# 8. VHDL'de Sıralı Atama Sözdizimleri

VHDL dilinde kodların anlamlı bir şekilde sözdizimleriyle düzenlenmesi için kullanıcıya sıralı atama sözdizimleri sağlamıştır. Sıralı atama sözdizimleri **`if`**, **`case`** ve **`loop`** olmak üzere 3 çeşittir. Bu sıralı sözdizimlerinin kullanımları mimari içerisinde **`process`** söz dizimi ile yapılmalıdır.

## 8.1. PROCESS Sözdizimi

Sıralı sözdizimleri, VHDL dilinde paralel işlemler için kullanılan sözdizimlerinden farklı olarak kullanılmalıdır. Bu nedenle VHDL dilinde sıralı sözdizimleri mimari içerisinde **`process`** sözdizimi kullanılarak tanımlanmaktadır. **`process`** için genel tanımlama aşağıda verilmiştir.

```
..
..
process_etiketi:process(sinyal_adi {, sinyal_adi})
      variable tanımlama
begin
      wait söz dizimleri
      basit sinyal atamaları
      variable atama sözdizimleri
      if söz dizimleri
      case söz dizimleri
      loop söz dizimleri
end process process_etiketi;
..
..
```

**`process`**'lerin tasarlandıkları işleri yapabilmeleri için dışarıdan uygulanan bir tetikleme işaretine ihtiyaçları vardır. Bu tetikleme işareti herhangi bir sinyalin değişimi olabileceği gibi, harici bir kaynak tarafından üretilen bir işarette olabilir.

**`process`** sözdizimi içerisinde parantezle tanımlanan sinyaller, tasarlanan devrenin hassasiyet listesini oluşturmaktadır. Hassasiyet listesinde bulunan sinyallerde meydana gelen değişimler ile **`process`** aktif hale gelmektedir. **`variable`** söz dizimi tanımlamaları **`process`** içerisinde yapılmaktadır. **`process`** dışında kullanılacak **`variable`** değeri, **`signal`** sinyaline atanarak aktarılır.

**`Örnek 8.1:`** **`process`** söz dizimi için **`process_ornek_1.vhd`** VHDL kodu aşağıda verilmiştir. Kodda 17. satırda **`process`** söz dizimi parantez içerisinde yazılan sinyaller, tasarlanan devrenin hassasiyet listesini

oluşturmaktadır. **process_ornek_1** varlığının hassasiyet listesini **in_giris_1**, **in_giris_2** ve **in_giris_3** oluşturmaktadır. Yani **in_giris_1, in_giris_2** ve **in_giris_3** giriş portlarından biri değişim gösterdiği zaman **process** aktif hale gelecektir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity process_ornek_1 is
5.   Port (
6.     in_giris_1 : in std_logic;
7.     in_giris_2 : in std_logic;
8.     in_giris_3 : in std_logic;
9.     out_sonuc : out std_logic
10.  );
11.end process_ornek_1;
12.
13.architecture Behavioral of process_ornek_1 is
14.
15.begin
16.
17.  process_etiketi:process(in_giris_1, in_giris_2, in_giris_3)
18.  begin
19.
20.    out_cikis <= (in_giris_1 and in_giris_2) or in_giris_3;
21.
22.  end process process_etiketi;
23.end Behavioral;
```

Yukarıdaki verilen **process_ornek_1** varlığının ait örnek girişlerin ürettiği sonuçlar Tablo 8-1'de verilmiştir ve Şekil 8-1'de benzetim çıktısı gösterilmiştir. Tablo 8-1'de verilen sonuçlarda:

1. adımda hassasiyet listesinde bulunan tüm giriş portlarında değişim meydana geldiğinden **process** aktif hale gelmektedir. **process**'in aktif hale gelmesi ile birlikte **out_cikis** çıkış portuna (**0 and 1**) **or 1** işleminin sonucu atanmaktadır ve **out_cikis** çıkış portu değeri **'1'** olmaktadır.

2. adımda hassasiyet listesinde bulunan tüm giriş portlarında değişim meydana geldiğinden **process** aktif hale gelmektedir. **process**'in aktif hale gelmesi ile birlikte **out_cikis** çıkış portuna (**1 and 0**) **or 0** işleminin sonucu atanmaktadır ve **out_cikis** çıkış portu değeri **'0'** olmaktadır.

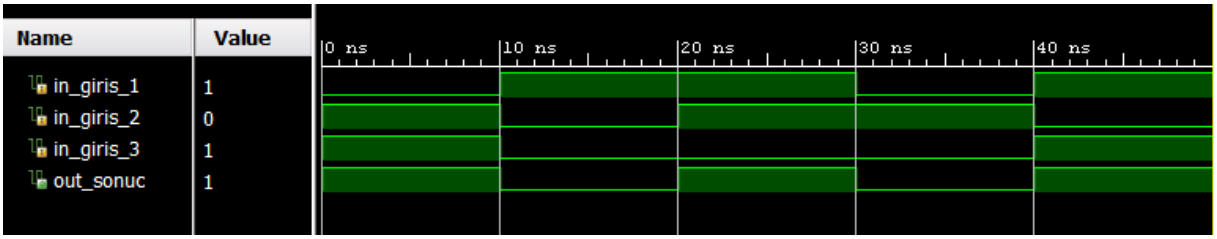
3. adımda hassasiyet listesinde bulunan **in_giris_2** giriş portunda değişim meydana geldiğinden **process** aktif hale gelmektedir. **process**'in aktif hale gelmesi ile birlikte **out_cikis** çıkış portuna (**1 and 1**) **or 0** işleminin sonucu atanmaktadır ve **out_cikis** çıkış portu değeri **'1'** olmaktadır.

4. adımda hassasiyet listesinde bulunan **in_giris_1** giriş portunda değişim meydana geldiğinden **process** aktif hale gelmektedir. **process**'in aktif hale gelmesi ile birlikte **out_cikis** çıkış portuna (**0 and 1**) **or 0** işleminin sonucu atanmaktadır ve **out_cikis** çıkış portu değeri **'0'** olmaktadır.

5. adımda hassasiyet listesinde bulunan tüm giriş portunda değişim meydana geldiğinden **process** aktif hale gelmektedir. **process**'in aktif hale gelmesi ile birlikte **out_cikis** çıkış portuna (**1 and 0**) **or 1** işleminin sonucu atanmaktadır ve **out_cikis** çıkış portu değeri **'1'** olmaktadır.

Tablo 8-1 process_ornek_1 varlığının ilgili girişlere ait ürettiği çıkış değerleri

| | in_giris_1 | in_giris_2 | in_giris_3 | out_cikis |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 |
| 2 | 1 | 0 | 0 | 0 |
| 3 | 1 | 1 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 |




Şekil 8-1 process_ornek_1 varlığı benzetim çıktısı

**Örnek 8.2:** Aşağıda hassasiyet listesinde tek değişkenin bulunduğu **process_ornek_2.vhd** VHDL kodu verilmiştir. Kodda 17. satırda **process** söz dizimi içerisinde parantezle yazılan **in_giris_3** sinyali tasarlanan **process_ornek_2** varlığının hassasiyet listesini oluşturmaktadır. Yani **in_giris_3** sinyalinde değişim meydana geldiği zaman **process** aktif hale gelecektir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity process_ornek_2 is
5.   Port (
6.      in_giris_1 : in std_logic;
7.      in_giris_2 : in std_logic;
8.      in_giris_3 : in std_logic;
9.      out_sonuc : out std_logic
10.  );
11.end process_ornek_2;
12.
13.architecture Behavioral of process_ornek_2 is
14.
15.begin
16.
```

```
17.   process_etiketi:process(in_giris_3)
18.   begin
19.
20.       out_sonuc <= (in_giris_1 and in_giris_2) or in_giris_3;
21.
22.   end process process_etiketi;
23.end Behavioral;
```

**process_ornek_2** varlığının ait örnek girişlerin ürettiği sonuçlar Tablo 8-2'de verilmiştir ve Şekil 8-2'de gösterilmiştir çıktısı verilmiştir. Tablo 8-2'de verilen sonuçlarda:

1. adımda **process** hassasiyet listesinde bulunan **in_giris_3** giriş portunda değişim meydana geldiğinden **process** aktif hale gelmektedir. **process**'in aktif hale gelmesi ile birlikte **out_cikis** çıkış portuna (**0 and 1**) **or 1** işleminin sonucu atanmaktadır ve **out_cikis** çıkış portu değeri **'1'** olmaktadır.

2. adımda **process** hassasiyet listesinde bulunan **in_giris_3** giriş portunda değişim meydana geldiğinden **process** aktif hale gelmektedir. **process**'in aktif hale gelmesi ile birlikte **out_cikis** çıkış portuna (**1 and 0**) **or 0** işleminin sonucu atanmaktadır ve **out_cikis** çıkış portu değeri **'0'** olmaktadır.

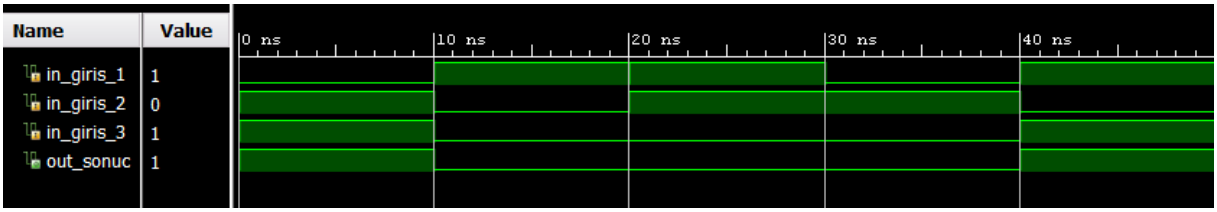
3. adımda **process** hassasiyet listesinde bulunan **in_giris_3** giriş portunda değişim meydana gelmediğinden **process** aktif hale gelmemektedir. Bu nedenle **out_cikis** çıkış portu değeri değişmemiştir.

4. adımda **process** hassasiyet listesinde bulunan **in_giris_3** giriş portunda değişim meydana gelmediğinden **process** aktif hale gelmemektedir. Bu nedenle **out_cikis** çıkış portu değeri değişmemiştir.

5. adımda **process** hassasiyet listesinde bulunan **in_giris_3** giriş portunda değişim meydana geldiğinden **process** aktif hale gelmektedir. **process**'in aktif hale gelmesi ile birlikte **out_cikis** çıkış portuna (**1 and 0**) **or 1** işleminin sonucu atanmaktadır ve **out_cikis** çıkış portu değeri **'1'** olmaktadır.

Tablo 8-2 process_ornek_2 varlığının ilgili girişlere ait ürettiği çıkış değerleri

| | in_giris_1 | in_giris_2 | in_giris_3 | out_cikis |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 |
| 2 | 1 | 0 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 |




Şekil 8-2 process_ornek_2 varlığı benzetim çıktısı

# 8.2. variable Kullanımı

**Örnek 8.3:** Aşağıda **signal** veri nesnesi kullanılarak 8 bitlik giriş portu değerinde bulunan **'1'** bitlerinin sayısını çıkışa aktaran programı gerçeklemeyi amaçlayan **bit_say_signal.vhd** VHDL kodu verilmiştir. Kodda 17. satırda **process** hassasiyet listesinde **in_giris** girişi mevcuttur. Bunun anlamı **in_giris** giriş portunda meydana gelen değişikliklerde **process** aktif hale gelecektir. **process**'in aktif olması ile birlikte 19. satırda tanımlanan varsayılan atama işlemi gerçekleştirlecektir ve **r_sayac** sinyalinin değeri **0** olacaktır. Daha sonra 21. satırda tanımlanan döngü aktif hale gelecektir. Döngünün aktif hale gelmesi ile birlikte **n_i** değişkeni **7**'den **0**'a doğru aşağıya doğru azalacaktır. Azalma işlemini her gerçekleşmesinde 22. satırda tanımlanan kontrol söz dizimi ile birlikte **n_i** değeri için **in_giris** değerinin ilgili **n_i.** bitinin **'1'** olup olmadığının kontrolü yapılmaktadır. Eğer kontrol söz dizimi doğru ise 24. satırda tanımlanan söz dizimi ile birlikte **r_sayac** sinyalinin değerinin 1 arttırılması beklenmektedir. Mimari içerisinde 29. satırda tanımlanan atama ifadesi ile **in_giris** giriş portunda bulunan **'1'** bitlerinin sayısının varlık dışına aktarımının yapılması beklenmektedir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity bit_say_signal is
5.   port(
6.     in_giris : in std_logic_vector(0 to 7);
7.     out_sayac : out integer
8.   );
9. end bit_say_signal;
10.
11.architecture Behavioral of bit_say_signal is
12.
13.  signal r_sayac : integer := 0;
14.
15.begin
16.
17.  process(in_giris)
18.  begin
19.    r_sayac<= 0;
20.
21.    for n_i in 7 downto 0 loop
22.      if in_giris(n_i) = '1' then
23.        r_sayac<= r_sayac + 1;
24.      end if;
25.    end loop;
26.
27.  end process;
28.
```

```
29.  out_sayac <= r_sayac;
30.
31.end Behavioral;
```

**signal** veri nesnesinin özelliği **process**'in sonlanmasıyla birlikte yeni değerini almasıdır. Bu nedenle **in_giris** giriş portunun değerinde **'1'** biti olmaması durumunda sonuç her zaman sıfır olacaktır. Aksi durumda başlangıç değeri sıfır olarak atanmış olan **r_sayac** sinyali **process**'in aktif olması ile birlikte bir arttırılarak devam edecektir. **bit_say_signal** varlığına ait örnek girişlerin ürettiği sonuçlar Tablo 8-3'de verilmiştir ve Şekil 8-3'de benzetim çıktısı gösterilmiştir. Tablo 8-3'de verilen sonuçlarda:

1. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı değeri **0** olan **r_sayac** sinyalinin yeni değeri **1** olmuştur. Fakat bu giriş değeri için **r_sayac** sinyal değerinin **4** olması gerekmektedir. **signal** veri nesnesi **process** sonunda yeni değerini aldığı için sayaç artırım işlemi bir kere yapılmaktadır ve değeri **1** olmaktadır.

2. adımda **in_giriş** giriş portu değerinde bulunan **'1'** bitinden dolayı değeri **1** olan **r_sayac** sinyalinin yeni değeri **2** olmuştur. Fakat bu giriş değeri için **r_sayac** sinyal değerinin **4** olması gerekmektedir. **signal** veri nesnesi **process** sonunda yeni değerini aldığı için sayaç artırım işlemi bir kere yapılmaktadır ve değeri **2** olmaktadır.

3. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı değeri **2** olan **r_sayac** sinyalinin yeni değeri **3** olmuştur. Fakat bu giriş değeri için **r_sayac** sinyal değerinin **6** olması gerekmektedir. **signal** veri nesnesi **process** sonunda yeni değerini aldığı için sayaç artırım işlemi bir kere yapılmaktadır ve değeri **3** olmaktadır.

4. adımda **in_giris** sinyali içerisinde bulunan **'1'** bitinin bulunmamasından dolayı **r_sayac** sinyal değeri **0** olmaktadır. **signal** veri nesnesi **process** sonunda yeni değerini aldığı için 20. satırda tanımlanan atama işlemi gerçekleştirilmiştir.

5. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı değeri **0** olan **r_sayac** sinyalinin yeni değeri **1** olmuştur. Fakat bu giriş değeri için **r_sayac** sinyal değerinin **8** olması gerekmektedir. **signal** veri nesnesi **process** sonunda yeni değerini aldığı için sayaç artırım işlemi bir kere yapılmaktadır ve değeri **1** olmaktadır.

6. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı değeri **1** olan **r_sayac** sinyalinin yeni değeri **2** olmuştur. Fakat bu giriş değeri için **r_sayac** sinyal değerinin **4** olması gerekmektedir. **signal** veri nesnesi **process** sonunda yeni değerini aldığı için sayaç artırım işlemi bir kere yapılmaktadır ve değeri **2** olmaktadır.

7. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinin bulunmamasından dolayı **r_sayac** sinyal değeri **0** olmaktadır. **signal** veri nesnesi **process** sonunda yeni değerini aldığı için 20. satırda tanımlanan atama işlemi gerçekleştirilmiştir.

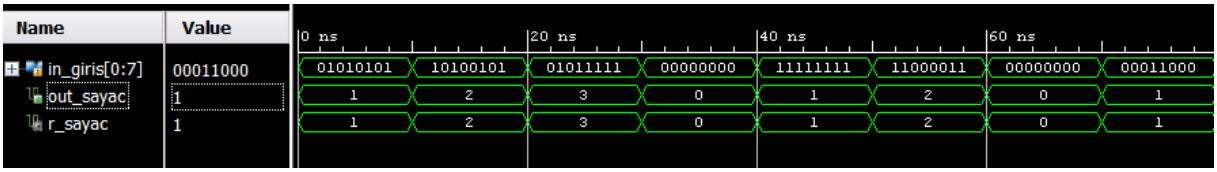
8. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı değeri **0** olan **r_sayac** sinyalinin yeni değeri **1** olmuştur. Fakat bu giriş değeri için **r_sayac** sinyal değerinin **1** olması gerekmektedir. **signal** veri nesnesi **process** sonunda yeni değerini aldığı için sayaç artırım işlemi bir kere yapılmaktadır ve değeri **1** olmaktadır.

Tablo 8-3 ve Şekil 8-3'de gösterilen benzetim sonuçlarından da görüleceği üzere **bit_say_signal.vhd** VHDL kodu ile istenen amaca ulaşılamamıştır.

Tablo 8-3 bit_say_signal varlığının ilgili girişlere ait ürettiği çıkış değerleri

| | in giriş | out_sayac |
|---|---|---|
| 1 | "01010101" | 1 |
| 2 | "10100101" | 2 |
| 3 | "01011111" | 3 |

| | | |
|---|---|---|
| 4 | "00000000" | 0 |
| 5 | "11111111" | 1 |
| 6 | "11000011" | 2 |
| 7 | "00000000" | 0 |
| 8 | "00011000" | 1 |




Şekil 8-3 bit_say_signal varlığı benzetim çıktısı

**process** içerisinde **variable** veri nesnesi tanımlama işlemi **process** ile **process**'e ait **begin** söz dizimi arasında yapılmalıdır. **variable** veri nesnesi tanımlamaya ilişkin söz dizimi aşağıdaki gibidir.

```
process_etiketi:process( sinyal_adi {, sinyal_adi})
    variable degisken_adi : tip_adi;
begin
    ..
    ..
end processprocess_etiketi;
```

**Örnek 8.4 : Örnek 8.3**'de **signal** veri nesnesi kullanılarak giriş portu değerindeki **'1'** bitlerini saymayı amaçlayan **bit_say_signal.vhd** VHDL kodu **variable** veri nesnesi kullanılarak **bit_say_variable.vhd** VHDL kodu aşağıda verilmiştir. Kodda 17. satırda **process** hassasiyet listesinde **in_giris** girişi mevcuttur. Bunun anlamı **in_giris** giriş portunda meydana gelen değişikliklerde **process** aktif hale gelecektir. **process**'in aktif olması ile birlikte 22. satırda tanımlanan varsayılan atama işlemi gerçekleştirlecektir ve **v_sayac** sinyalinin değeri **0** olacaktır. **variable** veri nesnesinde atama işlemleri **:=** operatörü ile yapılmaktadır. Daha sonra 23. satırda tanımlanan döngü aktif hale gelecektir. Döngünün aktif hale gelmesi ile birlikte **n_i** değişkeni **7**'den **0**'a doğru aşağıya doğru azalacaktır. Azalma işlemini her gerçekleşmesinde 24. satırda tanımlanan kontrol söz dizimi ile birlikte **n_i** değeri için **in_giris** değerinin ilgili bitinin **'1'** olup olmadığının kontrolü yapılmaktadır. Eğer kontrol söz dizimi doğru ise 25. satırda tanımlanan söz dizimi ile birlikte **v_sayac** değerinin 1 arttırılması beklenmektedir. Mimari içerisinde 29. satırda tanımlanan atama ifadesi ile sayaç değeri **process** dışına çıkarılmaktadır. 32. satırda tanımlanan ifade ile **in_giris** girişinde bulunan **'1'** bitlerinin sayısının varlık dışına aktarımının yapılması beklenmektedir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity bit_say_variable is
5.   port (
6.     in_giris : in std_logic_vector(0 to 7);
7.     out_sayac : out integer
8.   );
```

```
9. end bit_say_variable;
10.
11.architecture Behavioral of bit_say_variable is
12.
13.  signal r_sayac : integer := 0;
14.
15.begin
16.
17.  process(in_giris)
18.
19.    variable v_sayac : integer := 0;
20.  begin
21.
22.    v_sayac := 0;
23.    for n_i in 7 downto 0 loop
24.      if in_giris(n_i) = '1' then
25.        v_sayac := v_sayac + 1;
26.      end if;
27.    end loop;
28.
29.    r_sayac <= v_sayac;
30.
31.  end process;
32.  out_sayac <= r_sayac;
33.
34.end Behavioral;
```

**variable** veri nesnesinin özelliği işlemleri gecikme olmadan gerçekleştirmesidir. Bu nedenle **in_giris** giriş portu değerinde **'1'** biti olmaması durumunda sonuç her zaman sıfır olacaktır. Aksi durumda başlangıç değeri sıfır olarak atanmış olan **v_sayac variable** değişkenin **process**'in aktif olması ile birlikte **in_giris** girişine ait **'1'** bit değerlerini hızlıca saymaktadır. Aşağıda **bit_say_variable** varlığına ait örnek girişlerin ürettiği sonuçlar Tablo 8-4'de gösterilmiştir verilmiştir ve Şekil 8-4'de benzetim çıktısı gösterilmiştir. Tablo 8-4'de verilen sonuçlarda:

1. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı 23. satırda tanımlanan başlangıç değeri ataması ile **0** olan **v_sayac** değişkeninin yeni değeri **4** olmuştur. **v_sayac** değişken değerinin **r_sayac** sinyaline atanması ile değeri **4** olmaktadır ve bu değer **out_cikis** çıkış portuna aktarılmaktadır.

2. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı 23 satırda tanımlanan başlangıç değeri ataması ile **0** olan **v_sayac** değişkeninin yeni değeri **4** olmuştur. **v_sayac** değişken değerinin **r_sayac** sinyaline atanması ile değeri **4** olmaktadır ve bu değer **out_cikis** çıkış portuna aktarılmaktadır.

3. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı 23. satırda tanımlanan başlangıç değeri ataması ile **0** olan **v_sayac** değişkeninin yeni değeri **6** olmuştur. **v_sayac** değişken

değerinin **r_sayac** sinyaline atanması ile değeri **6** olmaktadır ve bu değer **out_cikis** çıkış portuna aktarılmaktadır.

4. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinin bulunmamasından dolayı 23. satırda tanımlanan başlangıç değeri ataması ile **v_sayac** sinyal değeri **0** olmaktadır. **v_sayac** değişken değerinin **r_sayac** sinyaline atanması ile değeri **0** olmaktadır ve bu değer **out_cikis** çıkış portuna aktarılmaktadır.

5. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı 23. satırda tanımlanan başlangıç değeri ataması ile **0** olan **v_sayac** değişkeninin yeni değeri **8** olmuştur. **v_sayac** değişken değerinin **r_sayac** sinyaline atanması ile değeri **8** olmaktadır ve bu değer **out_cikis** çıkış portuna aktarılmaktadır.

6. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı 23 satırda tanımlanan başlangıç değeri ataması ile **0** olan **v_sayac** değişkeninin yeni değeri **4** olmuştur. **v_sayac** değişken değerinin **r_sayac** sinyaline atanması ile değeri **4** olmaktadır ve bu değer **out_cikis** çıkış portuna aktarılmaktadır.

7. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinin bulunmamasından dolayı 23. satırda tanımlanan başlangıç değeri ataması ile **v_sayac** sinyal değeri **0** olmaktadır. **v_sayac** değişken değerinin **r_sayac** sinyaline atanması ile değeri **0** olmaktadır ve bu değer **out_cikis** çıkış portuna aktarılmaktadır.

8. adımda **in_giris** giriş portu değerinde bulunan **'1'** bitinden dolayı 23. satırda tanımlanan başlangıç değeri ataması ile **0** olan **v_sayac** değişkeninin yeni değeri **2** olmuştur. **v_sayac** değişken değerinin **r_sayac** sinyaline atanması ile değeri **2** olmaktadır ve bu değer **out_cikis** çıkış portuna aktarılmaktadır.

Tablo 8-4 ve Şekil 8-4'de gösterilen benzetim sonuçlarından da görüleceği üzere **bit_say_variable.vhd** VHDL kodu ile istenen amaca ulaşılmıştır.

Tablo 8-4 bit_say_variable varlığının ilgili girişlere ait ürettiği çıkış değerleri

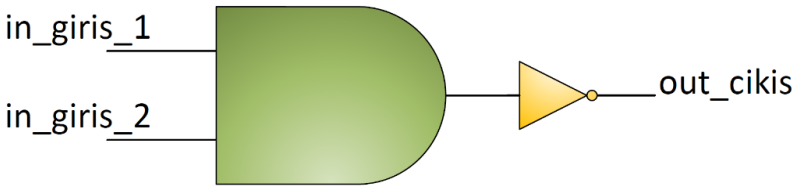
| | in_giris | out_sayac |
|---|---|---|
| 1 | "01010101" | 4 |
| 2 | "10100101" | 4 |
| 3 | "01011111" | 6 |
| 4 | "00000000" | 0 |
| 5 | "11111111" | 8 |
| 6 | "11000011" | 4 |
| 7 | "00000000" | 0 |
| 8 | "00011000" | 2 |


Şekil 8-4 bit_say_variable varlığı benzetim çıktısı

**Örnek 8.3** ve **Örnek 8.4**'te verilen uygulamalarda **signal** ve **variable** veri nesneleri arasındaki kullanım farklılıkları göstermek amaçlanmıştır. Yine bu iki veri nesnesinin kullanım farklılıklarını göstermek amacıyla **VE DEĞİL (NAND)** mantıksal kapı uygulama örnekleri üzerinde durulacaktır.

Şekil 8-5'de **DEĞİL (NAND)** mantık devresi gösterimi verilmiştir. Tablo 8-5'de ise **VE DEĞİL (NAND)** mantık kapısına ilişkin doğruluk tablosu verilmiştir.

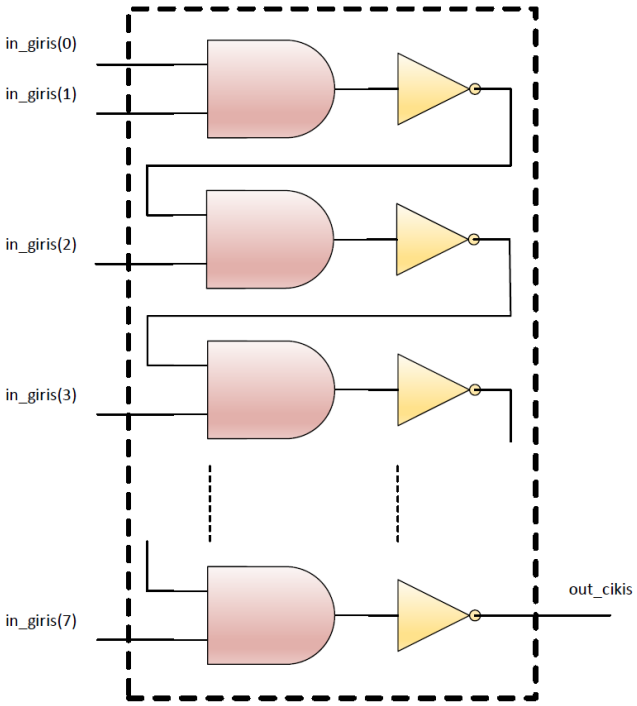
Şekil 8-6'de **VE DEĞİL (NAND)** kapısı kullanılarak gerçeklenmek istenen uygulama blok diyagramı verilmiştir. Şekil 8-6'den de görüleceği üzere **in_giris** portunun **0.** biti ile **1.** biti **VE DEĞİL** işlemine tabi tutulduktan sonra elde edilen sonuç değeri **in_giris** portunun **2.** biti ile **VE DEĞİL** işlemine tabi tutulur. Benzer şekilde işlem sonucu **in_giris** portunun **3.** biti ile **VE DEĞİL** işlemine tabi tutulur. **in_giris** giriş portunun tüm bitleri benzer işlemlere tabi tutularak çıkış değeri elde edilir.




Şekil 8-5 nand kapısı lojik devresi

Tablo 8-5 VE DEĞİL lojik kapısı doğruluk tablosu

| in_giris_1 | in_giris_2 | out_cikis |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




Şekil 8-6 Örnek 7.5 ve Örnek 7.6 içim uygulanacak VE DEĞİL lojik devresi

**Örnek 8.5 :** Şekil 8-6'de verilen **VE DEĞİL** lojik devresi problemini **signal** veri nesnesi kullanarak çözmeyi amaçlayan **nand_kapi_signal.vhd** VHDL kodu aşağıda verilmiştir.

Kodda 19. satırda hassasiyet listesinde **in_giris** giriş portu mevcuttur. Bunun anlamı **in_giris** giriş portunda meydana gelen değişikliklerde **process** aktif hale gelecektir ve 23. satırda tanımlanan atama işlemi gerçekleşmektedir. Daha sonra 24. satırda tanımlanan **n_i** değişkeni ile tanımlanmış döngü aktif hale gelecektir. Döngü **1**'den **7**'e doğru artarken her **n_i** değeri için 25. Satırda tanımlanan **in_giris** değerinin ilgili biti ile **r_nand_sonuc** sinyali ile **VE** işlemine tabi tutulmakta ve elde edilen sonucun tersi tekrar **r_nand_sonuc** sinyaline atanmaktadır. Mimari içerisinde 17. satırda tanımlanan atama ifadesi ile **in_giris** giriş portu değerinde bulunan bitlerinin **VE DEĞİL** işlemi sonucunun varlık dışına aktarılması beklenmektedir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity nand_kapi_signal is
5.   port(
6.     in_giris : in std_logic_vector(7 downto 0);
7.     out_cikis : out std_logic
8.   );
9. end nand_kapi_signal;
10.
11.architecture Behavioral of nand_kapi_signal is
12.
13.  signal r_nand_sonuc : std_logic := '0';
14.
15.begin
16.
17.  out_cikis <= r_nand_sonuc;
18.
19.  process(in_giris)
20.    variable v_nand_sonuc : std_logic := '0';
21.
22.  begin
23.    r_nand_sonuc <= in_giris(0);
24.    for n_i in 1 to 7 loop
25.      r_nand_sonuc <= not(r_nand_sonuc and in_giris(n_i));
26.    end loop;
27.
28.  end process;
29.end Behavioral;
```

**process**'in aktif olması ile birlikte **signal** veri nesnesinin değerini **process**'in bitimi ile almasından dolayı **r_nand_sonuc** sinyalinin yeni değeri **r_nand_sonuc <= not(r_nand_sonuc and**

**in_giris(7))** söz dizimi ifadesi ile elde edilmektedir. **in_giris(7)** bit değeri Tablo 8-6'da kırmızı renk ile gösterilmiştir.

Tablo 8-6'da **nand_kapi_signal** varlığına ait örnek girişlerin ürettiği sonuçlar gösterilmiştir. Şekil 8-7'de benzetim çıktısı verilmiştir. Tablo 8-6'da verilen sonuçlarda:

1. adımda **signal** veri nesnesinin değerini **process**'in sonunda almasından dolayı **in_giris** sinyalinin en anlamlı biti ile **r_nand_sonuc** sinyali **VE** işlemi sonucunun tersi tekrar **r_nand_sonuc** sinyaline atanmaktadır ve bu değer **'1'** olmaktadır. **r_nand_sonuc** sinyalinin **out_cikis** portuna atanmasıyla sonuç varlık dışına aktarılmaktadır.

2. adımda **signal** veri nesnesinin değerini **process**'in sonunda almasından dolayı **in_giris** sinyalinin en anlamlı biti ile 1. adımda elde edilen **r_nand_sonuc** sinyali **VE** işlemi sonucunun tersi tekrar **r_nand_sonuc** sinyaline atanmaktadır ve bu değer **'1'** olmaktadır. **r_nand_sonuc** sinyalinin **out_cikis** portuna atanmasıyla sonuç varlık dışına aktarılmaktadır.

3. adımda **signal** veri nesnesinin değerini **process**'in sonunda almasından dolayı **in_giris** sinyalinin en anlamlı biti ile 2 adımda elde edilen **r_nand_sonuc** sinyali **VE** işlemi sonucunun tersi tekrar **r_nand_sonuc** sinyaline atanmaktadır ve bu değer **'0'** olmaktadır. **r_nand_sonuc** sinyalinin **out_cikis** portuna atanmasıyla sonuç varlık dışına aktarılmaktadır.

4. adımda **signal** veri nesnesinin değerini **process**'in sonunda almasından dolayı **in_giris** sinyalinin en anlamlı biti ile 3. adımda elde edilen **r_nand_sonuc** sinyali **VE** işlemi sonucunun tersi tekrar **r_nand_sonuc** sinyaline atanmaktadır ve bu değer **'1'** olmaktadır. **r_nand_sonuc** sinyalinin **out_cikis** portuna atanmasıyla sonuç varlık dışına aktarılmaktadır.

5. adımda **signal** veri nesnesinin değerini **process**'in sonunda almasından dolayı **in_giris** sinyalinin en anlamlı biti ile 4. adımda elde edilen **r_nand_sonuc** sinyali **VE** işlemi sonucunun tersi tekrar **r_nand_sonuc** sinyaline atanmaktadır ve bu değer **'1'** olmaktadır. **r_nand_sonuc** sinyalinin **out_cikis** portuna atanmasıyla sonuç varlık dışına aktarılmaktadır.

6. adımda **signal** veri nesnesinin değerini **process**'in sonunda almasından dolayı **in_giris** sinyalinin en anlamlı biti ile 5. adımda elde edilen **r_nand_sonuc** sinyali **VE** işlemi sonucunun tersi tekrar **r_nand_sonuc** sinyaline atanmaktadır ve bu değer **'0'** olmaktadır. **r_nand_sonuc** sinyalinin **out_cikis** portuna atanmasıyla sonuç varlık dışına aktarılmaktadır.

Tablo 8-6 nand_kapi_signal varlığının ilgili girişlere ait ürettiği çıkış değerleri

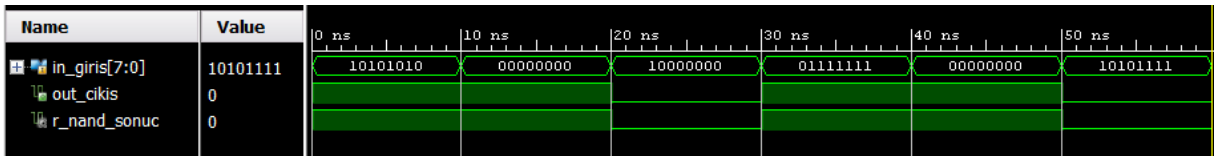
| | in_giris | | out_cikis |
|---|---|---|---|
| 1 | "10101010" | not('0' and '1') | 1 |
| 2 | "00000000" | not('1' and '0') | 1 |
| 3 | "10000000" | not('1' and '1') | 0 |
| 4 | "01111111" | not('0' and '0') | 1 |
| 5 | "00000000" | not('1' and '0') | 1 |
| 6 | "10101111" | not('1' and '1') | 0 |

1. adımda verilen giriş değeri için döngü içerisinde her adımda **r_nand_sonuc** sinyalinin alacağı değer ve **out_cikis** çıkış portuna aktarılacak değerler aşağıdaki gibi elde edilmektedir.

```
1. r_nand_sonuc <= in_giris(0) <= '0';
2. r_nand_sonuc <= not(r_nand_sonuc and in_giris(1))
                <= not('0' and '1') <= '1';
3. r_nand_sonuc <= not(r_nand_sonuc and in_giris(2))
```

```
                <= not('1' and '0') <= '1';
4. r_nand_sonuc <= not(r_nand_sonuc and in_giris(3))
                <= not('1' and '1') <= '0';
5. r_nand_sonuc <= not(r_nand_sonuc and in_giris(4))
                <= not('0' and '0') <= '1';
6. r_nand_sonuc <= not(r_nand_sonuc and in_giris(5))
                <= not('1' and '1') <= '0';
7. r_nand_sonuc <= not(r_nand_sonuc and in_giris(6))
                <= not('0' and '0') <= '1';
8. r_nand_sonuc <= not(r_nand_sonuc and in_giris(7))
                <= not('1' and '1') <= '0';
out_cikis <= r_nand_sonuc<= '0';
```

Yukarıda yapılan açıklama ile birlikte Tablo 8-6' ve Şekil 8-7'de gösterilen benzetim sonuçlarından da görüleceği üzere **nand_kapi_signal.vhd** VHDL kodu ile istenen amaca ulaşılamamıştır.




Şekil 8-7 nand_kapi_signal varlığı benzetim çıktısı

**Örnek 8.6 :** Şekil 8-6'de verilen **VE DEĞİL** lojik devresi problemini **variable** veri nesnesi kullanarak çözmeyi amaçlayan **nand_kapi_variable.vhd** VHDL kodu aşağıda verilmiştir.

Kodda 19. satırda hassasiyet listesinde **in_giris** girişi mevcuttur. Bunun anlamı **in_giris** değerinde meydana gelen değişikliklerde **process** aktif hale gelecektir ve 23. satırda tanımlanan atama işlemi gerçekleşmektedir. Daha sonra 24. satırda tanımlanan **n_i** değişkeni ile tanımlanmış döngü aktif hale gelecektir. Döngü **1**'den **7**'e doğru artarken her **n_i** değeri için 25. satırda tanımlanan **in_giris** değerinin ilgili biti ile **v_nand_sonuc** değişkenini ile **VE** işlemine tabi tutulmakta ve elde edilen sonucun tersi tekrar **v_nand_sonuc** değişkenine atanmaktadır. 27. satırda tanımlanan atama ifadesi ile birlikte **v_nand_sonuc** değişkeninde tutulan sonuç değeri **process** dışına aktarılır. Mimari içerisinde 17. satırda tanımlanan atama ifadesi ile **in_giris** girişinde bulunan bitlerinin **VE DEĞİL** işlemi sonucunun varlık dışına aktarılması beklenmektedir.

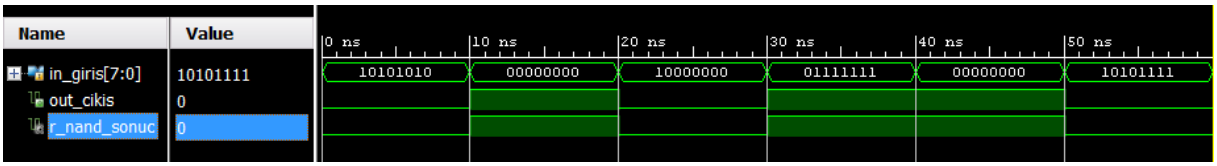
```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity nand_kapi_variable is
5.   port (
6.     in_giris : in std_logic_vector(7 downto 0);
7.     out_cikis : out std_logic
8.   );
9. end nand_kapi_variable;
```

```
10.
11.architecture Behavioral of nand_kapi_variable is
12.
13.  signal r_nand_sonuc : std_logic := '0';
14.
15.begin
16.
17.  out_cikis <= r_nand_sonuc;
18.
19.  process(in_giris)
20.
21.    variable v_nand_sonuc : std_logic := '0';
22.  begin
23.    v_nand_sonuc := in_giris(0);
24.    for n_i in 1 to 7 loop
25.      v_nand_sonuc := not(v_nand_sonuc and in_giris(n_i));
26.    end loop;
27.    r_nand_sonuc <= v_nand_sonuc;
28.
29.  end process;
30.end Behavioral;
```

**variable** veri nesnesinin yeni değerini gecikme olmadan alması nedeniyle **nand_kapi_variable.vhd** VHDL kodu doğru sonuç üretmektedir. Tablo 8-7'de uygulamaya ait örnek girişlere ait sonuçlar verilmiştir. Şekil 8-8'da **nand_kapi_variable** varlığına ait benzetim çıktısı gösterilmektedir. Tablo 8-7 ve Şekil 8-8'dan da görüleceği üzere **variable** veri nesnesi kullanılarak istenen amaca ulaşılmıştır.

Tablo 8-7 nand_kapi_variable varlığının ilgili girişlere ait ürettiği çıkış değerleri

| | in_giris | out_cikis |
|---|---|---|
| 1 | "10101010" | 0 |
| 2 | "00000000" | 1 |
| 3 | "10000000" | 0 |
| 4 | "01111111" | 1 |
| 5 | "00000000" | 1 |
| 6 | "10101111" | 0 |



Şekil 8-8 nand_kapi_variable varlığı benzetim çıktısı

## 8.3. if Sözdizimi

VHDL dilinde **if** sözdizimi, bir veya birden fazla koşula bağlı olarak, koşulların bağlı olduğu sözdizimlerinden birini seçmektedir ve seçilen koşula ait söz dizim ifadesi gerçeklenmektedir. Eğer koşul ifadelerinin hiç biri sağlanmamış ise bu durumda hiçbir işlem yapılmamaktadır. Aşağıda VHDL dilinde **if** sözdizimi tanımı verilmiştir. Verilen sözdiziminde **if** söz dizimi ile ilgili koşul denetlenmektedir. Eğer bu koşul sağlanırsa koşula bağlı sözdizimleri gerçeklenir. Aksi durumda ise **elsif** söz dizimi ile ilgili koşul denetlenmektedir ve koşul sağlanırsa koşula bağlı sözdizimleri gerçeklenir. Aksi durumda ise **else** söz dizimi ile ilgili koşullar gerçeklenmektedir.

```
if koşul then
    sözdizimi;
    {sözdizimi;}
elsif koşul then
    Sözdizimi;
    {sözdizimi;}
else
    sözdizimi;
    {sözdizimi;}
end if ;
```

**Örnek 8.7:** Aşağıda verilen **if_ornek_process.vhd** VHDL kodunda **if** sözdizimi **process** içerisinde kullanılmıştır. **out_cikis** çıkış portunun alacağı değer 3 farklı koşulda belirlenmektedir. Eğer **in_giris_secme** giriş port değeri **'0'** ise **out_cikis** çıkış portuna **in_giris_1** giriş portunun değeri atanmaktadır. Eğer **in_giris_secme** giriş port değeri **'1'** ise **out_cikis** çıkış portuna **in_giris_2** giriş portunun değeri atanmaktadır. **in_giris_secme** giriş port değeri **'0'** ve **'1'** haricinde başka değerler alması durumunda ise **out_cikis** çıkış portuna **'0'** değeri atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity if_ornek_process is
5.   Port (
6.     in_giris_1 : in std_logic;
7.     in_giris_2 : in std_logic;
8.     in_giris_secme : in std_logic;
a.     out_cikis : out std_logic
9.   );
10.end if_ornek_process;
11.
```

```
12.architecture Behavioral of if_ornek_process is
13.
14.begin
15.
16.  process(in_giris_1, in_giris_2, in_giris_secme)
17.  begin
18.
19.    if in_giris_secme = '0' then
20.      out_cikis <= in_giris_1;
21.    elsif in_giris_secme = '1' then
22.      out_cikis <= in_giris_2;
23.    else
24.      out_cikis <= '0';
25.    end if;
26.
27.  end process;
28.end Behavioral;
```

**Örnek 8.8 :** Aşağıda verilen **if_ornek_function.vhd** VHDL kodunda **if** sözdizimi **function** içerisinde kullanılmıştır. **seçme_if** fonksiyonun döndürdüğü değer **out_cikis** çıkış portuna atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity if_ornek_function is
5.   Port (
6.     in_giris_1 : in std_logic;
7.     in_giris_2 : in std_logic;
8.     in_giris_secme : in std_logic;
9.     out_cikis : out std_logic
10.  );
11.end if_ornek_function;
12.
13.architecture Behavioral of if_ornek_function is
14.
15.  function secme_if(sinyal_1, sinyal_2, sinyal_secme : std_logic)
   return std_logic is
16.
17.    variable sinyal_cikis : std_logic;
18.
19.  begin
```

```
20.     if sinyal_secme = '0' then
21.       sinyal_cikis := sinyal_1;
22.     elsif sinyal_secme = '1' then
23.       sinyal_cikis := sinyal_2;
24.     else
25.       sinyal_cikis := '0';
26.     end if;
27.     return sinyal_cikis;
28.   end secme_if;
29.
30.begin
31.
32.   out_cikis <= secme_if(in_giris_1, in_giris_2, in_giris_secme);
33.
34.end Behavioral;
```

# 8.4. case Sözdizimi

Aşağıda VHDL dilinde **case** sözdizimi tanımı verilmiştir. Verilen sözdiziminde her **when** ifadesine bağlı sabit değerlerin **case** sözdizimidne tanımlanmış ifade değerine eşit olması durumunda, o sabit değere ilişkin sözdizimleri gerçklenmektedir.

```
case ifade is
    when sabit_deger =>
        Sözdizimi;
        {Sözdizimi;}
    when sabit_deger=>
        Sözdizimi;
        {Sözdizimi;}
        ..
        ..
    when others =>
       Sözdizimi;
       {Sözdizimi;}
    end case ;
```

**Örnek 8.9 :** Aşağıda verilen **case_ornek_process.vhd** VHDL kodunda **case** sözdizimi **process** içerisinde kullanılmıştır. **out_cikis** çıkış portunun alacağı değer 3 farklı koşulda belirlenmektedir. Eğer **in_giris_secme** giriş port değeri **'0'** ise **out_cikis** çıkış portuna **in_giris_1** giriş portunun değeri atanmaktadır. Eğer **in_giris_secme** giriş port değeri **'1'** ise **out_cikis** çıkış portuna **in_giris_2**

giriş portunun değeri atanmaktadır. **in_giris_secme** giriş port değeri **'0'** ve **'1'** haricinde başka değerler alması durumunda ise **out_cikis** çıkış portuna **'0'** değeri atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity case_ornek_process is
5.   Port (
6.     in_giris_1 : in std_logic;
7.     in_giris_2 : in std_logic;
8.     in_giris_secme : in std_logic;
9.     out_cikis : out std_logic
10.  );
11.end case_ornek_process;
12.
13.architecture Behavioral of case_ornek_process is
14.
15.begin
16.  process(in_giris_1, in_giris_2, in_giris_secme)
17.  begin
18.    case in_giris_secme is
19.      when '0' =>
20.        out_cikis <= in_giris_1;
21.
22.      when '1' =>
23.        out_cikis <= in_giris_2;
24.
25.      when others =>
26.        out_cikis <= '0';
27.    end case;
28.  end process;
29.end Behavioral;
```

**Örnek 8.10 :** Aşağıda verilen **case_ornek_function.vhd** VHDL kodunda **if** sözdizimi **function** içerisinde kullanılmıştır. **seçme_case** fonksiyonun döndürdüğü değer **out_cikis** çıkış portuna atanmaktadır.

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

```
5.   Port (
6.     in_giris_1 : in std_logic;
7.     in_giris_2 : in std_logic;
8.     in_giris_secme : in std_logic;
9.     out_cikis : out std_logic
10.  );
11.end case_ornek_function;
12.
13.architecture Behavioral of case_ornek_function is
14.
15.  function secme_case(sinyal_1, sinyal_2, sinyal_secme : std_logic)
   return std_logic is
16.
17.    variable sinyal_cikis : std_logic;
18.
19.  begin
20.
21.    case sinyal_secme is
22.      when '0' =>
23.        sinyal_cikis := sinyal_1;
24.
25.      when '1' =>
26.        sinyal_cikis := sinyal_2;
27.
28.      when others =>
29.        sinyal_cikis := '0';
30.    end case;
31.    return sinyal_cikis;
32.  end secme_case;
33.
34.begin
35.
36.  out_cikis <= secme_case(in_giris_1, in_giris_2, in_giris_secme);
37.
38.end Behavioral;
```

**Örnek 8.11:** Aşağıda verilen **case_ornek_type.vhd** VHDL kodunda **case** sözdizimi **process** içerisinde kullanılmıştır. **case** sözdiziminde kullanılacak olan sabit değer ifadeleri **t_MATH_ISLEM** tipinde tanımlanmış olan **TOPLA** ve **CIKAR** ifadeleridir. **r_MATH_ISLEM** sinyali de **t_MATH_ISLEM** tipinde sinyal olarak tanımlanmıştır. **case r_MATH_ISLEM is** söz dizimi ile **r_MATH_ISLEM** sinyalinin **TOPLA** değerine eşit olması durumunda **out_cikis** çıkış portuna **in_giris_1** ve **in_giris_2** giriş port değerlerinin toplamı atanmaktadır. **r_MATH_ISLEM** sinyalinin **CIKAR** değerine eşit olması durumunda **out_cikis** çıkış

portuna **in_giris_1** ve **in_giris_2** giriş port değerlerinin farkı atanmaktadır. **r_MATH_ISLEM** sinyalinin alacağı diğer durumlarda ise **out_cikis** değerinin tüm bitlerine **'0'** atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use IEEE.STD_LOGIC_SIGNED.ALL;
4.
5. entity case_ornek_type 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 : out std_logic_vector(3 downto 0)
10.  );
11.end case_ornek_type;
12.
13.architecture Behavioral of case_ornek_type is
14.
15.  type t_MATH_ISLEM is (TOPLA, CIKAR);
16.  signal r_MATH_ISLEM : t_MATH_ISLEM := TOPLA;
17.
18.begin
19.
20.  process(in_giris_1, in_giris_2)
21.  begin
22.    case r_MATH_ISLEM is
23.      when TOPLA =>
24.        r_MATH_ISLEM <= CIKAR;
25.        out_cikis <= in_giris_1 + in_giris_2;
26.
27.      when CIKAR =>
28.        r_MATH_ISLEM <= TOPLA;
29.        out_cikis <= in_giris_1 - in_giris_2;
30.
31.      when others =>
32.        out_cikis <= (others => '0');
33.    end case;
34.  end process;
35.end Behavioral;
```

# 8.5. loop Sözdizimi

Döngü deyimi tekrarlı sıralı sözdizimlerinden meydana gelmektedir. Aynı zamanda söz dizimi ardışık tekrar veya iterasyon numarasının atanması için koşulu listeler. VHDL iki tip döngü tipini desteklemektedir:

- **for- loop**
- **while- loop**

**for loop** döngü sözdizimi için genel yazım formu aşağıda verilmiştir.

```
donguetiketi : for degisken_adi in aralik loop
    sözdizimi;
    {sözdizimi;}
 end loop donguetiketi ;
```

**Örnek 8.12 :** Aşağıda **for_ornek_process.vhd** VHDL kodunda **process** içerisinde **for loop** sözdizimi kullanımı örneği verilmiştir. **process** hassasiyet listesinde **in_giris** sinyali mevcuttur. Bu durumda **in_giris** sinyalinde meydana gelecek olan değişimlerde **process** aktif hale gelecektir. 25. satırda **process** içerisinde **v_sinyal_sonuc** değişkeninin 0. bitine **'0'** atanmaktadır. 27. satırda **n_i** değişkeni **0**'dan **7**'e kadar artış gösterecek şekilde **for loop** döngüsü tanımlanmaktadır. Döngü içerisinde **v_sinyal_sonuc** değişkeninin ve **in_giris** giriş portunun **n_i.** bitleri **or** işlemine tabi tutulduktan sonra **v_sinyal_sonuc** değişkeninin **n_i+1.** bitine atanmaktadır. 31. Satırda tanımlı atama işlemi ile **v_sinyal_sonuc** değişkeninin ilk 8 biti **r_sinyal_sonuc** sinyaline atanmaktadır. 17. satırda tanımlı atama ifadesi ile **r_sinyal_sonuc** sinyali **out_cikis** çıkış portuna aktarılmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity for_ornek_process is
5.   Port(
6.     in_giris : in std_logic_vector(7 downto 0);
7.     out_cikis : out std_logic_vector(7 downto 0)
8.   );
9. end for_ornek_process;
10.
11.architecture Behavioral of for_ornek_process is
12.
13.  signal r_sinyal_sonuc : std_logic_vector(7 downto 0);
14.
15.begin
16.
17.  out_cikis <= r_sinyal_sonuc;
```

```
18.
19.  process(in_giris)
20.
21.    variable v_sinyal_sonuc : std_logic_vector(8 downto 0);
22.
23.  begin
24.
25.    v_sinyal_sonuc(0) := '0';
26.
27.    for n_i in 0 to 7 loop
28.      v_sinyal_sonuc   (n_i   +   1)   :=   v_sinyal_sonuc   (n_i)   or
   in_giris(n_i);
29.    end loop;
30.
31.    r_sinyal_sonuc <= v_sinyal_sonuc(8 downto 1);
32.
33.  end process;
34.
35.end Behavioral;
```

**Örnek 8.13 :** Aşağıda verilen **for_ornek_function.vhd** VHDL kodunda **for loop** sözdizimi **function** içerisinde kullanılmıştır. **fonksiyon_for** fonksiyonun döndürdüğü değer **out_cikis** çıkış portuna atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity for_ornek_function is
5.   Port (
6.     in_giris : in std_logic_vector(7 downto 0);
7.     out_cikis : out std_logic_vector(7 downto 0)
8.   );
9. end for_ornek_function;
10.
11.architecture Behavioral of for_ornek_function is
12.
13.  function fonksiyon_for(sinyal_giris : std_logic_vector(7 downto 0))
   return std_logic_vector is
14.
15.    variable sinyal_cikis : std_logic_vector(8 downto 0);
16.
17.  begin
```

```
18.
19.     sinyal_cikis(0) := '0';
20.     for n_i in 0 to 7 loop
21.       sinyal_cikis(n_i     +     1)     :=     sinyal_cikis(n_i)     or
   sinyal_giris(n_i);
22.     end loop;
23.     return sinyal_cikis(8 downto 1);
24.   end fonksiyon_for;
25.
26.begin
27.
28.   out_cikis <= fonksiyon_for(in_giris);
29.
30.end Behavioral;
```

**while loop** döngü sözdizimi için genel yazım formu aşağıda verilmiştir.

```
donguetiketi : while kosul loop
    sözdizimi;
    {sözdizimi;}
end loop donguetiketi ;
```

**Örnek 8.14 :** Aşağıda **while_ornek_process.vhd** VHDL kodunda **process** içerisinde **for loop** sözdizimi kullanımı örneği verilmiştir. **process** hassasiyet listesinde **in_giris** sinyali mevcuttur. Bu durumda **in_giris** sinyalinde meydana gelecek olan değişimlerde **process** aktif hale gelecektir. 27. satırda **process** içerisinde **v_sinyal_sonuc** değişkeninin 0. bitine **'0'** atanmaktadır. 28. satırda **n_i** değişkeninin **8**'den küçük olma koşulu ile **while loop** döngüsü tanımlanmaktadır. Döngü içerisinde **v_sinyal_sonuc** değişkeninin ve **in_giris** giriş portunun **n_i.** bitleri **or** işlemine tabi tutulduktan sonra **v_sinyal_sonuc** değişkeninin **n_i+1.** bitine atanmaktadır. 33. satırda tanımlı atama işlemi ile **v_sinyal_sonuc** değişkeninin ilk 8 biti **r_sinyal_sonuc** sinyaline atanmaktadır. 17. satırda tanımlı atama ifadesi ile **r_sinyal_sonuc** sinyali **out_cikis** çıkış portuna aktarılmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity while_ornek_process is
5.   Port (
6.     in_giris : in std_logic_vector(7 downto 0);
7.     out_cikis : out std_logic_vector(7 downto 0)
8.   );
9. end while_ornek_process;
10.
11.architecture Behavioral of while_ornek_process is
```

```
12.
13.  signal r_sinyal_sonuc : std_logic_vector(7 downto 0);
14.
15.begin
16.
17.  out_cikis <= r_sinyal_sonuc;
18.
19.  process(in_giris)
20.
21.    variable v_sinyal_sonuc : std_logic_vector(8 downto 0);
22.    variable n_i : integer := 0;
23.
24.  begin
25.
26.    n_i := 0;
27.    v_sinyal_sonuc(0) := '0';
28.    while n_i < 8 loop
29.      v_sinyal_sonuc(n_i + 1) := v_sinyal_sonuc(n_i) or
   in_giris(n_i);
30.      n_i := n_i + 1;
31.    end loop;
32.
33.    r_sinyal_sonuc <= v_sinyal_sonuc(8 downto 1);
34.
35.  end process;
36.end Behavioral;
```

**Örnek 8.15 :** Aşağıda verilen **while_ornek_function.vhd** VHDL kodunda **while loop** sözdizimi **function** içerisinde kullanılmıştır. **fonksiyon_while** fonksiyonun döndürdüğü değer **out_cikis** çıkış portuna atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity while_ornek_function is
5.   Port (
6.     in_giris : in std_logic_vector(7 downto 0);
7.     out_cikis : out std_logic_vector(7 downto 0)
8.   );
9. end while_ornek_function;
10.
```

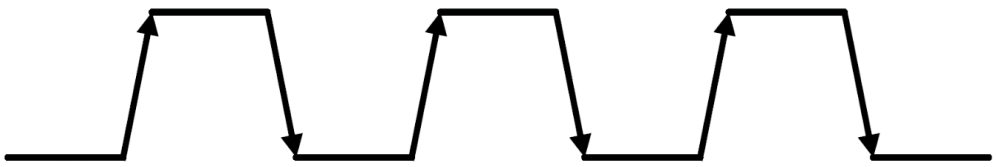
```
11.architecture Behavioral of while_ornek_function is
12.
13.  function fonksiyon_while(sinyal_giris : std_logic_vector(7 downto
   0)) return std_logic_vector is
14.    variable sinyal_cikis : std_logic_vector(8 downto 0);
15.    variable n_i : integer := 0;
16.  begin
17.    sinyal_cikis(0) := '0';
18.    while n_i < 8 loop
19.      sinyal_cikis(n_i + 1) := sinyal_cikis(n_i) or
   sinyal_giris(n_i);
20.      n_i := n_i + 1;
21.    end loop;
22.    return sinyal_cikis(8 downto 1);
23.  end fonksiyon_while;
24.
25.begin
26.
27.  out_cikis <= fonksiyon_while(in_giris);
28.
29.end Behavioral;
```

## 8.6. Saat Darbesi Kullanımı

Genelde **process** yapılarını tetikleme için harici bir kaynak tarafından üretilen kare dalga işareti kullanılmaktadır. Bu kare dalga işaretinin değişimlerinde ise **process** yapısı tetiklenmektedir. Bu dışarıdan uygulanan (tasarıma göre FPGA içerisinde de üretilebilir) tetikleme işareti çoğu zaman "saat darbesi" (clock) olarak adlandırılır.

Şekil 8-9'da saat darbesi gösterimi verilmiştir. Şekil 8-9'dan da görüleceği üzere saat darbesi işaretin 0'dan 1'e veya 1'den 0'a değişim göstermektedir. Bu değişimler iki şekilde adlandırılır:

- Yükselen Kenar Tetikleme (Rising Edge Triggered)
- Düşen Kenar Tetikleme (Falling Edge Triggered)'dir.



Şekil 8-9 Saat darbesi gösterimi

VHDL dilinde yükselen kenar durumunu tespiti için iki tip sözdizimi mevcuttur:

- `if in_clk'event and in_clk='1' then`
- `if rising_edge(in_clk) then`

Yukarıda verilen sözdizimleri **in_clk** sinyalinde değişim meydana geldiğinde ve **in_clk** değerinin 0'dan 1'e değiştiğinde koşul ifadelerini aktif etmektedir.

VHDL dilinde düşen kenar durumunu tespiti için iki tip sözdizimi mevcuttur:

- `if in_clk'event and in_clk='0' then`
- `if falling_edge(in_clk) then`

Yukarıda verilen sözdizimleri **in_clk** sinyalinde değişim meydana geldiğinde ve **in_clk** değerinin 1'den 0'a değiştiğinde koşul ifadelerini aktif etmektedir.

**Örnek 8.16 :** Aşağıda **saat_darbesi_ornek.vhd** VHDL kodunda saat darbesi uygulaması gerçeklenmiştir. **process** hassasiyet listesinde **in_clk** giriş portu mevcuttur. 20. satırda tanımlanan koşul ifadesi ile **in_clk** giriş portunda değişim meydana geldiğinde ve **in_clk** giriş portu değeri 0'dan 1'e değiştiğinde: **in_giris_secme** giriş port değeri **'0'** ise **out_cikis** çıkış portuna **in_giris_1** giriş portunun değeri atanmaktadır. Eğer **in_giris_secme** giriş port değeri **'1'** ise **out_cikis** çıkış portuna **in_giris_2** giriş portunun değeri atanmaktadır. **in_giris_secme** giriş port değeri **'0'** ve **'1'** haricinde başka değerler alması durumunda ise **out_cikis** çıkış portuna **'0'** değeri atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity saat_darbesi_ornek is
5.   Port (
6.     in_clk : in std_logic;
7.     in_giris_1 : in std_logic;
8.     in_giris_2 : in std_logic;
9.     in_giris_secme : in std_logic;
10.    out_cikis : out std_logic
11.  );
12.end saat_darbesi_ornek;
13.
14.architecture Behavioral of saat_darbesi_ornek is
15.begin
16.
17.  process(in_clk)
18.  begin
19.
20.    if rising_edge(in_clk) then
21.      if in_giris_secme = '0' then
22.        out_cikis <= in_giris_1;
```

```
23.       elsif in_giris_secme = '1' then
24.         out_cikis <= in_giris_2;
25.       else
26.         out_cikis <= '0';
27.       end if;
28.     end if;
29.   end process;
30.end Behavioral;
```

Bu noktada önemli bir uyarı yapmak gerekmektedir. Bir **process** içerisinden aynı anda hem yükselen hem de düşen kenar kontrolü **yapılamaz.** Bu şekilde bir kod yazıldığında sentezleyici hata verecektir.

# 8.7. WAIT UNTIL Söz dizimi

Aşağıda VHDL dilinde **wait until** sözdizimi tanımı verilmiştir:

```
wait until koşul;
```

**process** içerisinde **wait until** söz dizimi kullanımı özel bir durumudur. Çünkü **wait until** kullanım durumunda hassasiyet listesi ihmal edilir.

**Örnek 8.17:** Aşağıda verilen **wait_until_ornek.vhd** VHDL kodunda saat darbesi uygulaması gerçeklenmiştir. 20. satırda tanımlanan sözdizimi ile **in_clk** giriş portunda değişim meydana geldiğinde ve **in_clk** giriş portu değeri 0'dan 1'e değiştiğinde **out_cikis** çıkış portunun alacağı değer 3 farklı koşulda belirlenmektedir. Eğer **in_giris_secme** giriş port değeri **'0'** ise **out_cikis** çıkış portuna **in_giris_1** giriş portunun değeri atanmaktadır. Eğer **in_giris_secme** giriş port değeri **'1'** ise **out_cikis** çıkış portuna **in_giris_2** giriş portunun değeri atanmaktadır. **in_giris_secme** giriş port değeri **'0'** ve **'1'** haricinde başka değerler alması durumunda ise **out_cikis** çıkış portuna **'0'** değeri atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity wait_until_ornek is
5.   Port (
6.     in_clk : in std_logic;
7.     in_giris_1 : in std_logic;
8.     in_giris_2 : in std_logic;
9.     in_giris_secme : in std_logic;
10.    out_cikis : out std_logic
11.  );
12.end wait_until_ornek;
13.
```

```
14.architecture Behavioral of wait_until_ornek is
15.
16.begin
17.
18.  process
19.  begin
20.
21.    wait until in_clk'event and in_clk = '1';
22.
23.      case in_giris_secme is
24.        when '0' =>
25.          out_cikis <= in_giris_1;
26.        when '1' =>
27.          out_cikis <= in_giris_2;
28.        when others =>
29.          out_cikis <= '0';
30.      end case;
31.
32.  end process;
33.
34.end Behavioral;
```

## 8.8. İfade Düzenleme

Aşağıda verilen kodda, **sinyal_secme** sinyalinin **'1'** olma durumunda **siynal_cikis** sinyaline **sinyal_2** sinyali atanmakta aksi durumda ise **sinyal_1** sinyali atanmaktadır. **sinyal_secme**, **sinyal_1** ve **sinyal_2** sinyallerinin herhangi birinde meydan gelen değişimde **process** aktif olmakta ve koşul işlemektedir.

```
..
..
signal sinyal_1 : std_logic;
signal sinyal_2 : std_logic;
signal sinyal_secme : std_logic;
signal sinyal_sonuc : std_logic;
..
..
process_etiketi:process(sinyal_secme, sinyal_1, sinyal_2)
begin
    if sinyal_secme = '1' then
        sinyal_cikis<= sinyal_2;
```

```
    else
        sinyal_cikis<= sinyal_1;
    end if;
end process process_etiketi;
..
..
```

VHDL dilinde **process** içerisinde işlemlerin sıralı olarak yapılmasından dolayı yukarıda verilen uygulama aşağıdaki gibide ifade edilebilmektedir. **sinyal_secme**, **sinyal_1** ve **sinyal_2**sinyallerinin herhangi birinde meydan gelen değişimle **process** aktif olmakta ve **sinyal_cikis** sinyaline **sinyal_1** sinyali atanmaktadır. Eğer **sinyal_secme** sinyali **'1'** ise **sinyal_cikis** sinyaline **sinyal_2** sinyali atanmaktadır.

```
..
..
signal sinyal_1 : std_logic;
signal sinyal_2 : std_logic;
signal sinyal_secme : std_logic;
signal sinyal_sonuc : std_logic;
..
..
process_etiketi:process(sinyal_secme, sinyal_1, sinyal_2)
begin
    sinyal_cikis<= sinyal_1;
    if sinyal_secme = '1' then
        sinyal_cikis<= sinyal_2;
    end if;
end process process_etiketi;
..
..
```