# 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.

## 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. **portmap** 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} ) ;

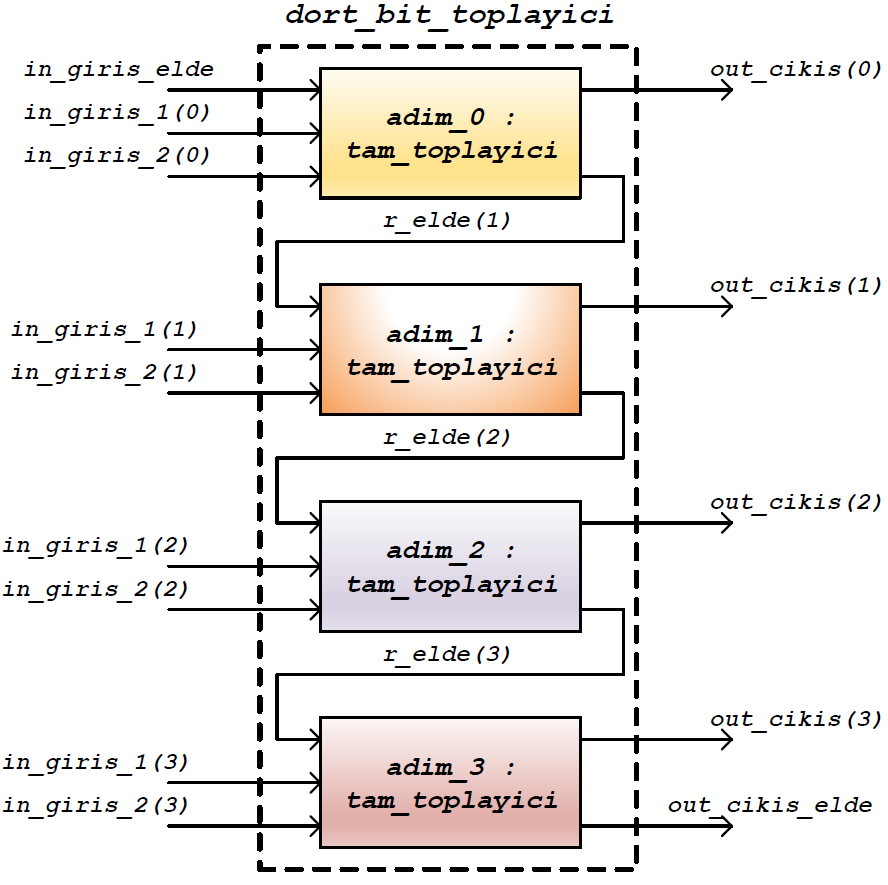
**end component**;

..

..

**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 ‑ 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. **entity** dort\_bit\_toplayici **is**
4. **Port**(
5. in\_giris\_elde : **in** std\_logic;
6. in\_giris\_1 : **in** std\_logic\_vector(**3** downto **0**);
7. in\_giris\_2 : **in** std\_logic\_vector(**3** downto **0**);
8. out\_cikis : **out** std\_logic\_vector(**3** downto **0**);
9. out\_cikis\_elde : **out** std\_logic
10. );
11. **end** dort\_bit\_toplayici;
12. **architecture** Behavioral **of** dort\_bit\_toplayici **is**
13. **component** tam\_toplayici
14. **Port** (
15. in\_giris\_elde : **in** std\_logic;
16. in\_giris\_1 : **in** std\_logic;
17. in\_giris\_2 : **in** std\_logic;
18. out\_cikis : **out** std\_logic;
19. out\_cikis\_elde : **out** std\_logic
20. );
21. **end component**;
22. **signal** r\_elde : std\_logic\_vector(**1** to **3**) ;
23. **begin**
24. adim\_0: tam\_toplayici **port map** (in\_giris\_elde, in\_giris\_1(0),
25. in\_giris\_2(0), out\_cikis(0), r\_elde(1) ) ;
26. adim\_1: tam\_toplayici **port map** (r\_elde(1), in\_giris\_1(1),
27. in\_giris\_2(1), out\_cikis(1), r\_elde(2) ) ;
28. adim\_2: tam\_toplayici **port map** (r\_elde(2), in\_giris\_1(2),
29. in\_giris\_2(2), out\_cikis(2), r\_elde(3) ) ;
30. adim\_3: tam\_toplayici **port map** (
31. in\_giris\_elde => r\_elde(3),
32. in\_giris\_1 => in\_giris\_1(3),
33. in\_giris\_2 => in\_giris\_2(3),
34. out\_cikis => out\_cikis(3),
35. out\_cikis\_elde => out\_cikis\_elde) ;
36. **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. **entity** tam\_toplayici i**s**
4. **Port** (
5. in\_giris\_elde : **in** std\_logic;
6. in\_giris\_1 : **in** std\_logic;
7. in\_giris\_2 : **in** std\_logic;
8. out\_cikis : **out** std\_logic;
9. out\_cikis\_elde : **out** std\_logic
10. );
11. **end** tam\_toplayici;
12. **architecture** Behavioral **of** tam\_toplayici **is**
13. **begin**
14. out\_cikis <= in\_giris\_elde **xor** in\_giris\_1 **xor** in\_giris\_2;
15. out\_cikis\_elde <= (in\_giris\_elde **and** in\_giris\_1) **or**
16. (in\_giris\_elde **and** in\_giris\_2) **or**
17. (in\_giris\_1 **and** in\_giris\_2);
18. **end** Behavioral;

## GENERIC Kullanarak ENTITY Tanımlama

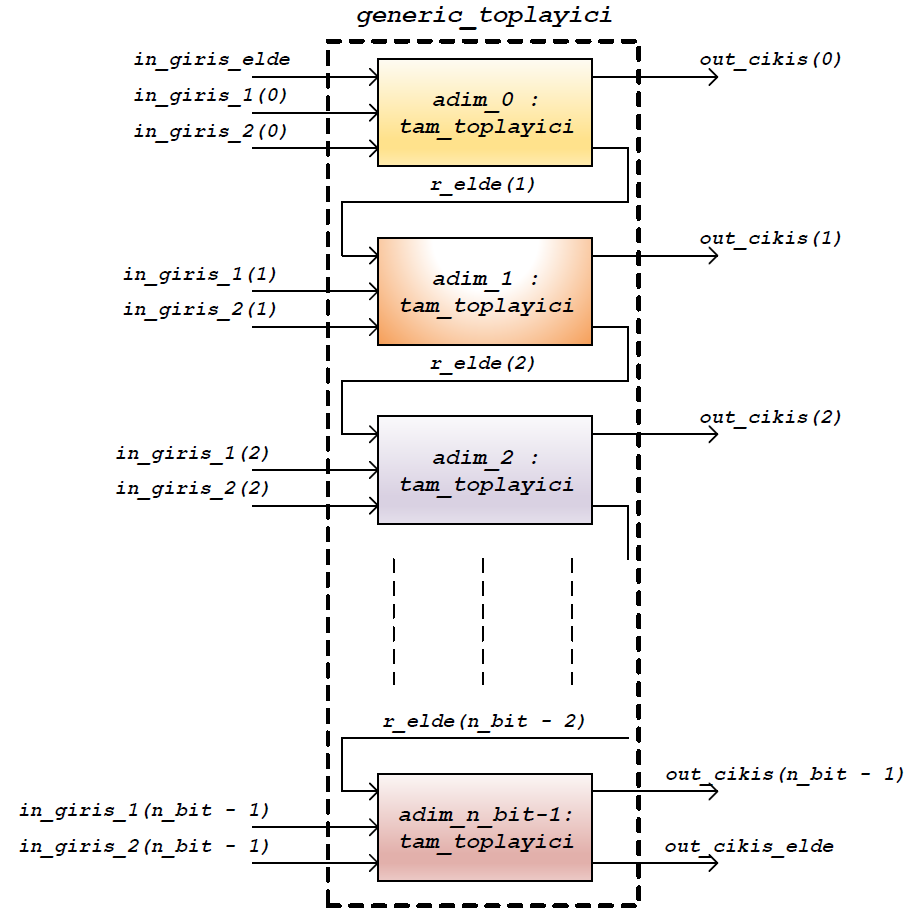
4 bitlik bir toplayıcı için art arda 4 adet tam toplayıcı devresinin birbirine bağlanması gerekmektedir. Eğer tasarlanan sistem 4 bitlik yerine artık 8 bitlik bir toplayıcı olarak kullanılacaksa bu sefer 8 adet tam toplayıcı devresinin birbirine bağlanması gerekmektedir. Bu sayı arttıkça kod içerisinde 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 ‑ 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. **entity** generic\_toplayici **is**
4. **Generic**(
5. n\_bit : integer := 8
6. );
7. **Port**(
8. in\_giris\_elde : **in** std\_logic;
9. in\_giris\_1 : **in** std\_logic\_vector(**n\_bit - 1** downto **0**);
10. in\_giris\_2 : **in** std\_logic\_vector(**n\_bit - 1** downto **0**);
11. out\_cikis : **out** std\_logic\_vector(**n\_bit - 1** downto **0**);
12. out\_cikis\_elde : **out** std\_logic
13. );
14. **end** generic\_toplayici;
15. **architecture** Behavioral **of** generic\_toplayici **is**
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. **signal** r\_elde : std\_logic\_vector(**0** to **n\_bit**) ;
26. **begin**
27. r\_elde(0) <= in\_giris\_elde;
28. for\_kontrol: **for** n\_i **in** **0** to **n\_bit - 1** **generate**
29. adim: tam\_toplayici **port map** (
30. in\_giris\_elde => r\_elde(n\_i),
31. in\_giris\_1 => in\_giris\_1(n\_i),
32. in\_giris\_2 => in\_giris\_2(n\_i),
33. out\_cikis => out\_cikis(n\_i),
34. out\_cikis\_elde => r\_elde(n\_i + 1)
35. );
36. **end generate** for\_kontrol;
37. out\_cikis\_elde <= r\_elde(n\_bit);
38. **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. **entity** tb\_toplayici **is**
4. **end** tb\_toplayici;
5. **architecture** Behavioral **of** tb\_toplayici **is**
6. **component** generic\_toplayici
7. **Generic**(
8. n\_bit : integer := 8
9. );
10. **Port** (
11. in\_giris\_elde : **in** std\_logic;
12. in\_giris\_1 : **in** std\_logic\_vector(**n\_bit - 1** downto **0**);
13. in\_giris\_2 : **in** std\_logic\_vector(**n\_bit - 1** downto **0**);
14. out\_cikis : **out** std\_logic\_vector(**n\_bit - 1** downto **0**);
15. out\_cikis\_elde : **out** std\_logic
16. );
17. **end component**;
18. **signal** in\_giris4\_1 : std\_logic\_vector(**3** downto **0**) := X"2";
19. **signal** in\_giris4\_2 : std\_logic\_vector(**3** downto **0**) := X"1";
20. **signal** out\_cikis4 : std\_logic\_vector(**3** downto **0**);
21. **signal** out\_cikis\_elde4 : std\_logic;
22. **signal** in\_giris8\_1 : std\_logic\_vector(**7** downto **0**) := X"12";
23. **signal** in\_giris8\_2 : std\_logic\_vector(**7** downto **0**) := X"22";
24. **signal** out\_cikis8 : std\_logic\_vector(**7** downto **0**);
25. **begin**
26. generic\_toplayici\_4\_bit : generic\_toplayici
27. **Generic map**( n\_bit => 4 )
28. **Port map** (
29. in\_giris\_elde => '0',
30. in\_giris\_1 => in\_giris4\_1,
31. in\_giris\_2 => in\_giris4\_2,
32. out\_cikis => out\_cikis4,
33. out\_cikis\_elde => out\_cikis\_elde4
34. );
35. generic\_toplayici\_8\_bit : generic\_toplayici
36. **Port map** (
37. in\_giris\_elde => '0',
38. in\_giris\_1 => in\_giris8\_1,
39. in\_giris\_2 => in\_giris8\_2,
40. out\_cikis => out\_cikis8,
41. out\_cikis\_elde => **open**
42. );
43. **end** Behavioral;

## 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. **variable** v\_elde : std\_logic;
3. **variable** v\_toplam : std\_logic\_vector(4 downto 0);
4. **begin**
5. v\_elde:= in\_giris\_elde;
6. v\_toplam := (others => '0');
7. **for** n\_i in 0 to 3 **loop**
8. v\_toplam(n\_i) := in\_giris\_1(n\_i) xor in\_giris\_2(n\_i) xor v\_elde;
9. v\_elde := (in\_giris\_1(n\_i) **and** in\_giris\_2(n\_i)) **or**
10. (in\_giris\_1(n\_i) **and** v\_elde) **or**
11. (in\_giris\_2(n\_i) **and** v\_elde);
12. **end loop**;
13. v\_toplam (**4**) := v\_elde;
14. **return** v\_toplam;
15. **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. **entity** function\_ornek **is**
4. **Port** (
5. in\_giris\_elde : **in** std\_logic;
6. in\_giris\_1 : **in** std\_logic\_vector(**3** downto **0**);
7. in\_giris\_2 : **in** std\_logic\_vector(**3** downto **0**);
8. out\_cikis : **out** std\_logic\_vector(**3** downto **0**);
9. out\_cikis\_elde : **out** std\_logic
10. );
11. **end** function\_ornek;
12. **architecture** Behavioral **of** function\_ornek **is**
13. **function** toplayici\_4\_bit (in\_giris\_1, in\_giris\_2 : std\_logic\_vector(**3** downto **0**); in\_giris\_elde: std\_logic)
14. **return** std\_logic\_vector **is**
15. **variable** v\_elde : std\_logic;
16. **variable** v\_toplam : std\_logic\_vector(**4** downto **0**);
17. **begin**
18. v\_elde := in\_giris\_elde;
19. v\_toplam := (**others** => '0');
20. **for** n\_i **in** **0** to **3** **loop**
21. v\_toplam(n\_i) := in\_giris\_1(n\_i) **xor** in\_giris\_2(n\_i) **xor** v\_elde;
22. v\_elde := (in\_giris\_1(n\_i) **and** in\_giris\_2(n\_i)) **or**
23. (in\_giris\_1(n\_i) **and** v\_elde) **or**
24. (in\_giris\_2(n\_i) **and** v\_elde);
25. **end** **loop**;
26. v\_toplam (4) := v\_elde;
27. **return** v\_toplam;
28. **end** toplayici\_4\_bit;
29. **signal** r\_toplama\_sonuc : std\_logic\_vector(**4** downto **0**) := (**others** => '0');
30. **begin**
31. r\_toplama\_sonuc <= toplayici\_4\_bit(in\_giris\_1, in\_giris\_2, in\_giris\_elde);
32. out\_cikis <= r\_toplama\_sonuc(**3** downto **0**);
33. out\_cikis\_elde <= r\_toplama\_sonuc(4);
34. **end** Behavioral;

## 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. **variable** v\_elde: std\_logic\_vector(**4** downto **0**);
8. **begin**
9. v\_elde(0) := in\_giris\_elde;
10. **for** n\_i **in** **0** to **3** **loop**
11. out\_cikis (n\_i) <= in\_giris\_1(n\_i) **xor** in\_giris\_2(i) **xor** v\_elde(n\_i);
12. v\_elde(n\_i + 1) := (in\_giris\_1(n\_i) **and** in\_giris\_2(n\_i)) **or**
13. (in\_giris\_2(n\_i) **and** v\_elde(n\_i)) **or**
14. (in\_giris\_2(n\_i) **and** v\_elde(n\_i));
15. **end loop**;
16. out\_cikis\_elde <= v\_elde(4);
17. **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. **entity** procedure\_ornek **is**
4. **Port** (
5. in\_giris\_elde : **in** std\_logic;
6. in\_giris\_1 : **in** std\_logic\_vector(**3** downto **0**);
7. in\_giris\_2 : **in** std\_logic\_vector(**3** downto **0**);
8. out\_cikis : **out** std\_logic\_vector(**3** downto **0**);
9. out\_cikis\_elde : **out** std\_logic
10. );
11. **end** procedure\_ornek;
12. **architecture** Behavioral **of** procedure\_ornek **is**
13. **procedure** toplayici\_4\_bit(
14. in\_giris\_elde : **in** std\_logic;
15. in\_giris\_1 : **in** std\_logic\_vector(**3** downto **0**);
16. in\_giris\_2 : **in** std\_logic\_vector(**3** downto **0**);
17. out\_cikis : **out** std\_logic\_vector(**3** downto **0**);
18. out\_cikis\_elde : **out** std\_logic ) **is**
19. **variable** v\_elde: std\_logic\_vector(**4** downto **0**);
20. **begin**
21. v\_elde(0) := in\_giris\_elde;
22. **for** n\_i **in** **0** to **3** **loop**
23. out\_cikis (n\_i) := in\_giris\_1(n\_i) **xor** in\_giris\_2(n\_i) **xor** v\_elde(n\_i);
24. 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));
25. **end** **loop**;
26. out\_cikis\_elde := v\_elde(4);
27. **end** toplayici\_4\_bit;
28. **begin**
29. **process**(in\_giris\_elde, in\_giris\_1, in\_giris\_2)
30. **variable** v\_cikis : std\_logic\_vector(**3** downto **0**);
31. **variable** v\_cikis\_elde : std\_logic;
32. **begin**
33. toplayici\_4\_bit(in\_giris\_elde, in\_giris\_1, in\_giris\_2, v\_cikis, v\_cikis\_elde );
34. out\_cikis <= v\_cikis;
35. out\_cikis\_elde <= v\_cikis\_elde;
36. **end** **process**;
37. **end** Behavioral;

## 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. **package** benim\_paketim **is**
4. **signal** r\_giris\_1 : std\_logic := '0';
5. **signal** r\_giris\_2 : std\_logic := '0';
6. **constant** VERI\_UZUNLUGU : integer := 6;
7. **signal** in\_giris\_1 : std\_logic\_vector(**VERI\_UZUNLUGU - 1** downto **0**) := (**others** => '0');
8. **signal** in\_giris\_2 : std\_logic\_vector(**VERI\_UZUNLUGU - 1** downto **0**) := (**others** => '0');
9. **type** t\_Kelime **is** **array** (**9** downto **0**) **of** std\_logic;
10. **signal** r\_Kelime\_1 : t\_kelime := (**others** => '0');
11. **signal** r\_Kelime\_2 : t\_kelime := (**others** => '0');
12. **function** buyuk\_bul(in\_Kelime\_1, in\_Kelime\_2 : t\_Kelime) **return** t\_Kelime;
13. **component** generic\_toplayici
14. **Generic**(
15. n\_bit : integer := 8
16. );
17. **Port** (
18. in\_giris\_elde : **in** std\_logic;
19. in\_giris\_1 : **in** std\_logic\_vector(**n\_bit - 1** downto **0**);
20. in\_giris\_2 : **in** std\_logic\_vector(**n\_bit - 1** downto **0**);
21. out\_cikis : **out** std\_logic\_vector(**n\_bit - 1** downto **0**);
22. out\_cikis\_elde : **out** std\_logic
23. );
24. **end** **component**;
25. **end** benim\_paketim;
26. **package** **body** benim\_paketim **is**
27. **function** buyuk\_bul(in\_Kelime\_1, in\_Kelime\_2 : t\_Kelime) **return** t\_Kelime **is**
28. **variable** v\_buyuk : t\_Kelime;
29. **begin**
30. v\_buyuk := in\_Kelime\_1;
31. **if** v\_buyuk < in\_Kelime\_2 **then**
32. v\_buyuk := in\_Kelime\_2;
33. **end** **if**;
34. **return** v\_buyuk;
35. **end** buyuk\_bul;
36. **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. **library** work;
4. **use** work.benim\_paketim.all;
5. **entity** paket\_kullanimi **is**
6. **end** paket\_kullanimi;
7. **architecture** Behavioral **of** paket\_kullanimi **is**
9. **signal** r\_Buyuk : t\_Kelime;
10. **begin**
11. r\_Buyuk <= buyuk\_bul(r\_Kelime\_1, r\_Kelime\_2);
12. generic\_toplayici\_4\_bit : generic\_toplayici
13. **Generic** map**(** n\_bit => VERI\_UZUNLUGU )
14. **Port** **map** (
15. in\_giris\_elde => r\_giris\_1,
16. in\_giris\_1 => in\_giris\_1,
17. in\_giris\_2 => in\_giris\_2,
18. out\_cikis => **open**,
19. out\_cikis\_elde => r\_giris\_2
20. );
21. **end** Behavioral;