4. 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.

4.1. 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:

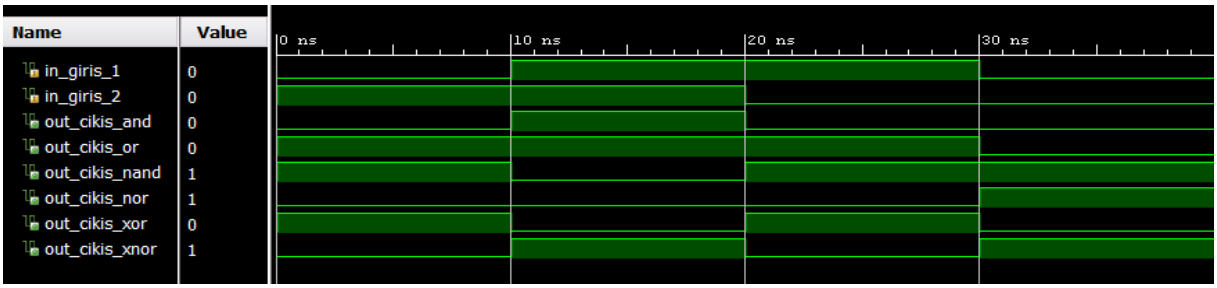
- **mantiksal_operatorler** varlığı **in_giris_1** ve **in_giris_2** giriş portlarına sahiptir.
- **out_cikis_and** çıkış portuna **in_giris_1** ve **in_giris_2** giriş port değerlerinin **and** işlemi sonucu atanmaktadır.
- **out_cikis_or** çıkış portuna **in_giris_1** ve **in_giris_2** giriş port değerlerinin **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.
4. entity mantiksal_operatorler is
5.   Port (
6.     in_giris_1 : in std_logic;
7.     in_giris_2 : in std_logic;
8.     out_cikis_and : out std_logic;
9.     out_cikis_or : out std_logic;
10.    out_cikis_nand : out std_logic;
```

```
11.     out_cikis_nor : out std_logic;
12.     out_cikis_xor : out std_logic;
13.     out_cikis_xnor : out std_logic
14.   );
15.end mantiksal_operatorler;
16.
17.architecture Behavioral of mantiksal_operatorler is
18.
19.begin
20.
21.   out_cikis_and   <= in_giris_1 and in_giris_2;
22.   out_cikis_or    <= in_giris_1 or in_giris_2;
23.   out_cikis_nand  <= in_giris_1 nand in_giris_2;
24.   out_cikis_nor   <= in_giris_1 nor in_giris_2;
25.   out_cikis_xor   <= in_giris_1 xor in_giris_2;
26.   out_cikis_xnor  <= in_giris_1 xnor in_giris_2;
27.
28.end Behavioral;
```



Şekil 4-1 mantiksal_operatorler varlığının ilgili çıkışlara ilişkin benzetim çıktısı

4.2. 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.

4.3. 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);
```

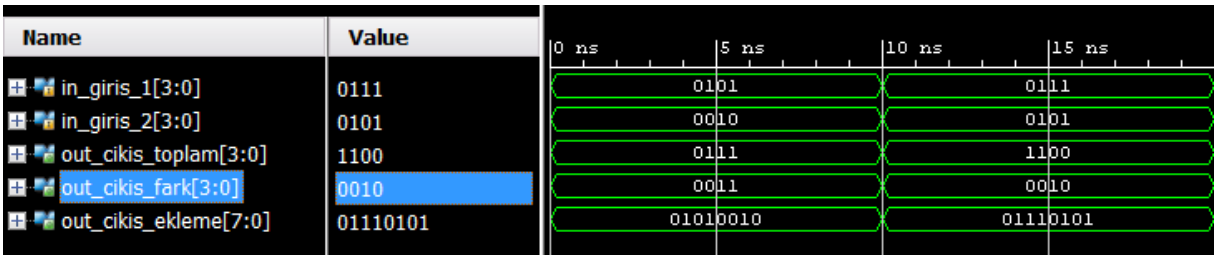
Örnek 4.2 : Toplama operatörlerinin kullanıldığı **toplama_operatorleri.vhd** VHDL kodu aşağıda gösterilmiştir. **toplama_operatorleri** varlığı 4 bitlik **in_giris_1** ve **in_giris_2** giriş portlarına sahiptir.

- 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.
5. entity toplama_operatorleri is
6.   Port (
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_toplam : out std_logic_vector(3 downto 0);
10.    out_cikis_fark : out std_logic_vector(3 downto 0);
11.    out_cikis_ekleme : out std_logic_vector(7 downto 0)
12.  );
13.end toplama_operatorleri;
14.
15.architecture Behavioral of toplama_operatorleri is
16.
17.begin
18.
19.  process(in_giris_1, in_giris_2)
20.  begin
21.
22.    out_cikis_toplam  <= in_giris_1 + in_giris_2;
23.    out_cikis_fark    <= in_giris_1 - in_giris_2;
24.    out_cikis_ekleme  <= in_giris_1 & in_giris_2;
25.
26.  end process;
27.
```

```
28.end Behavioral;
```



Şekil 4-2 toplama_operatorleri varlığının ilgili çıkışlara ilişkin benzetim çıktısı

4.4. 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

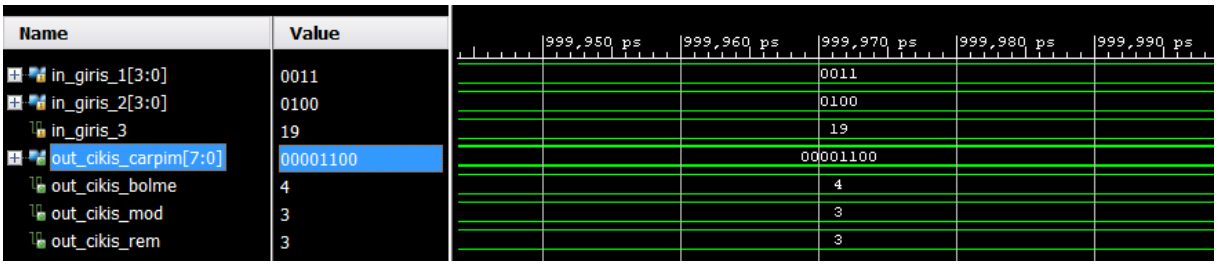
```
out_cikis <= in_giris_1 rem in_giris_2;
```

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.
5. entity carpim_operatorleri is
6.   Port (
7.     in_giris_1 : in std_logic_vector(3 downto 0) := "0011";
8.     in_giris_2 : in std_logic_vector(3 downto 0) := "0100";
9.     in_giris_3 : in integer := 19;
10.    out_cikis_carpim : out std_logic_vector(7 downto 0);
11.    out_cikis_bolme : out integer;
12.    out_cikis_mod : out integer;
13.    out_cikis_rem : out integer
14.  );
15.end carpim_operatorleri;
16.
17.architecture Behavioral of carpim_operatorleri is
18.
19.begin
20.
21.  process(in_giris_1, in_giris_2, in_giris_3)
22.  begin
23.
24.    out_cikis_carpim <= in_giris_1 * in_giris_2;
25.    out_cikis_bolme <= in_giris_3 / conv_integer(in_giris_2);
26.    out_cikis_mod <= in_giris_3 mod conv_integer(in_giris_2);
27.    out_cikis_rem <= in_giris_3 rem conv_integer(in_giris_2);
28.
29.  end process;
30.
31.end Behavioral;
```



Şekil 4-3 carpim_operatorleri varlığının ilgili çıkışlara ilişkin benzetim çıktısı

4.5. 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.

4.6. 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);
```

4.7. Nitelikler (Attributes)

Nitelikler önceden tanımlı ve kullanıcı tanımlı olmak üzere ikiye ayrılmaktadır.

4.7.1. 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.

4.7.2. 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";
```