)   :=   in_giris_1(n_i)   xor   in_giris_2(n_i)   xor
   v_elde;
28.      v_elde := (in_giris_1(n_i) and in_giris_2(n_i)) or
29.                (in_giris_1(n_i) and v_elde) or
30.                (in_giris_2(n_i) and v_elde);
31.    end loop;
32.
33.    v_toplam (4) := v_elde;
34.    return v_toplam;
35.  end toplayici_4_bit;
36.
37.  signal r_toplama_sonuc : std_logic_vector(4 downto 0) := (others =>
   '0');
38.
```

```
39.begin
40.
41.  r_toplama_sonuc    <=    toplayici_4_bit(in_giris_1,    in_giris_2,
   in_giris_elde);
42.
43.  out_cikis <= r_toplama_sonuc(3 downto 0);
44.  out_cikis_elde <= r_toplama_sonuc(4);
45.
46.end Behavioral;
```

6.4. Procedure

Fonksiyonun aksine **procedure** VHDL dilinde tanımlanan diğer söz dizimlerini de kullanılabilmektedir. Fonksiyon gibi bir değer döndürmez. VHDL kodundaki pozisyonlarına (mimari veya process içerisinde) bağlı olarak eş zamanlı veya sıralı olarak gerçeklenebilmektedir.

procedure modül içerisinde VHDL kodunun analizini kolaylaştırmaktadır. Çıkış parametrelerini kullanarak değişkenlerin sayılarını döndürebilir. Söz dizimi aşağıdaki gibidir :

```
procedure procedure_adi [(procedures_arayuz_listesi)] is
    tanımlamalar
begin
    Söz dizimleri
end procedures_adi;
```

Aşağıda 4 bitlik tam toplayıcı devresinin **procesdure** kullanılarak gerçeklenmesi gösterilmiştir.

```
1. procedure toplayici_4_bit(
2.   in_giris_elde : in std_logic;
3.   in_giris_1 : in std_logic_vector(3 downto 0);
4.   in_giris_2 : in std_logic_vector(3 downto 0);
5.   out_cikis : out std_logic_vector(3 downto 0);
6.   out_cikis_elde : out std_logic ) is
7.
8.   variable v_elde: std_logic_vector(4 downto 0);
9.
10.begin
11.
12.  v_elde(0) := in_giris_elde;
13.  for n_i in 0 to 3 loop
14.    out_cikis   (n_i)   <=   in_giris_1(n_i)   xor   in_giris_2(i)   xor
   v_elde(n_i);
```

```
15.     v_elde(n_i + 1) := (in_giris_1(n_i) and in_giris_2(n_i)) or
16.                        (in_giris_2(n_i) and v_elde(n_i)) or
17.                        (in_giris_2(n_i) and v_elde(n_i));
18.   end loop;
19.
20.   out_cikis_elde  <= v_elde(4);
21.
22.end toplayici_4_bit;
```

Örnek 6.4: Yukarıda tanımlanan **toplayici_4_bit procedure**'nin kullanıldığı **procedure_ornek.vhd** VHDL kodu aşağıda verilmiştir. Kodda **procedure** tanımlama işlemleri 16-38. satırlar arasında yapılmaktadır. 44. Satırda **toplayici_4_bit** fonksiyonu çağrılmaktadır. Fonksiyon girişleri **in_giris_1**, **in_giris_2** ve **in_giris_elde** giriş portlarıdır. Fonksiyonun dönderdiği değer **r_toplama_sonuc** sinyaline atanmaktadır. 45. satırda toplama sonucu **out_cikis** çıkış portuna **r_toplama_sonuc** sinyalinin sağdan 4 biti atanmaktadır. 46. satırda elde sonuç değeri **out_cikis_elde** portuna **r_toplama_sonuc** sinyalinin en anlamlı biti atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity procedure_ornek is
5.   Port (
6.     in_giris_elde : in std_logic;
7.     in_giris_1 : in std_logic_vector(3 downto 0);
8.     in_giris_2 : in std_logic_vector(3 downto 0);
9.     out_cikis : out std_logic_vector(3 downto 0);
10.    out_cikis_elde : out std_logic
11.  );
12.end procedure_ornek;
13.
14.architecture Behavioral of procedure_ornek is
15.
16.  procedure toplayici_4_bit(
17.    in_giris_elde : in std_logic;
18.    in_giris_1 : in std_logic_vector(3 downto 0);
19.    in_giris_2 : in std_logic_vector(3 downto 0);
20.    out_cikis : out std_logic_vector(3 downto 0);
21.    out_cikis_elde : out std_logic ) is
22.
23.    variable v_elde: std_logic_vector(4 downto 0);
24.
```

```
25.  begin
26.
27.    v_elde(0) := in_giris_elde;
28.    for n_i in 0 to 3 loop
29.      out_cikis (n_i) := in_giris_1(n_i) xor in_giris_2(n_i) xor
   v_elde(n_i);
30.      v_elde(n_i + 1) := (in_giris_1(n_i) and in_giris_2(n_i)) or
   (in_giris_2(n_i) and v_elde(n_i)) or (in_giris_2(n_i) and
   v_elde(n_i));
31.    end loop;
32.
33.    out_cikis_elde := v_elde(4);
34.
35.  end toplayici_4_bit;
36.
37.begin
38.
39.  process(in_giris_elde, in_giris_1, in_giris_2)
40.    variable v_cikis : std_logic_vector(3 downto 0);
41.    variable v_cikis_elde : std_logic;
42.  begin
43.
44.    toplayici_4_bit(in_giris_elde, in_giris_1, in_giris_2, v_cikis,
   v_cikis_elde );
45.    out_cikis <= v_cikis;
46.    out_cikis_elde <= v_cikis_elde;
47.  end process;
48.
49.end Behavioral;
```

6.5. VHDL'de PACKAGE Kullanımı

package, VHDL dilinde tanımlanacak olan **type**, **function** gibi genel ifadeleri tanımlamada kullanılır. **package** iki temel bömlümden oluşur:

- *paket tanımlama*
- *paket gövdesi*

Genel olarak paket dosyasında tanımlama işlemi aşağıdaki gibi yapışmaktadır.

```
package paket_adi is
    Tip ve alt tip tanimlama
    Alt programlar
```

```
    Sabitler, sinyaller, vb
end paket_adi;

package body paket_adi is
    Öncelikli tanımlamalar
    Sabitler
    Altprogramlar
    Tip ve alt tip tanimlama
    Alt programlar
    Sabitler, sinyaller, vb
end paket_adi;
```

Örnek 6.5: Aşağıda örnek olarak oluşturulan **benim_paketim.vhd** VHDL paketi verilmiştir. **benim_paketim** paketi içerisinde tanımlama bölgesinde 6-7. Satırlarda bir bitlik **std_logic** tipinde **r_giris_1**, **r_giris_2** sinyalleri, 8. satırda **integer** tipinde **VERI_UZUNLUGU** sabiti ve 9-10. satırlarda **VERI_UZUNLUGU** boyutunda **in_giris_1** ve **in_giris_2** sinyalleri, 12-14. satırlarda **t_Kelime** tipi ve bu tip ile tanımlanmış **r_Kelime_1**, **r_Kelime_2** sinyalleri, 16. satırda **buyuk_bul** isimli fonksiyonun tanımı ve 18-29. satırlarda **generic_toplama** alt devresine ilişkin **component** tanımlama işlemleri yapılmıştır. Paket gövdesi bölümünde ise 34-42. satırlarda **buyuk_bul** fonksiyonunun işlevi tanımlanmıştır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. package benim_paketim is
5.
6.   signal r_giris_1 : std_logic := '0';
7.   signal r_giris_2 : std_logic := '0';
8.   constant VERI_UZUNLUGU : integer := 6;
9.   signal in_giris_1 : std_logic_vector(VERI_UZUNLUGU - 1 downto 0) :=
   (others => '0');
10.  signal in_giris_2 : std_logic_vector(VERI_UZUNLUGU - 1 downto 0) :=
   (others => '0');
11.
12.  type t_Kelime is array (9 downto 0) of std_logic;
13.  signal r_Kelime_1 : t_kelime := (others => '0');
14.  signal r_Kelime_2 : t_kelime := (others => '0');
15.
16.  function buyuk_bul(in_Kelime_1, in_Kelime_2 : t_Kelime) return
   t_Kelime;
17.
18.  component generic_toplayici
19.  Generic(
```

```
20.     n_bit : integer := 8
21.   );
22.   Port (
23.     in_giris_elde : in std_logic;
24.     in_giris_1 : in std_logic_vector(n_bit - 1 downto 0);
25.     in_giris_2 : in std_logic_vector(n_bit - 1 downto 0);
26.     out_cikis : out std_logic_vector(n_bit - 1 downto 0);
27.     out_cikis_elde : out std_logic
28.   );
29.   end component;
30.
31.end benim_paketim;
32.
33.package body benim_paketim is
34.   function  buyuk_bul(in_Kelime_1,  in_Kelime_2  :  t_Kelime)  return
   t_Kelime is
35.     variable v_buyuk : t_Kelime;
36.   begin
37.     v_buyuk := in_Kelime_1;
38.     if v_buyuk < in_Kelime_2 then
39.       v_buyuk := in_Kelime_2;
40.     end if;
41.     return v_buyuk;
42.   end buyuk_bul;
43.end benim_paketim;
```

benim_paketim isimli paket dosyasının kullanılacağı VHDL kodunun kütüphane bildirim kısmında bildirimi aşağıda verilen söz dizimi ile yapılmaktadır.

```
LIBRARY work;
USE work.benim_paketim.all;
```

Aşağıda **benim_paketim** paketi kullanılarak paket içerisinde kullanılan tüm tanımlamaların kullanıldığı **paket_kullanimi.vhd** VHDL kodu aşağıda verilmiştir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. library work;
5. use work.benim_paketim.all;
6.
```

```
7. entity paket_kullanimi is
8. end paket_kullanimi;
9.
10.architecture Behavioral of paket_kullanimi is
11.
12.  signal r_Buyuk : t_Kelime;
13.
14.begin
15.
16.  r_Buyuk <= buyuk_bul(r_Kelime_1, r_Kelime_2);
17.  generic_toplayici_4_bit : generic_toplayici
18.    Generic map( n_bit => VERI_UZUNLUGU )
19.    Port map (
20.      in_giris_elde => r_giris_1,
21.      in_giris_1 => in_giris_1,
22.      in_giris_2 => in_giris_2,
23.      out_cikis => open,
24.      out_cikis_elde => r_giris_2
25.  );
26.end Behavioral;
```