# VHDL Operatörleri ve Nitelikleri (Attributes)

VHDL dilinde tasarım yaparken kullanılabilecek 3 tür operatör mevcuttur. Bunlar sırasıyla:

* Mantıksal (Booelean) Operatörler
* Aritmetik Operatörler
* İlişkisel Operatörler’dir.

Yukarıda verilen liste operatörlerin öncelik sırasına göre sıralanmış olup, aynı tür operatörlerin kendi aralarında bir öncelik sırası yoktur.

VHDL dilinde ayrıca kod yazmayı kolaylaştıran ve tekrar kullanılabilirliği arttıran nitelik (**attributes**) tanımlamaları da mevcuttur. VHDL dilinde ön tanımlı olarak gelen nitelik (**attributes**) tanımlamaları olduğu gibi, kullanıcılar da kendi tanımlamalarını oluşturabilmektedir.

Bu bölümdeki başlıklar operatörlerin öncelik sırasına göre düzenlenmiş olup operatör tanımlamalarından sonra ise nitelik (**attributes**) tanımlamalarından bahsedilmiştir. Nitelik tanımlamaları özellikle genelleştirilebilir (**generic design**) tasarım yapmak adına oldukça faydalı araçlardır.

## Mantıksal Operatörler

VHDL dilinde kullanılan mantıksal operatörler aşağıda listelenmiştir:

* **and:** Mantıksal **VE** işlemi

out\_cikis <= in\_giris\_1 **and** in\_giris\_2;

Yukarıda verilen tanımlamada **out\_cikis** değerine **in\_giris\_1** ve **in\_giris\_2** değerlerinin mantıksal **VE** işleminin sonucu atanmaktadır.

* **or:** Mantıksal **VEYA** işlemi

out\_cikis <= in\_giris\_1 **or** in\_giris\_2;

Yukadrıda verilen tanımlamada **out\_cikis** değerine **in\_giris\_1** ve **in\_giris\_2** değerlerinin mantıksal **VEYA** işleminin sonucu atanmaktadır.

* **nand:** Mantıksal **VE DEĞİL** işlemi

out\_cikis <= in\_giris\_1 **nand** in\_giris\_2;

Yukarıda verilen tanımlamada **out\_cikis** değerine **in\_giris\_1** ve **in\_giris\_2** değerlerinin mantıksal **VE DEĞİL** işleminin sonucu atanmaktadır.

* **nor:** Mantıksal **VEYA DEĞİL** işlemi

out\_cikis <= in\_giris\_1 **nor** in\_giris\_2;

Yukarıda verilen tanımlamada **out\_cikis** değerine **in\_giris\_1** ve **in\_giris\_2** değerlerinin mantıksal **VEYA DEĞİL** işleminin sonucu atanmaktadır.

* **xor:** Mantıksal **ÖZEL VEYA** işlemi

out\_cikis <= in\_giris\_1 **xor** in\_giris\_2;

Yukarıda verilen tanımlamada **out\_cikis** değerine **in\_giris\_1** ve **in\_giris\_2** değerlerinin mantıksal **ÖZEL VEYA** işleminin sonucu atanmaktadır

* **xnor:** Mantıksal **ÖZEL VEYA DEĞİL** işlemi

out\_cikis <= in\_giris\_1 **xnor** in\_giris\_2;

Yukarıda verilen tanımlamada **out\_cikis** değerine **in\_giris\_1** ve **in\_giris\_2** değerlerinin lojik **ÖZEL VEYA DEĞİL** işleminin sonucu atanmaktadır.

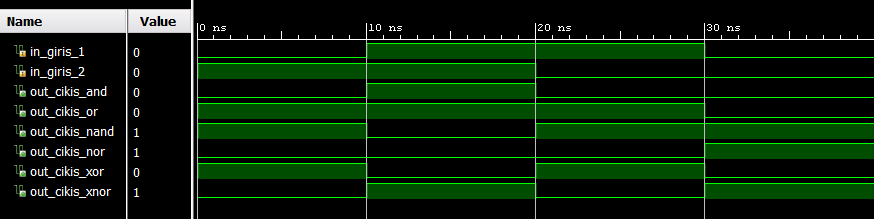
**Örnek 4.1 :** Mantıksal operatörlerin kullanıldığı **mantiksal\_operatorler.vhd** VHDL kodu aşağıda verilmiştir. Verilen koda ile tasarlanan devreye ait bilgiler aşağıdaki gibidir:

* **mantiksal\_operatorler** varlığı **in\_giris\_1** ve **in\_giris\_2** giriş portlarına sahiptir.
* **out\_cikis\_and** çıkış portuna **in\_giris\_1** ve **in\_giris\_2** giriş port değerlerinin **and** işlemi sonucu atanmaktadır.
* **out\_cikis\_or** çıkış portuna **in\_giris\_1** ve **in\_giris\_2** giriş port değerlerinin **or** işlemi sonucu atanmaktadır.
* **out\_cikis\_nand** çıkış portuna **in\_giris\_1** ve **in\_giris\_2** giriş port değerlerinin **nand** işlemi sonucu atanmaktadır.
* **out\_cikis\_nor** çıkış portuna **in\_giris\_1** ve **in\_giris\_2** giriş port değerlerinin **nor** işlemi sonucu atanmaktadır.
* **out\_cikis\_xor** çıkış portuna **in\_giris\_1** ve **in\_giris\_2** giriş port değerlerinin **xor** işlemi sonucu atanmaktadır.
* **out\_cikis\_xnor** çıkış portuna **in\_giris\_1** ve **in\_giris\_2** giriş port değerlerinin **xnor** işlemi sonucu atanmaktadır.

Şekil 4‑1’de **mantiksal\_operatorler** varlığının ilgili çıkışlara ilişkin benzetim çıktısı gösterilmiştir. Verilen benzetim çıktısına göre **in\_giris\_1** giriş portunun **‘0’** ve **in\_giris\_2** giriş portunun **‘1’** değerleri için;

* **out\_cikis\_and** çıkış portu değeri (**0** **and** **1**) => **0**,
* **out\_cikis\_or** çıkış portu değeri (**0** **or** **1**) => **1**,
* **out\_cikis\_nand** çıkış portu değeri (**0** **nand 1**) => **1**,
* **out\_cikis\_nor** çıkış portu değeri (**0** **nor** **1**) => **0**,
* **out\_cikis\_xor** çıkış portu değeri (**0** **xor** **1**) => **1**,
* **out\_cikis\_xnor** çıkış portu değeri (**0** **xnor** **1**) => **0** olmaktadır.

1. **library** IEEE;
2. **use** IEEE.STD\_LOGIC\_1164.ALL;
3. **entity** mantiksal\_operatorler **is**
4. **Port** (
5. in\_giris\_1 : **in** std\_logic;
6. in\_giris\_2 : **in** std\_logic;
7. out\_cikis\_and : **out** std\_logic;
8. out\_cikis\_or : **out** std\_logic;
9. out\_cikis\_nand : **out** std\_logic;
10. out\_cikis\_nor : **out** std\_logic;
11. out\_cikis\_xor : **out** std\_logic;
12. out\_cikis\_xnor : **out** std\_logic
13. );
14. **end** mantiksal\_operatorler;
15. **architecture** Behavioral **of** mantiksal\_operatorler **is**
16. **begin**
17. out\_cikis\_and <= in\_giris\_1 **and** in\_giris\_2;
18. out\_cikis\_or <= in\_giris\_1 **or** in\_giris\_2;
19. out\_cikis\_nand <= in\_giris\_1 **nand** in\_giris\_2;
20. out\_cikis\_nor <= in\_giris\_1 **nor** in\_giris\_2;
21. out\_cikis\_xor <= in\_giris\_1 **xor** in\_giris\_2;
22. out\_cikis\_xnor <= in\_giris\_1 **xnor** in\_giris\_2;
23. **end** Behavioral;

****

Şekil ‑ mantiksal\_operatorler varlığının ilgili çıkışlara ilişkin benzetim çıktısı

## İlişkisel Operatörler

VHDL dilinde kullanılan ilişkisel operatörler aşağıda listelenmiştir:

* **=** : eşittir.

**if** A = B **then**

out\_cikis<= in\_giris\_1;

**else**

out\_cikis<= in\_giris\_2;

**end if**;

Yukarıda verilen tanımlamada eğer **A** ve **B** değerleri birbirine eşit ise **out\_cikis** değerine **in\_giris\_1**, aksi durumda **in\_giris\_2** değeri atanmaktadır.

* **/=**: eşit değil

**if** A /= B **then**

out\_cikis<= in\_giris\_1;

**else**

out\_cikis<= in\_giris\_2;

**end** **if**;

Yukarıda verilen tanımlamada eğer **A** ve **B** değerleri birbirine eşit değil ise **out\_cikis** değerine **in\_giris\_1**, aksi durumda **in\_giris\_2** değeri atanmaktadır.

* **<** : küçük

**if** A < B **then**

out\_cikis<= in\_giris\_1;

**else**

out\_cikis<= in\_giris\_2;

**end** **if**;

Yukarıda verilen tanımlamada eğer **A** değeri **B** değerinden küçük ise **out\_cikis** değerine **in\_giris\_1**, aksi durumda **in\_giris\_2** değeri atanmaktadır.

* **<=** : küçük eşit

**if** A <= B **then**

out\_cikis<= in\_giris\_1;

**else**

out\_cikis<= in\_giris\_2;

**end** **if**;

Yukarıda verilen tanımlamada eğer **A** değeri **B** değerinden küçük ve eşit ise **out\_cikis** değerine **in\_giris\_1**, aksi durumda **in\_giris\_2** değeri atanmaktadır.

* **>** : büyük

**if** A > B **then**

out\_cikis<= in\_giris\_1;

**else**

out\_cikis<= in\_giris\_2;

**end** **if**;

Yukarıda verilen tanımlamada eğer **A** değeri **B** değerinden büyük ise **out\_cikis** değerine **in\_giris\_1**, aksi durumda **in\_giris\_2** değeri atanmaktadır.

* **>=** : büyük eşit

**if** A >= B **then**

out\_cikis<= in\_giris\_1;

**else**

out\_cikis<= in\_giris\_2;

**end** **if**;

Yukarıda verilen tanımlamada eğer **A** değeri **B** değerinden büyük ve eşit ise **out\_cikis** değerine **in\_giris\_1**, aksi durumda **in\_giris\_2** değeri atanmaktadır.

## Toplama, Çıkarma ve Ekleme Operatörleri

VHDL dilinde kullanılan toplama, çıkarma ve ekleme operatörleri aşağıda listelenmiştir:

* **+** : toplama

out\_cikis <= in\_giris\_1 + in\_giris\_2;

Yukarıda verilen tanımlamada çıkış değerine **in\_giris\_1** ve **in\_giris\_2** değerlerinin toplamı atanmaktadır. Örneğin **in\_giris\_1** değeri"**1010**" ve **in\_giris\_2** değeri **"0101"** olsun. Bu durumda **out\_cikis** değerine **"1111"** olmaktadır.

* **-** : çıkarma

out\_cikis <= in\_giris\_1 - in\_giris\_2;

Yukarıda verilen tanımlamada çıkış değerine **in\_giris\_1** değerinden **in\_giris\_2** değerinin farkı atanmaktadır. Örneğin **in\_giris\_1** değeri **"1010"** ve **in\_giris\_2** değeri **"0101"** olsun. Bu durumda **out\_cikis** değerine **"0101"** olmaktadır.

* **&** : ekleme

out\_cikis <= in\_giris\_1 & in\_giris\_2;

Yukarıda verilen tanımlamada çıkış değerine **in\_giris\_1** değerine **in\_giris\_2** değeri eklenerek atanmaktadır. Örneğin **in\_giris\_1** değeri **"1010"** ve **in\_giris\_2** değeri **"0101"** olsun. Bu durumda **out\_cikis** değerine **"10100101"** olmaktadır.

Bu operatör aynı zamanda kaydırma (shifting) işlemleri için de kullanılmaktadır. Bu sayede sağa ya da sola kaydırma yapılabilir. Bunun için sınır değerlerin kontrolü gerekmektedir. Örneğin **gelen\_veri** değeri 8 bit uzunluğunda “**10100101**” başlangıç değerinde verilmiş olsun. Verilen veriyi sola kaydırmak için yazmamız gereken kod aşağıda verilmiştir:

**signal** gelen\_veri : std\_logic\_vector(**7** downto **0**):=”10100101”;

..

..

gelen\_veri <= gelen\_veri(**6** downto **0**) & ‘0’;

Eğer kaydırma işlemini diğer yöne, sağa yaptırmak isteseydik yazmamız gereken kod aşağıdaki gibi olacaktır:

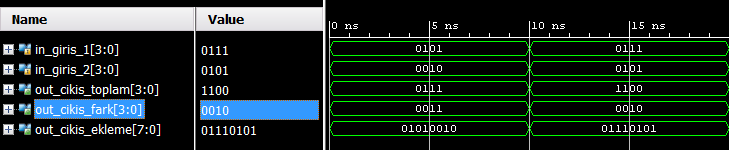
gelen\_veri <= ‘0’ & gelen\_veri(**7** downto **1**);

**Örnek 4.2 :** Toplama operatörlerinin kullanıldığı **toplama\_operatorleri.vhd** VHDL kodu aşağıda gösterilmiştir. **toplama\_operatorleri** varlığı 4 bitlik **in\_giris\_1** ve **in\_giris\_2** giriş portlarına sahiptir.

* 4 bitlik **out\_cikis\_toplam** çıkış portuna **in\_giris\_1** ve **in\_giris\_2** giriş port değerlerinin toplama işlemi sonucu atanmaktadır.
* 4 bitlik **out\_cikis\_fark** çıkış portuna **in\_giris\_1** ve **in\_giris\_2** giriş port değerlerinin çıkarma işlemi sonucu atanmaktadır.
* 8 bitlik **out\_cikis\_ekleme** çıkış portuna **in\_giris\_1** ve **in\_giris\_2** giriş port değerleri eklenerek atanmaktadır.

Şekil 4‑2’de **toplama\_operatorleri** varlığının ilgili çıkışlara ilişkin benzetim çıktısı gösterilmiştir. Şekil 4‑2’de **in\_giris\_1** giriş portunun **“0101”** ve **in\_giris\_2** giriş portunun **“0010”** değeri için; **out\_cikis\_toplam** çıkış portu değeri **“0111”, out\_cikis\_fark** çıkış portu değeri **“0011”** ve . **out\_cikis\_ekleme** çıkış portu değeri **“01010010”** olmaktadır.

1. **library** IEEE;
2. **use** IEEE.STD\_LOGIC\_1164.ALL;
3. **use** IEEE.STD\_LOGIC\_SIGNED.ALL;
4. **entity** toplama\_operatorleri **is**
5. **Port** (
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\_toplam : **out** std\_logic\_vector(**3** downto **0**);
9. out\_cikis\_fark : **out** std\_logic\_vector(**3** downto **0**);
10. out\_cikis\_ekleme : **out** std\_logic\_vector(**7** downto **0**)
11. );
12. **end** toplama\_operatorleri;
13. **architecture** Behavioral **of** toplama\_operatorleri **is**
14. **begin**
15. **process**(in\_giris\_1, in\_giris\_2)
16. **begin**
17. out\_cikis\_toplam <= in\_giris\_1 + in\_giris\_2;
18. out\_cikis\_fark <= in\_giris\_1 - in\_giris\_2;
19. out\_cikis\_ekleme <= in\_giris\_1 & in\_giris\_2;
20. **end process**;
21. **end** Behavioral;



Şekil ‑ toplama\_operatorleri varlığının ilgili çıkışlara ilişkin benzetim çıktısı

## Çarpma, Bölme, Mod ve Artan Operatörleri

VHDL dilinde kullanılan çarpma, bölme, mod ve artan operatörleri aşağıda listelenmiştir:

* **\*** : çarpma

out\_cikis<= in\_giris\_1 \* in\_giris\_2;

Yukarıda verilen tanımlamada çıkış değerine **in\_giris\_1** değerine **in\_giris\_2** değeri ile çarpılarak atanmaktadır. Örneğin **in\_giris\_1** değeri "**1010**" ve **in\_giris\_2** değeri "**0101**" olsun. Bu durumda **out\_cikis** değerine **"00110010"** atanmaktadır.

* **/** : bölme

out\_cikis <= in\_giris\_1 / in\_giris\_2;

Yukarıda verilen tanımlamada çıkış değerine **in\_giris\_1** değerine **in\_giris\_2** değerine bölünerek atanmaktadır. Örneğin **in\_giris\_1** değeri **19** ve **in\_giris\_2** değeri **4** olsun. Bu durumda **out\_cikis** değerine **4** atanmaktadır.

VHDL dilinde var olan bölme operatörü sadece 2’nin kuvvetleri şeklinde ifade edilebilen sayılar üzerinde sentezlenebilir sonuç üretmektedir.

* **mod** : mod alma

out\_cikis <= in\_giris\_1 **mod** in\_giris\_2;

Yukarıda verilen tanımlamada çıkış değerine **in\_giris\_1** değerinin **in\_giris\_2** değerine göre modu atanmaktadır. Örneğin **in\_giris\_1** değeri **19** ve **in\_giris\_2** değeri **4** olsun. Bu durumda **out\_cikis** değerine **19 mod 4 = 3** atanmaktadır. Eğer **in\_giris\_1** değeri **-19** olsaydı **out\_cikis** değerine -**19 mod 4 = 1** atanacaktır.

VHDL dilinde var olan **mod** operatörü sadece 2’nin kuvvetleri şeklinde ifade edilebilen sayılar üzerinde sentezlenebilir sonuç üretmektedir.

* **rem** : artan

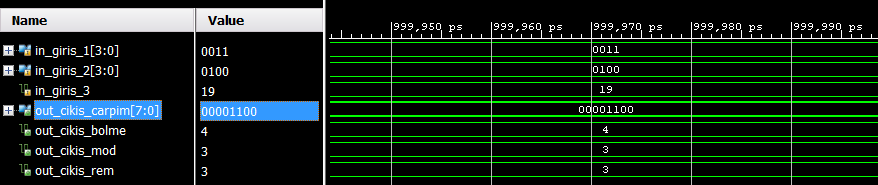
out\_cikis <= in\_giris\_1 **rem** in\_giris\_2;

Yukarıda verilen tanımlamada çıkış değerine, **in\_giris\_1** değerinin **in\_giris\_2** değerine göre artan değeri atanmaktadır. Örneğin **in\_giris\_1** değeri **19** ve **in\_giris\_2** değeri **4** olsun. Bu durumda **out\_cikis** değerine 19 **rem 4 = 3** atanmaktadır. Eğer **in\_giris\_1** değeri **-19** olsaydı **out\_cikis** değerine -**19 mod 4 = -3** atanacaktır.

VHDL dilinde var olan **rem** operatörü sadece 2’nin kuvvetleri şeklinde ifade edilebilen sayılar üzerinde sentezlenebilir sonuç üretmektedir.

**Örnek 4.3 :** Çarpım operatörlerinin kullanıldığı **carpim\_operatorleri.vhd** VHDL kodu aşağıda gösterilmiştir. Şekil 4‑3’te **carpim\_operatorleri** varlığının ilgili çıkışlara ilişkin benzetim çıktısı gösterilmiştir. Örnek içerisinde 25., 26. ve 27. satırlarda kullanılan **conv\_integer** fonksiyonu **std\_logic\_vector** türünde sinyali **integer** tipine tür dönüşüm işlemini gerçekleştirmektedir. Detaylı olarak Bölüm 5 içerisinde anlatılacaktır.

1. **library** IEEE;
2. **use** IEEE.STD\_LOGIC\_1164.ALL;
3. **use** IEEE.STD\_LOGIC\_SIGNED.ALL;
4. **entity** carpim\_operatorleri **is**
5. **Port** (
6. in\_giris\_1 : **in** std\_logic\_vector(**3** downto **0**) := “0011”;
7. in\_giris\_2 : **in** std\_logic\_vector(**3** downto **0**) := “0100”;
8. in\_giris\_3 : **in** integer := 19;
9. out\_cikis\_carpim : **out** std\_logic\_vector(**7** downto **0**);
10. out\_cikis\_bolme : **out** integer;
11. out\_cikis\_mod : **out** integer;
12. out\_cikis\_rem : **out** integer
13. );
14. **end** carpim\_operatorleri;
15. **architecture** Behavioral **of** carpim\_operatorleri **is**
16. **begin**
17. **process**(in\_giris\_1, in\_giris\_2, in\_giris\_3)
18. **begin**
19. out\_cikis\_carpim <= in\_giris\_1 \* in\_giris\_2;
20. out\_cikis\_bolme <= in\_giris\_3 / conv\_integer(in\_giris\_2);
21. out\_cikis\_mod <= in\_giris\_3 **mod** conv\_integer(in\_giris\_2);
22. out\_cikis\_rem <= in\_giris\_3 **rem** conv\_integer(in\_giris\_2);
23. **end process**;
24. **end** Behavioral;



Şekil ‑ carpim\_operatorleri varlığının ilgili çıkışlara ilişkin benzetim çıktısı

## Diğer Operatörler

* **\*\*** : Üs alma

out\_cikis <= in\_giris\_1 \*\* 3;

Yukarıda verilen tanımlamada çıkış değerine, **in\_giris\_1** değerinin 3. dereceden kuvveti atanmaktadır. Örneğin **in\_giris\_1** değeri "**010**" olsun. Bu durumda **out\_cikis** değerine "**000001000**" atanmaktadır.

* **abs :** Mutlak değer

out\_cikis <= **abs**(in\_giris\_1);

Yukarıda verilen tanımlamada çıkış değerine, **in\_giris\_1** değerinin mutlak değeri atanmaktadır. Örneğin **in\_giris\_1** değeri **"010**" olsun. Bu durumda **out\_cikis** değerine "**010**" atanmaktadır. **in\_giris\_1** değeri **"101**" olduğu durumda ise **out\_cikis** değerine "**011**" atanmaktadır.

* **not** : Tersi

out\_cikis<= **not**(in\_giris\_1);

Yukarıda verilen tanımlamada çıkış değerine, **in\_giris\_1** değerinin tersi atanmaktadır. Örneğin **in\_giris\_1** değeri"**010**" olsun. Bu durumda çıkış **out\_cikis** değerine "**101**" atanmaktadır.

## Operatörlerin Kullanımı

Aşağıda verilen iki tanımlamayı inceleyelim:

1. out\_cikis <= in\_giris\_1 **and** in\_giris\_2 **and** in\_giris\_3 **and** in\_giris\_4;
2. out\_cikis <= in\_giris\_1 **\*** in\_giris\_2 **\*** in\_giris\_3 **+** in\_giris\_4;

Yukarıda verilen iki tanımlama aynı çıkış değerini üretmesine rağmen aynı anlama gelmemektedir. 1. ifade de **and** operatörünün **or** operatörüne üstünlüğü yoktur. 2. ifade de ise **\*** operatörünün **+** operatörüne göre üstünlüğü vardır. Bu nedenle tanımlamalar kullanılırken hata yapılma olasılığı yüksektir. Kullanıcının gerçekleştirmek istediği işleme göre parantez kullanımı ile hata olasılığını azaltabilir.

Aşağıda verilen iki ifade parantez kullanımı ile farklı tanımlama yapılmıştır. Örneğin **in\_giris\_1= '0'**, **in\_giris\_2 = '1', in\_giris\_3 = '0'** ve **in\_giris\_4 = '1'** olması durumunda 1. **ifade (0 and 1 and 0) or 1 = 1** çıkışını verirken 2. ifade **(0 and 1) and (0 or 1) = 0** çıkışını vermektedir.

1. out\_cikis <= (in\_giris\_1 **and** in\_giris\_2 **and** in\_giris\_3) **or** in\_giris\_4;
2. out\_cikis <= (in\_giris\_1 **and** in\_giris\_2) **and** (in\_giris\_3 **or** in\_giris\_4);

## Nitelikler (Attributes)

Nitelikler önceden tanımlı ve kullanıcı tanımlı olmak üzere ikiye ayrılmaktadır.

### Önceden Tanımlı Nitelikler (Pre-defined Attributes)

Aşağıda önceden tanımlı nitelikler gösterilmiştir:

**Veri Nitelikleri**

* **LOW:** Dizinin alt indisini döndürür.
* **HIGH:** Dizinin üst indisini döndürür.
* **LEFT:** Dizinin en soldaki indisini döndürür.
* **RIGHT:** Dizinin en sağdaki indisini döndürür.
* **LENGTH:** Vektör uzunluğunu döndürür.
* **RANGE:** Vektör aralığını döndürür.
* **REVERSE\_RANGE:** Vektör aralığının tersini döndürür.

Aşağıda tanımlanan **ornek\_sinyal** sinyali kullanarak veri niteliklerinin kullanımına ilişkin örnekler verilmiştir.

**signal** ornek\_sinyal : std\_logic\_vector(**3** downto **0**) := “0101”;

* A <= ornek\_sinyal(ornek\_sinyal’low) ;

A <= ornek\_sinyal(0) ;

* B <= ornek\_sinyal(ornek\_sinyal’high) ;

B <= ornek\_sinyal(3) ;

* C <= ornek\_sinyal(ornek\_sinyal’left) ;

C <= ornek\_sinyal(3) ;

* D <= ornek\_sinyal(ornek\_sinyal’right) ;

D <= ornek\_sinyal(0) ;

* E <= ornek\_sinyal’range ;

**for** n\_i **in** E **loop**

**for** n\_i **in** **3** downto **0** **loop**

* F <= ornek\_sinyal’reverse\_range ;

**for** n\_i **in** F **loop**

**for** n\_i **in** **0** to **3** **loop**

Sinyal tanımlama işleminde listeleme tipi tür kullanılmış ise aşağıda verilen nitelikler kullanılır.

* **VAL :** Tip değerinin pozisyonunu döndürür.

**type** t\_Kontrol **is** (BOSTA, BASLA, OKU, YAZ, TAMAM);

**signal** pozisyon : integer := t\_Kontrol’pos(Yaz);

**signal** pozisyon : integer := 3;

* **POS :** Tanımlı pozisyondaki tip değerinin döndürür.

**type** t\_Kontrol **is** (BOSTA, BASLA, OKU, YAZ, TAMAM);

**signal**  deger: t\_Kontrol := t\_Kontrol’val(3);

**signal**  deger : t\_Kontrol := Yaz;

* **LEFTOF :** Tip değerinin solundaki değeri döndürür.

**type** t\_Kontrol **is** (BOSTA, BASLA, OKU, YAZ, TAMAM);

**signal**  deger: t\_Kontrol := t\_Kontrol’leftof(Yaz);

**signal**  deger : t\_Kontrol := Oku;

* **RIGHTOF :** Tip değerinin sağındaki değeri döndürür.

**type** t\_Kontrol **is** (BOSTA, BASLA, OKU, YAZ, TAMAM);

**signal** deger: t\_Kontrol := t\_Kontrol’rightof(Yaz);

**signal** deger : t\_Kontrol := TAMAM;

**Sinyal Nitelikleri**

* **EVENT:** Olay gerçekleştiğinde doğru değeri döndürür.

**if** (in\_clk'event **and** in\_clk='1') **then**

* **STABLE:** Olay gerçekleşmediğinde doğru değeri döndürür.

**if** (**not** in\_clk'stable **and** in\_clk='1') **then**

* **ACTIVE:** Herhangi bir hareket meydana geldiğinde doğru değeri döndürür.
* **QUIET(süre) :** Belirtilen zaman diliminde hiçbir hareket meydana gelmediğinde doğru değeri döndürür.
* **LAST\_VALUE:** Son olaydan önceki değeri döndürür.

### Kullanıcı Tanımlı Nitelikler (User Defined Attributes)

Aşağıda kullanıcı tanımlı nitelik söz dizimi verilmiştir.

**attribute** nitelik\_ismi : nitelik\_tipi;

**attribute** nitelik\_ismi **of** hedef\_isim : nesne **is** deger;

Aşağıda kullanıcı tanımlı niteliklerde kullanılabilecek **nitelik\_tipi**, **nesne** ve **deger** örnekleri gösterilmiştir.

**nitelik\_tipi** : bit, integer, std\_logic\_vector(tüm veri tipleri)

**nesne** : **type**, **signal**, **procedure**, v.b

**deger** : ‘1’, 254, v.b

Aşağıda tanımlanan **kosul\_nitelik** niteliği **boolean** nitelik tipindedir ve **in\_clk** sinyalinin **true** olduğu durumlarda doğru değeri döndürür.

**attribute** kosul\_nitelik : boolean;

**attribute** kosul\_nitelik **of** in\_clk: **signal is** “true”;