5. VHDL Veri Nesneleri

VHDL'de tasarım yaparken kullanmak istediğimiz verileri yönetebileceğimiz üç temel veri nesnesi mevcuttur:

- **signal**
- **variable**
- **constant**

Veri nesnelerinden **signal** ve **variable** içeriği değişebilen verilerde kullanılırken sabit değerler için **constant** veri nesnesi kullanılmaktadır. Bilgisayar programlama dillerinden farklı olarak VHDL dili ile yapılan tasarımlar karşılığında fiziksel bir devre sentezlendiği unutulmamalı ve bu başlık altında anlatılan kavramlara her zaman bu açıdan bakılmalıdır.

5.1. **signal** Veri Nesnesi

signal veri nesneleri elektronik devreler üzerinde bulunan kablolara/yollara benzemektedir. Bu kablolar/yollar kullanılarak yapılan tasarımda, haberleşmesi gereken birimler birbiri ile veri alış-verişi yapabilmektedir. **signal** veri nesnesi sözdizimi aşağıdaki gibi gibidir.

```
signal sinyal_adı : tip_adi := baslangic_degeri;
```

Yukarıda verilen tanımlamada **:=** işaretinden sonra verilen kısımda tanımlanan sinyale ilk değer ataması (tanımlanan sinyalin tipine uygun şekilde) yapılmaktadır. Bu kullanım isteğe bağlı olup istendiği takdirde yapılmayabilir fakat genel bir tavsiye olarak ilk değer ataması önerilen bir kullanımdır.

Aşağıda **signal** bildirim örnekleri verilmiştir. Bu bildirimlerde:

- **elde** 1 bitlik **bit** tipinde,
- **toplam** 4 bitlik **bit_vector** tipinde,
- **saat** bir bitlik **std_logic** tipinde,
- **kelime** 10 bitlik **std_logic_vector** tipinde,
- **sayac** 0 ile 127 aralığında tanımlı **integer** tipinde
- **kosul** **boolean** tipinde tanımlanmış sinyallerdir.

```
signal elde : bit;
signal toplam : bit_vector(0 to 3);
signal saat : std_logic;
signal kelime : std_logic_vector(9 downto 0);
signal sayac : integer range 0 to 127;
signal kosul : boolean;
```

signal veri nesnesi VHDL kodunda dört yerde tanımlanabilmektedir ve tanımlama yerleri aşağıda birer örnekle açıklanmıştır.

1) **entity** tanımlamalarında :

```
..
..
entity ornek is
  port(
    toplam : std_logic_vector(3 downto 0)
  );
end ornek;
..
..
```

2) Mimarinin (**Architecture**) tanımlama bölümünde:

```
..
..
architecture Behavioral of ornek is
  signal bayrak: std_logic;
  signal toplam : bit_vector(0 to 3);
  signal sayac : integer range 0 to 255;
begin
..
..
```

3) Paket (**Package**) tanımlama bölümünde:

```
..
..
package ornek is
  signal bayrak : std_logic;
  signal toplam : bit_vector(0 to 3);
  signal sayac : integer range 0 to 255;
end ornek;
..
..
```

4) Blok (**Block**) tanımlama bölümünde :

```
..
..
block
  signal bayrak : std_logic;
  signal toplam : bit_vector(0 to 3);
  signal sayac : integer range 0 to 255;
```

```
end block;
..
..
```

5.2. **variable** Veri Nesnesi

variable veri nesnesi ile **signal** veri nesnesi arasındaki en önemli fark, **variable** veri nesnesinin içeriğinin hemen güncellenmesidir. Bu yüzden içeriğin sıklıkla değiştiği döngüler, değişken indisleri, saklanması gereken ara işlem değerleri v.b. gibi durumlarda kullanılmaktadır.

Aşağıda **variable** veri nesnesi kullanımına ait sözdizimi verilmiştir.

```
variable degisken_adi : tip_adi := baslangic_degeri;
```

Yukarıda verilen tanımlamada **:=** işaretinden sonra verilen kısımda tanımlanan **variable** değişkene ilk değer ataması (tanımlanan **variable** değişkenin tipine uygun şekilde) yapılmaktadır. Bu kullanım isteğe bağlı olup, istendiği takdirde yapılmayabilir fakat genel bir tavsiye olarak ilk değer ataması önerilen bir kullanımdır.[1]

variable veri nesnesi VHDL kodunda üç yerde tanımlanabilmektedir.

1) **process** tanımlama bölümünde:

```
..
..
process(…)
  variable bayrak : std_logic;
  variable toplam : bit_vector(0 to 3);
  variable sayac : integer range 0 to 255;
begin
  ..
  ..
end process;
..
..
```

2) **function** tanımlama bölümünde :

```
..
```

[1] VHDL-93 standartı ile birlikte VHDL diline **shared variable** kullanım imkânı sunulmuştur. Fakat **shared variable** nesneleri **sentezlenemeyen** yapılar olup daha çok benzetim ortamları için kullanılmaktadır. Bununla beraber yaratacağı olası sorunlar yüzünden (aynı **shared variable** nesnesinin birden fazla kaynak tarafından aynı anda değiştirilmek istenmesi gibi durumlar) kullanılması tavsiye edilmemektedir.

```
..
function fonksiyon_adi(parametreler listesi) return donus_tipi is
  variable bayrak : std_logic;
  variable toplam : bit_vector(0 to 3);
  variable sayac : integer range 0 to 255;
begin
  ..
  ..
end fonksiyon_adi;
..
..
```

3) **procedure** tanımlama bölümünde

```
..
..
procedure procedure_adi(parametreler listesi)is
  variable bayrak : std_logic;
  variable toplam : bit_vector(0 to 3);
  variable sayac : integer range 0 to 255;
begin
  ..
  ..
end procedure_adi;
..
..
```

5.3. **constant** Veri Nesnesi

signal ve **variable** nesnelerinin aksine **constant** nesnelerinin değeri değiştirilemez. Bu yüzden tasarımda kullanılacak sabit değerlerin saklanması için kullanılmalıdır. Bu fark dışında **constant** tanımlaması **signal** ve **variable** ile benzerdir. Aşağıda **constant** veri nesnesi kullanımına ait sözdizimi verilmiştir.

```
constant sabit_adi : tip_adi := sabit_deger;
```

Sabit kullanımının amacı, sayı veya değerin yerine, sabit ismi kullanarak kodun okunabilirliğini arttırmaktır. Aşağıda örnek **constant** bildirimleri verilmiştir:

```
constant bir : bit_vector(7 downto 0) := "00000001";
constant bir : std_logic_vector(7 downto 0) := "00000001";
constant bir : integer := 1;
```

Bu tanımlamayla birlikte VHDL kodunda **bir** değeri **bit_vector** tipinde 8 bitlik **"00000001"**, **std_logic_vector** tipinde 8 bitlik **"00000001"** ve **integer** tipinde 1 değerleri yerine kullanılabilmektedir.

constant veri nesnesi VHDL kodunda sekiz yerde tanımlanabilmektedir. Bunlar:

- **package,**
- **package**
- **body,**
- **block,**
- **entity,**
- **architecture,**
- **process,**
- **procedure,**
- **function**.

constant veri nesnesinin yukarıda verilen tanımlanma alanlarından **architecture** ve **package** içerisinde kullanım örnekleri aşağıda verilmiştir.

1) Mimarinin tanımlama bölümünde

```
..
..
architecture Behavioral ornek is
  constant bayrak : std_logic := '1';
  constant toplam : bit_vector(0 to 3) := "0010";
  constant sayac : integer range 0 to 255 := 128;
begin
..
..
```

2) Paket (**package**) tanımlama bölümünde.

```
..
..
package ornek is
  constant bayrak : std_logic := '1';
  constant toplam : bit_vector(0 to 3) := "0010";
  constant sayac : integer range 0 to 255 := 128;
end ornek;
..
..
```

5.4. VHDL'de Açıklama Metni Tanımlama

VHDL'de açıklama metinleri **'--'** karakterleri tanımlandıktan sonra açıklama metni yazılmaya başlanmaktadır. VHDL derleyicisi, bu değerleri **'--'** karakterleri tanımlandıktan sonraki satırdaki değerleri derleme esnasında yok sayacaktır.

```
-- Bu bir açıklama metnidir…
```

5.5. Veri Nesnelerinin Adlandırılması

Veri nesnelerinin adlandırılması için kurallar basittir. Adlandırma işlemi için beş kurala dikkat etmek gerekmektedir:

1) Adlandırılacak isim VHDL anahtar sözcüklerinden olmamalıdır.
2) Harf ile başlamalıdır.
3) Alt çizgi ile bitmemelidir,
4) Aynı anda 2 alt çizgi karakteri kullanılmamalıdır.
5) Türkçe'ye has karakterler **kullanılmamalıdır** (ç,ğ,ş,ı,ö) .

Kullanılabilir isimler örnek olarak,

- **kelime,**
- **Kelime10,**
- **kelime_10**
- **Kelime**

verilebilir. Kullanılamayacak isimlere de

- **_kelime,**
- **kelime__10,**
- **10Kelime,**
- **entity**

örnekleri verilebilir.

entity VHDL anahtar sözcüğü olduğu için tanımlanamaz. VHDL'de büyük veya küçük karakter kullanım önemli değildir. **'y'** ile **'Y'**, **'if'** ile **'IF'** aynıdır.

5.6. Veri Tipleri

VHDL kodunun daha etkin yazılması için VHDL veri tiplerinin bilinmesi gerekmektedir. Bu nedenle bu bölümde VHDL dilinin daha etkin kullanımı için temel veri tipleri tanıtılmaktadır. Belirtilen veri tipleri kullanılan kütüphanelerle birlikte ön tanımlı olarak gelmektedir.

5.6.1. **bit** ve **bit_cevtor** Tipleri

Bu tipler IEEE 1076 ve IEEE 1164 VHDL standartlarında önceden tanımlanmıştır. Bu nedenle bu tiplerin kullanımında kütüphaneye ihtiyaç yoktur. **bit** tanımlı nesneler **'0'** ve **'1'**değerlerini alabilirler. Aşağıda örnek **bit** tipinde **signal** bildirimleri verilmiştir:

```
signal clk : bit := baslangic_degeri;
```

bit_vector tanımlı nesneler ise **bit** nesnelerinden oluşan bir dizidir ve iki şekilde tanımlanmaktadır:

- **düşük_indis to yüksek_indis** sözdiziminde, bitlerden oluşan çoklu bitler tanımlamaları için kullanışlıdır. **hex_kelime** sinyalinde en anlamsız bit **düşük_indis**, en anlamlı bit ise **yüksek_indis** olmaktadır.

```
signal hex_kelime : bit_vector(0 to 3);
```

hex_kelime sinyali 4 bitlik diziyi göstermektedir. Bu sinyal çoklu olarak kullanılabileceği gibi tek tek de kullanılabilmektedir. Bu sinyalin tek tek kullanımına ait söz dizimi **hex_kelime(0), hex_kelime(1), hex_kelime(2), hex_kelime(3)** şeklinde olmaktadır. **bit_vector** tanımlı bir nesneye atama örneği aşağıda verilmiştir.

```
hex_kelime <= "0101";
```

Aynı zamanda atama işlemleri tek tek veya birkaç bit kullanımı ile de aşağıdaki örneklerdeki gibi yapılabilmektedir.

```
hex_kelime(0) <= '0',
hex_kelime(1) <= '1',
hex_kelime(2) <= '0',
hex_kelime(3) <= '1',
hex_kelime(2 downto 1) <= "10",
hex_kelime(0 to 1) <= "10",
```

- **yüksek_indis downto düşük_indis** sözdizimi, eğer sinyal ikili sayıların gösteriminde kullanılıyorsa kullanışlıdır. Bu sözdiziminde en anlamlı bit **yüksek_indis**, en anlamsız bit ise **düşük_indis** olmaktadır.

```
signal kelime_10 : bit_vector(9 downto 0);
..
..
kelime_10 <= "1010101010";
```

kelime_10 sinyali 10 bitlik diziyi göstermektedir. Bu sinyal çoklu olarak kullanılabileceği gibi tek tek de kullanılabilmektedir. Burada **kelime_10** sinyaline ait ilgili bit atamaları aşağıdaki gibidir:

```
kelime_10(9) <= '1';
kelime_10(8) <= '0';
kelime_10(7) <= '1';
kelime_10(6) <= '0';
kelime_10(5) <= '1';
kelime_10(4) <= '0';
kelime_10(3) <= '1';
kelime_10(2) <= '0';
kelime_10(1) <= '1';
kelime_10(0) <= '0';
```

Örnek 5.1: Aşağıda ise **bit** veri tipi kullanılarak tasarlanan yarı toplayıcı kodu verilmiştir. Kodlamada 2-7 satırları arasında görüleceği üzere **yari_topalyici_bit** varlığının (**entity**) port tanımlamaları yapılmıştır. **in_giris_1** ve **in_giris_2** portları **in** modunda **bit** veri tipindedir. **out_cikis** ve **out_cikis_elde** portları ise **out** modunda **bit** veri tipindedir. 14. satırda **in_giris_1** ve **in_giris_2** giriş portlarının **xor** işleminin sonucu **out_cikis** portuna atanmaktadır. 15. satırda **in_giris_1** ve **in_giris_2** giriş portlarının **and** işleminin sonucu **out_cikis_elde** portuna atanmaktadır.

```
1. entity yari_toplayici_bit is
2.   port (
3.     in_giris_1 : in bit;
4.     in_giris_2 : in bit;
5.     out_cikis : out bit;
6.     out_cikis_elde : out bit
7.   );
8. end yari_toplayici_bit;
9.
10.architecture Behavioral of yari_toplayici_bit is
11.
12.begin
13.
14.  out_cikis <= in_giris_1 xor in_giris_2;
15.  out_cikis_elde <= in_giris_1 and in_giris_2;
16.
17.end Behavioral;
```

5.6.2. std_logic ve std_logic_vector Tipleri

Aşağıda örnek **std_logic** tipinde **signal** bildirimleri verilmiştir:

```
signal clk : std_logic := baslangic_degeri;
```

Bu veri tipin kullanımı için VHDL kodumuzda, aşağıda verilen sözdizimi kütüphane kısmına eklenmelidir.

```
library ieee;
use ieee.std_logic_1164.all;
```

std_logic tipi ile **bit** tipi arasındaki temel fark **std_logic** tipinin sahip olabileceği durumların sayısının **bit** tipine oranla daha fazla olmasıdır. Bu açıklamada özellikle durum kelimesinin seçilmesinin yegâne nedeni, tanımlanan veri tipine göre sentezlenecek devrenin alabileceği değerlerin sadece **1** ve **0**'dan ibaret olmamasıdır. **std_logic** veri nesnesinin alabileceği değerler ve bu değerlerin sentezlenebilirlik durumları Tablo 4-1'de verilmiştir.

Tablo 4-1 **std_logic** veri nesnesinin alabileceği değerler ve sentezlenebilirlik durumu

Değerler	Açıklama	Sentezlenebilir
0	Güçlü 0	✓
1	Güçlü 1	✓
Z	Yüksek Empedans	✓
X	Bilinmeyen	✓
-	Önemsiz	×
L	Zayıf 0	×
H	Yüksek 1	×
U	Tanımlanmamış	×
W	Zayıf Bilinmeyen	×

Tablo 4-1'den de görüleceği üzere, ilk dört değer lojik devre sentezinde kullanılabilmektedir. Diğerleri ise sadece benzetim programlarında kullanılabilmektedir. Aşağıda örnek **signal** bildirimleri verilmiştir. **saat** sinyali tek bitlik **std_logic** tipinde, **hex_kelime** 4 bitlik **std_logic_vector** tipinde ve **kelime_12** 12 bitlik **std_logic_vector** tipinde sinyallerdir. Aynı şekilde bu tanımlamalar **variable** ve **constant** için de yapılabilir.

```
signal saat : std_logic;
signal hex_kelime : std_logic_vector(0 to 3);
signal kelime_12 : std_logic_vector(11 downto 0);
```

std_logic_vector türü üzerinde temel aritmetik işlemleri yapabilmek için VHDL kodunda aşağıda verilen kütüphanenin eklenmesi gerekmektedir. Bu sayede temel aritmetik işlemleri gerçekleştirmek mümkün olmaktadır.

```
use ieee.std_logic_signed.all;
```

Örnek 5.2: std_logic_signed paketi, "**+, -, ***" gibi aritmetik operatörlerin **std_logic_vector** sinyalleri ile kullanımına olanak sağlamaktadır. VHDL sentezleyicisi bu paket kullanıldığı zaman işaretli sayılar için devre oluşturacaktır. Bu pakete alternatif olarak **std_logic_unsigned** kullanılabilir. Bu durumda sentezleyici işaretsiz sayılar için devre oluşturacaktır. Aşağıda "**+, -, ***" işlemlerinin kullanıldığı **ornek_std_logic.vhd** VHDL kodu verilmiştir.

Kodda görüleceği üzere 1. satırda IEEE kütüphanelerinin kullanılacağına ilişkin bildirim yapılmaktadır. 2. satır da **std_logic** veri tipinin kullanımı içine gerekli kütüphane bildirimi yapılmaktadır. 3. satırda yapılan

tanımlama ile **std_logic** veri tipinde toplama ve çarpma işlemlerinin yapılması sağlanmaktadır. **ornek_std_logic** varlığının port tanımlama işlemleri 6-12. satırlar arasında yapılmıştır. **in_giris_1** ve **in_giris_2** giriş portları 2 bitlik **std_logic_vector** tipindedir. Toplam ve çıkarma işlemlerinin sonuçları yine 2 bitlik olacağından dolayı **out_cikis_toplam** ve **out_cikis_fark** çıkış portları 2 bitlik **std_logic_vector** tipindedir. Çarpma işleminin sonuçları çarpılacak olan değerlerin toplam bit sayısına eşit olması gerekmektedir. Bu nedenle **out_cikis_carpim** çıkış portu 4 bitlik **std_logic_vector** tipindedir. 19. satırda **in_giris_1** girişi ile **in_giris_2** girişi toplanarak **out_cikis_toplam** çıkış portuna atanmaktadır. 20. satırda **in_giris_1** girişinden **in_giris_2** girişi çıkarılarak **out_cikis_fark** çıkış portuna atanmaktadır. 21. satırda **in_giris_1** girişi ile **in_giris_2** girişi çarpılarak **out_cikis_carpim** çıkış portuna atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3. use IEEE.STD_LOGIC_SIGNED.all;
4.
5. entity ornek_std_logic is
6.   port(
7.     in_giris_1 : in  std_logic_vector(1 downto 0);
8.     in_giris_2 : in std_logic_vector(1 downto 0);
9.     out_cikis_toplam : out std_logic_vector(1 downto 0);
10.    out_cikis_fark : out std_logic_vector(1 downto 0);
11.    out_cikis_carpim : out std_logic_vector(3 downto 0)
12.  );
13.end ornek_std_logic;
14.
15.architecture Behavioral of ornek_std_logic is
16.
17.begin
18.
19.  out_cikis_toplam <= in_giris_1 + in_giris_2;
20.  out_cikis_fark <= in_giris_1 - in_giris_2;
21.  out_cikis_carpim <= in_giris_1 * in_giris_2;
22.
23.end Behavioral;
```

5.6.3. std_ulojik Tipi

std_logic tipi VHDL'de en çok kullanılan tip olmasına rağmen **std_ulogic** tipinin alt tipidir. **std_ulogic** tipindeki sinyal değerleri, **std_logic** tipindeki benzer değerleri almaktadır. Bu iki tip arasındaki fark, çözüm fonksiyonu kavramlarının kullanımı ile alakalıdır. VHDL'de çözüm fonksiyonu, bir sinyal için iki kaynak bulunduğunda sinyalin alacağı değeri belirlemede kullanılır.

5.6.4. signed ve unsigned Tipleri

signed ve **unsigned** veri tipi tanımlamaları **std_logic_vector**'e benzer şekilde yapılmaktadır. **signed** tipi VHDL dilinde kodda işaretli sayılar (2'ye tümleyen) için kullanılır. **unsigned** tipi işaretsiz sayılar için kullanılır.

Aşağıda örnek **signed** ve **unsigned** tipinde **signal** bildirimleri verilmiştir. **kelime_unsigned** sinyali 4 bitlik işaretsiz olarak tanımlanmıştır. Yani bu sinyal **0** ile **15** arasında on tabanındaki sayı değerlerini alabilecektir. **kelime_signed** sinyali 4 bitlik işaretli olarak tanımlandığından **-8** ile **7** arasında değerleri alabilir.

```
signal kelime_unsigned : unsigned(3 downto 0);
signal kelime_signed : signed(3 downto 0);
```

Bu veri tiplerinin kullanımı için aşağıda verilen paketin kütüphane bildirimine eklenmesi gerekmektedir.

```
use ieee.std_logic_arith.all;
```

Bu paket aritmetik operatörlerin gerçeklenmesinde kullanılan devre tiplerini tanımlar. **signed** ve **unsigned** tiplerini içermektedir. Bu tipler, **std_logic_vector** tipi ile temelde aynıdır. Çünkü **std_logic** sinyalleri bir dizi halinde gösterirler. **signed** ve **unsigned** tipleri, kullanıcının VHDL kodunda kullanılan sayı gösterim çeşidini belirlemesine izin vermektedir.

sinyal_1 ve **sinyal_2**'nin **signed** tipinde tanımlı 4 bitlik sinyaller olduğunu farz edelim. Bu iki sinyal arasındaki "**sinyal_1 < sinyal_2**" karşılaştırması yapılırsa **signed** tipinde sonuç doğru olacaktır. Çünkü **sinyal_1** sinyali **-6** değerini temsil etmekte ve **sinyal_2** sinyali **2** değerini temsil etmektedir.

```
sinyal_1 <= "1001"; -- signed sayı -6
sinyal_2 <= "0010"; -- signed sayı 2
```

sinyal_1 ve **sinyal_2**'nin **unsigned** tipinde tanımlı 4 bitlik sinyaller olduğunu farz edelim. Bu iki sinyal arasındaki "**sinyal_1 < sinyal_2**" karşılaştırması yapılırsa **unsigned** tipinde sonuç yanlış olacaktır. Çünkü **sinyal_1** sinyali **9** değerini temsil etmekte ve **sinyal_2** sinyali **2** değerini temsil etmektedir.

```
sinyal_1 <= "1001"; -- unsigned sayı 9
sinyal_2 <= "0010"; -- unsigned sayı 2
```

std_logic_signed paketinde, **std_logic_vector** tipine **signed** tipindeki gibi davranışlar tanımlanmıştır. Aynı şekilde **std_logic_unsigned** paketinde, **std_logic_uvector** tipine **unsigned** tipindeki gibi davranışlar tanımlanmıştır.

Örnek 5.3: Aşağıda verilen **ornek_signed.vhd** kodunda 1. satırda IEEE kütüphanelerinin kullanılacağına ilişkin bildirim yapılmaktadır. 2. satır da **std_logic** veri tipinin kullanımı içine gerekli kütüphane bildirimi yapılmaktadır. 3. satırda yapılan tanımlama ile **std_logic** veri tipinde toplama ve çarpma işlemlerinin yapılması sağlanmaktadır. 4. satırda yapılan tanımlama ile **signed** ve **unsigned** tanımlama işlemlerinin yapılması sağlanmaktadır. **ornek_signed** varlığının port tanımlama işlemleri 7-13. satırlar arasında yapılmıştır. **in_giris_1** ve **in_giris_2** giriş portları 4 bitlik **std_logic_vector** tipindedir. **in_giris_3** giriş portu ise 4 bitlik **signed** tipindedir. **out_cikis_1** çıkış portu 4 bitlik **std_logic_vector** tipindedir. **out_cikis_2** çıkış portu ise 4 bitlik **signed** tipindedir. 20. satırda tanımlanan söz diziminde **in_giris_3** giriş port **std_logic_vector** tipine dönüştürüldükten sonra 1 eklenerek **out_cikis_1** çıkış portuna atanmaktadır. 21. satırda **in_giris_1** ve **in_giris_2** giriş portları toplandıktan sonra **signed** tipine dönüştürülmekte ve **out_cikis_2** çıkış portuna atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3. use IEEE.STD_LOGIC_SIGNED.all;
4. use IEEE.STD_LOGIC_ARITH.all;
5.
6. entity ornek_signed is
7.   port (
8.     in_giris_1 : in std_logic_vector(3 downto 0);
9.     in_giris_2 : in std_logic_vector(3 downto 0);
10.    in_giris_3 : in signed(3 downto 0);
11.    out_cikis_1 : out std_logic_vector(3 downto 0);
12.    out_cikis_2 : out signed(3 downto 0)
13.  );
14.end ornek_signed;
15.
16.architecture Behavioral of ornek_signed is
17.
18.begin
19.
20.  out_cikis_1 <= std_logic_vector(in_giris_3) + 1;
21.  out_cikis_2 <= signed(in_giris_1 + in_giris_2);
22.
23.end Behavioral;
```

5.6.5. INTEGER Tipi

VHDL standardında aritmetik operatörlerin kullanımı ve ikili sayıların gösterimi için **integer** tipi kullanmak da mümkündür. **integer** tipinde, **std_logic_vector** tipinden farklı olarak bit sayısı belirtmeye gerek yoktur. Varsayılan olarak **integer** sayı tipi 32 bittir ve **$-(2^{31}-1)$** ile **$2^{31} - 1$** aralığındaki sayıları göstermektedir. Aşağıda örnek **integer** tipinde **signal**, **constant**, **variable** ve port bildirimleri verilmiştir.

```
sayi : in integer;
signal sayi : integer;
constant sayi : integer;
variable sayi : integer;
```

integer tipinde sayaç kullanılacak ve bu sayaç 0 ile 15 arasında değer alacaksa **integer** veri tipinin tüm bitlerinin kullanımın yerine sadece 5 bitinin (1 bit işaret biti, 4 bit değer bitleri) kullanımı yeterli olacaktır. Bu örnekte olduğu gibi **integer** tipinde, kullanılacak olan sayı aralığı tasarım aşamasında belirlenmiş ise bu sayı aralıkları tanımlanabilmektedir. Bu şekilde VHDL sentezleyicisi tanımlı olan aralığa uygun bit uzunluğunda

sentezleme işlemi yapabilmektedir. Sayı aralığı VHDL dilinde **range to/downto** sözdizimleri ile gerçeklenmektedir. Aşağıda tanımlanan sayı gösterimleri için sentezleyici 7 (1 bit işaret biti, 6 bit değer bitleri) bitlik değerler oluşturacaktır.

```
sayi : in integer range -63 to 63;
signal sayi : integer range -63 to 63;
constant sayi : integer range -63 to 63;
variable sayi : integer range -63 to 63;
```

Örnek 5.4: Aşağıda verilen **ornek_integer.vhd** VHDL kodunda 1. satırda IEEE kütüphanelerinin kullanılacağına ilişkin bildirim yapılmaktadır. 2. satır da **integer** veri tipinin kullanımı ve aritmetik işlemler için gerekli kütüphane bildirimi yapılmaktadır. **ornek_integer** varlığının port tanımlamaları 5-11. satırlar arasında yapılmıştır. Giriş-çıkış portlarının tümü **integer** tipinde tanımlanmıştır. **in_giris_1** giris portu 0 ile 15 aralığında tanımlanmıştır ve bu giriş portu derleyici tarafından 5 bit (1 bit işaret biti, 4 bit değer bitleri) ile sentezlenebilmektedir. **in_giris_2** giriş portu 0 ile 31 aralığında tanımlanmıştır ve bu giriş portu derleyici tarafından 6 bit (1 bit işaret biti, 5 bit değer bitleri) ile sentezlenebilmektedir. **in_giris_3** giriş portu 0 ile 128 aralığında tanımlanmıştır ve bu giriş portu derleyici tarafından 9 bit (1 bit işaret biti, 8 bit değer bitleri) ile sentezlenebilmektedir. **out_cikis_1** çıkış portu 0 ile 511 aralığında tanımlanmıştır ve bu çıkış portu derleyici tarafından 10 bit (1 bit işaret biti, 9 bit değer bitleri) ile sentezlenebilmektedir. **out_cikis_2** çıkış portunda aralık tanımlaması yapılmadığından bu çıkış portu derleyici tarafından 32 bit ile sentezlenebilmektedir. Kodda 18. satırda **in_giris_1** ve **in_giris_2** giriş portları çarpılarak **out_cikis_1** çıkış portuna atanmaktadır. 19. satırda **in_giris_2** ve **in_giris_3** giriş portları çarpılarak **out_cikis_2** çıkış portuna atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity ornek_signed is
5.   port (
6.     in_giris_1 : in integer range 0 to 15;
7.     in_giris_2 : in integer range 0 to 31;
8.     in_giris_3 : in integer range 0 to 128;
9.     out_cikis_1 : out integer range 0 to 511;
10.    out_cikis_2 : out integer
11.  );
12.end ornek_signed;
13.
14.architecture Behavioral of ornek_signed is
15.
16.begin
17.
18.  out_cikis_1 <= in_giris_1 * in_giris_2;
19.  out_cikis_2 <= in_giris_2 * in_giris_3;
20.
21.end Behavioral;
```

std_logic_vector veri tipinde aritmetik işlemleri yapabilmek için **std_logic_signed** veya **std_logic_unsigned** kütüphanelerine ihtiyaç duyulmaktadır. **integer** veri tipinde aritmetik işlemler **std_logic_1164** kütüphanesinde tanımlanmıştır.

5.6.6. boolean Tipi

boolean tipinde bir nesne iki değişkene sahiptir:

- **TRUE**
- **FALSE**

Eğer sonuç **TRUE** ise çıkış 1, **FALSE** ise çıkış 0 olmaktadır. Bu tipe ait tanımlama aşağıda verilmiştir.

```
signal bayrak : boolean;
```

5.6.7. Listeleme Tipi (Enumaration)

Listeleme tipi kullanıcının belirlediği olası değerlerdir. Bu tipe ait genel kullanıcı şekli aşağıda verilmiştir.

```
type listeleme_tipi_adi is (isim [, isim]);
```

Köşeli parantez, bir veya daha fazla eklenebilir öğelerin içirilebileceğini göstermektedir. Bu tip en çok sonlu durum makinelerinin durum gösterimi için kullanılır.

Aşağıda bir durum makinasında listeleme tipi kullanım örneği verilmiştir. Örnekten de görüleceği üzere **t_Kontrol** tipi **BOSTA**, **BASLA** ve **DUR** ifadeleri ile tanımlanmıştır. **r_Kontrol** sinyali de **t_Kontrol** tipinde tanımlanmıştır. Tanımlanan **r_Kontrol** sinyalinin değerleri **BOSTA, BASLA** ve **DUR'**dır. Kod VHDL sentezleyicisi tarafından derlendiği zaman, bit kalıplarına **BOSTA, BASLA** ve **DUR** gösterimleri atanmaktadır.

case r_Kontrol is söz dizimi ile oluşturulan durum makinasında, **when BOSTA =>** söz dizimi ile **r_Kontrol** sinyali **BOSTA** olduğu durumlarda bu duruma ait işlemler aktif olmaktadır. Aynı şekilde **when BASLA =>** söz dizimi ile r_Kontrol sinyali **BASLA** olduğu durumda ve **when DUR =>** söz dizimi ile **r_Kontrol** sinyali **DUR** durumunda bu durumlara ait işlemler aktif olmaktadır.

```
type t_Kontrol is (BOSTA, BASLA, DUR);
signal r_Kontrol : t_Kontrol;
..
..
case r_Kontrol is
    when BOSTA =>
        ..
        ..

    when BASLA =>
        ..
        ..

    when DUR =>
```

```
        ..
        ..

    when others => NULL;
end case;
```

5.6.8. Tip Dönüşümleri

VHDL güçlü bir tip kontrol dilidir. Bunun anlamı, bir sinyal tipinin, başka bir sinyal tipine atanmasına izin vermemektedir. Hatta **bit** ve **std_logic** gibi, birbiri ile uyumlu gibi görünen sinyallerin birarada kullanmasına da izin vermez. Bu durumu önlemek amacı ile kodlamaya başlamadan önce kullanılacak olan tip önceden belirlenmelidir. Kodlamada tip dönüşümü gereksinimi duyulduğunda bir türden diğer türe, tip dönüşüm fonksiyonu ile geçilebilmektedir.

Örneğin **integer** tipindeki bir sayıyı **std_logic_vector** tipine dönüştürelim. Dönüştürmek istediğimiz **integer** sayı **X** değişkenine atansın. Bu sayı 12 bitlik bir **std_logic_vector** tipinde **Y** değişkenine atanmak istenirse aşağıdaki işlem yapılmaktadır.

```
Y <= conv_std_logic_vector(X, 12);
```

Aynı şekilde **conv_std_logic_vector** tipindeki **Y** sayısını, **integer** tipindeki **X** değişkenine atama işlemi aşağıdaki gibi yapılmaktadır.

```
X <= conv_integer(Y);
```

Örnek 5.5: Aşağıda verilen **ornek_conv.vhd** VHDL kodunda 1-4. satırlarda gerekli kütüphanelerin kullanılacağına ilişkin bildirim yapılmaktadır. **ornek_conv** varlığının port tanımlamaları 7-12. satırlar arasında yapılmıştır. **in_giris_1** giriş portu 10 bitlik **std_logic_vector** tipinde ve **in_giris_2** giriş portu 0 ile 511 aralığında tanımlanmış **integer** tipindedir. **out_cikis_1** çıkış portu **integer** tipinde ve **out_cikis_2** portu 10 bitlik **std_logic_vector** tipindedir. Kodda 19. satırda **in_giris_1** portu **integer** tipine dönüştürüldükten sonra **out_cikis_1** portuna atanmaktadır. 20. satırda **in_girsi_2** portu 10 bitlik **std_logic_vector** tipine dönüştürüldükten sonra **out_cikis_2** portuna atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3. use IEEE.STD_LOGIC_SIGNED.all
4. use IEEE.STD_LOGIC_ARITH.all;
5.
6. entity ornek_conv is
7.   port (
8.     in_giris_1 : in std_logic_vector(9 downto 0);
9.     in_giris_2 : in integer range 0 to 511;
10.    out_cikis_1 : out integer;
11.    out_cikis_2 : out std_logic_vector(9 downto 0)
12.  );
```

```
13.end ornek_conv;
14.
15.architecture Behavioral of ornek_conv is
16.
17.begin
18.
19.  out_cikis_1 <= conv_integer(in_giris_1);
20.  out_cikis_2 <= conv_std_logic_vector(in_giris_2, 10);
21.
22.end Behavioral;
```

5.6.9. Alt-Tipler (Subtype)

VHDL dili var olan veri tiplerini kullanarak alt tipler oluşturmaya izin vermektedir. Bu sayede var olan veri tipinin özellikleri korunarak alt tipler oluşturmak mümkün olmaktadır. VHDL dilinde **subtype** söz dizimi ile tanımlı olan tiplerden, sınırlı alt tipler oluşturulabilmektedir. Aşağıda **subtype** söz dizimi verilmiştir.

```
subtype alt_tipi_adi is ana_tip sinir_limiti;
```

Aşağıda örnek alt tip tanımlamaları verilmiştir.

```
subtype hex_kelime is std_logic_vector(3 downto 0);
subtype byte is bit_vector(7 downto 0);
subtype sayac is integer range 0 to 15;
subtype dogal_sayilar is integer range 0 to 2^31-1;
subtype pozitif_sayilar is integer range 1 to 2^31-1;
```

5.6.10. Diziler

std_logic_vector ve **bit_vector** tipleri **std_logic** ve **bit** sinyallerinden oluşan dizilerdir. Bu dizilerin VHDL standardına göre gösterimi aşağıdaki gibidir.

```
type bit_vector is array (natural range<>) of bit;
type std_logic_vector is array (natural range<>) of std_logic;
```

Yukarıdaki gösterimden de görüleceği üzere dizilerin uzunlukları belirtilmemiştir. Dizi uzunluğu veri tipi tanımlanacağı zaman kullanıcı tarafından belirtilir.

```
type t_Kelime_10 is array (9 downto 0) of std_logic;
signal r_Kelime_10 : t_Kelime_10;
```

Yukarıda **t_Kelime_10** tipi ile tanımlanan **r_Kelime_10** sinyali, 10 elemanlı **std_logic** veri tipinden meydana gelmektedir.

VHDL'de diziler çok boyutlu olabilir. Dizi tanımlamalarını kavramak için öncelikle bir boyutlu dizi tanımlaması ile ilgili örnek aşağıda verilmiştir:

```
type t_dizi_1d is array (2 downto 0) of std_logic_vector(3 downto 0);
signal r_dizi_1d : t_dizi_1d;
```

r_dizi_1d sinyali her biri 4 bitlik **std_logic_vector** tipinde sinyallerinden oluşan 3 bileşenden meydana gelmektedir. Aşağıda **r_dizi_1d** bileşeni için bir atama işlemi gösterilmektedir.

```
r_dizi_1d <= ("0010", "1100", "1001"); -- Tüm değerlerin ataması.
r_dizi_1d(0) <= "1001"; -- 0. Elemanın değer ataması.
r_dizi_1d(1) <= "1100"; -- 1. Elemanın değer ataması.
r_dizi_1d(2) <= "0010"; -- 2. Elemanın değer ataması.
r_dizi_1d(2)(1) <= '1'; -- 2. Elemanın 1. bitine değer ataması.
```

VHDL ile iki boyutlu dizi tanımlaması bir boyutlu ile benzerdir. Aşağıda iki boyutlu dizi tanımlamasına ait örnek kullanım verilmiştir.

```
type t_dizi_2d is array (0 to 1, 2 downto 0) of std_logic_vector(3
downto 0);
signal r_dizi_2d : t_dizi_2d;
```

array_2d sinyali her biri 4 bitlik **std_logic_vector** sinyallerinden oluşan 6 bileşenden meydana gelmektedir. Aşağıda **array_2d** bileşeni için bir atama işlemi gösterilmektedir.

```
r_dizi_2d <= (("0010", "1100", "1001"),
              ("1101", "0011", "0110")); -- Tüm değerlerin atanması.
r_dizi_2d(0) <= ("0010", "1100", "1001");-- 0. satıra değer atanması.
r_dizi_2d(1) <= ("1101", "0011", "0110");-- 1. Satıra değer atanması.
r_dizi_2d(0, 2) <= "0010";--0. satırın 2. elemanına değer atanması.
r_dizi_2d(0, 1) <= "1100";--0. satırın 1. elemanına değer atanması.
r_dizi_2d(1, 1) <= "0011";--1. satırın 1. elemanına değer atanması.
r_dizi_2d(1, 0) <= "0110";--1. satırın 0. elemanına değer atanması.
r_dizi_2d(1, 0)(1) <= '1';--1. satır, 0. elemanın, 1. bitine değer
atama.
r_dizi_2d(1, 0)(0) <= '0';--1. satır, 0. elemanın, 0. bitine değer
atama.
```

5.6.11. Port Dizileri

VHDL dilinde **entity** kısmında doğrudan istenilen tipte veri nesnesi tanımlayıp kullanmak mümkün değildir. VHDL dili ancak önceden tanımlanmış veri nesnelerinin **entity** kısmında kullanılmasına izin vermektedir. Bu

kısıtlama ile en çok karşılaşılan durum ise **entity** kısmında çok boyutlu dizilerin kullanılmaya çalışıldığı anlardır.

Bu kısıtlamadan kurtulmak ve istediğiniz veri nesnesini (kendinize özel tanımladıklarınız da dâhil olmak üzere) kullanmak için kullanılan yöntem ise **package** tanımlamasıdır.

package içinde tanımlanan veri nesneleri ilgili **package**'in çağrıldığı tüm proje dosyaları tarafından **entity** kısmı da dahil olmak üzere her yerde kullanılabilir. **package** kullanımı ile ilgili ayrıntılı bilgili daha sonraki bölümlerde anlatılacak olup basit bir örnek aşağıda verilmiştir.

Örnek 5.6: Aşağıda port dizi tanımlama işleminin yapıldığı **port_dizi_paket.vhd** VHDL kodu verilmiştir. 6. satırda her biri 8 bitlik **std_logic_vector** tipinde tanımlanmış 4 bileşenden oluşan **port_dizi** tipi tanımlanmıştır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. package port_dizi_paket is
5.
6.   type port_dizi is array (3 downto 0) of std_logic_vector(0 to 7);
7.
8. end port_dizi_paket;
```

Aşağıda port dizi tanımlama işleminin yapıldığı **port_dizi_ornek.vhd** VHDL kodu verilmiştir. 3. satırda **port_dizi_paket package** tanımlama işlemi yapılarak **port_dizi** tipinin **port_dizi_ornek** varlığında kullanılabilir hale gelmesi sağlanmıştır. 7. satırda **in_giris** giriş portu **port_dizi** tipinde tanımlanmıştır. 8. satırda **out_cikis** çıkış portu **port_dizi** tipinde tanımlanmıştır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use work.port_dizi_paket.all;
4.
5. entity port_dizi_ornek is
6.   Port (
7.     in_giris : in port_dizi;
8.     out_cikis : out port_dizi
9.   );
10.end port_dizi_ornek;
11.
12.architecture Behavioral of port_dizi_ornek is
13.
14.begin
15.
16.  out_cikis <= in_giris;
17.
18.end Behavioral;
```

5.6.12. Record Tanımlaması

record yapısı diziler ile benzer olmakla berber en önemli farkı içeriğinde farklı veri tiplerine izin veriyor olmasıdır. Bir **record** tanımlaması içinde birbirinden farklı tiplerde veriler bulunabilir. Aşağıda **record** tanımlamasına ait kullanım verilmiştir.

```
type record_ornek is record
   bilesen_adi : bilesen_tipi;
   bilesen_adi : bilesen_tipi;
            …
      …
end record;
```

5.7. Veri Nesnesi Değerleri ve Numaraları

signal veri nesnesi devredeki tekli lojik sinyalleri (bit), çoklu lojik sinyalleri ve ikili sayıların gösteriminde kullanılır:

- Tekli **signal** değeri (bit) tekli tırnak ile gösterilmektedir ('0', '1').
- Çoklu **signal** değeri ise çift tırnak değeri ile gösterilmektedir ("0010", "1100101"). Çoklu **signal** gösterimi aynı zamanda ikili sayıların gösteriminde de kullanılır.
- **integer** veri nesnelerinin gösteriminde tırnak işareti kullanılmaz (**9**, **123**).

constant ve **variable** veri nesnelerinde gösterimi **signal** ile aynı şekilde yapılmaktadır.

5.8. Çoklu Veri Nesnesi Değer Atanması

Aşağıda çoklu sinyal atamaları örnekleri gösterilmiştir. Bu gösterimlerin hepsi aynı atamayı ifade etmektedir.

```
kelime <= "001100110011"; -- İkilik Tabanda(Binary) Atama İfadeleri
kelime <= B"001100110011"; -- İkilik Tabanda(Binary) Atama İfadeleri
kelime <= O"1463"; -- Sekizlik Tabanda (Octal) Atama İfadesi
kelime <= X"333"; -- Onaltılık Tabanda (Hex) Atama İfadesi
```