# 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.

## 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 portlarındanbiri değişim gösterdiği zaman **process** aktif hale gelecektir.

1. **library** IEEE;
2. **use** IEEE.STD\_LOGIC\_1164.ALL;
3. **entity** process\_ornek\_1 **is**
4. **Port** (
5. in\_giris\_1 : **in** std\_logic;
6. in\_giris\_2 : **in** std\_logic;
7. in\_giris\_3 : **in** std\_logic;
8. out\_sonuc : **out** std\_logic
9. );
10. **end** process\_ornek\_1;
11. **architecture** Behavioral **of** process\_ornek\_1 **is**
12. **begin**
13. process\_etiketi:**process**(in\_giris\_1, in\_giris\_2, in\_giris\_3)
14. **begin**
15. out\_cikis <= (in\_giris\_1 **and** in\_giris\_2) **or** in\_giris\_3;
16. **end** process process\_etiketi;
17. **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.

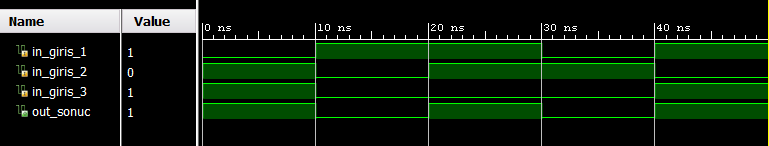
3. adımda hassasiyet listesinde bulunan **in\_giris\_2** giriş portunda değişim meydana geldiğinden **process** aktif hale gelmektedir. **process**’in aktif hale gelmesi ile birlikte **out\_cikis** çıkış portuna (**1** **and** **1**) **or** **0** işleminin sonucu atanmaktadır ve **out\_cikis** çıkış 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 ‑ 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 ‑ 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** sinyalitasarlanan **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. **entity** process\_ornek\_2 **is**
4. **Port** (
5. in\_giris\_1 : **in** std\_logic;
6. in\_giris\_2 : **in** std\_logic;
7. in\_giris\_3 : **in** std\_logic;
8. out\_sonuc : **out** std\_logic
9. );
10. **end** process\_ornek\_2;
11. **architecture** Behavioral **of** process\_ornek\_2 **is**
12. **begin**
13. process\_etiketi:**process**(in\_giris\_3)
14. **begin**
16. out\_sonuc <= (in\_giris\_1 **and** in\_giris\_2) **or** in\_giris\_3;
17. **end** process process\_etiketi;
18. **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.

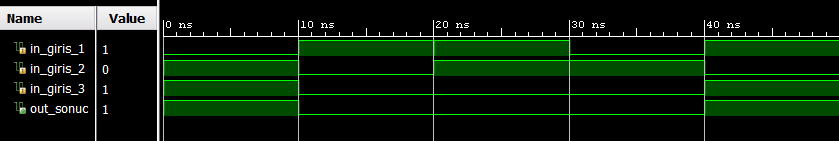
3. adımda **process** hassasiyet listesinde bulunan **in\_giris\_3** giriş portunda değişim meydana gelmediğinden **process** aktif hale gelmemektedir. Bu nedenle **out\_cikis** çıkış portu değeri değişmemiştir.

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 ‑ 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 ‑ process\_ornek\_2 varlığı benzetim çıktısı

## 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. **entity** bit\_say\_signal **is**
4. **port**(
5. in\_giris : **in** std\_logic\_vector(**0** to **7**);
6. out\_sayac : **out** integer
7. );
8. **end** bit\_say\_signal;
9. **architecture** Behavioral **of** bit\_say\_signal **is**
10. **signal** r\_sayac : integer := 0;
11. **begin**
12. **process**(in\_giris)
13. **begin**
14. r\_sayac<= 0;
15. **for** n\_i **in 7** downto **0** **loop**
16. **if** in\_giris(n\_i) = '1' **then**
17. r\_sayac<= r\_sayac + 1;
18. **end if**;
19. **end loop**;
20. **end process**;
22. out\_sayac <= r\_sayac;
23. **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.

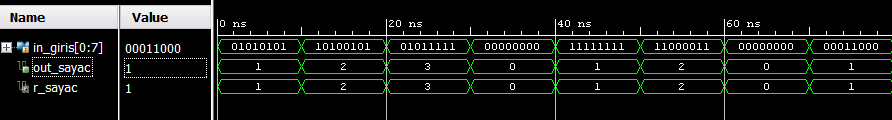
7. adımda **in\_giris** giriş portu değerinde bulunan **‘1’** bitinin bulunmamasından dolayı **r\_sayac** sinyal değeri **0** olmaktadır. **signal** veri nesnesi **process** sonunda yeni değerini aldığı için 20. satırda tanımlanan atama işlemi gerçekleştirilmiştir.

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 ‑ 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 ‑ 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. **entity** bit\_say\_variable **is**
4. **port** (
5. in\_giris : **in** std\_logic\_vector(**0** to **7**);
6. out\_sayac : **out** integer
7. );
8. **end** bit\_say\_variable;
9. **architecture** Behavioral **of** bit\_say\_variable **is**
10. **signal** r\_sayac : integer := 0;
11. **begin**
13. **process**(in\_giris)
15. **variable** v\_sayac : integer := 0;
16. **begin**
18. v\_sayac := 0;
19. **for** n\_i **in** **7** downto **0** **loop**
20. **if** in\_giris(n\_i) = '1' **then**
21. v\_sayac := v\_sayac + 1;
22. **end** **if**;
23. **end** **loop**;
25. r\_sayac <= v\_sayac;
27. **end** **process**;
28. out\_sayac <= r\_sayac;
30. **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.

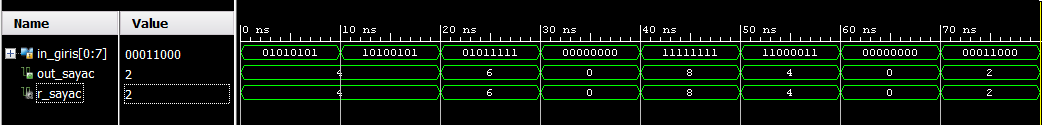
7. adımda **in\_giris** giriş portu değerinde bulunan **‘1’** bitinin bulunmamasından dolayı 23. satırda tanımlanan başlangıç değeri ataması ile **v\_sayac** sinyal değeri **0** olmaktadır. **v\_sayac** değişken değerinin **r\_sayac** sinyaline atanması ile değeri **0** olmaktadır ve bu değer **out\_cikis** 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 ‑ 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** |

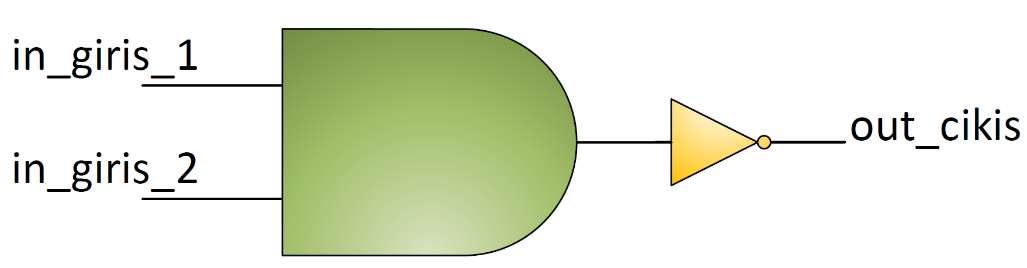


Şekil ‑ bit\_say\_variable varlığı benzetim çıktısı

**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.

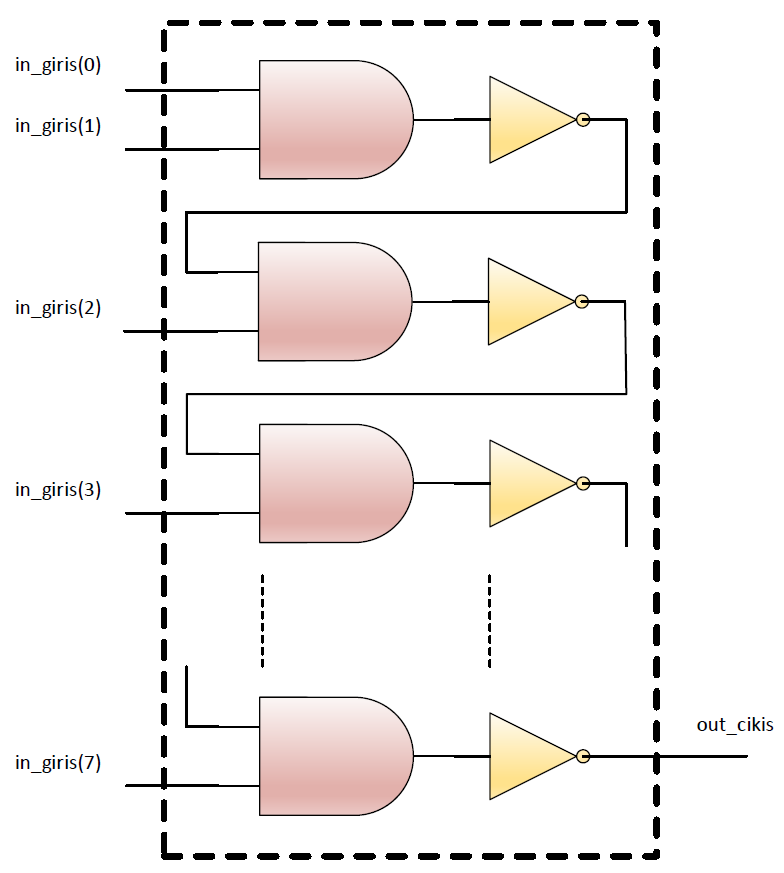
Şekil 8‑6’de **VE DEĞİL (NAND)** kapısı kullanılarak gerçeklenmek istenen uygulama blok diyagramı verilmiştir. Şekil 8‑6’den de görüleceği üzere **in\_giris** portunun **0.** biti ile **1.** biti **VE DEĞİL** işlemine tabi tutulduktan sonra elde edilen sonuç değeri **in\_giris** portunun **2.** biti 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 ‑ nand kapısı lojik devresi

Tablo ‑ 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 ‑ 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. **entity** nand\_kapi\_signal **is**
4. **port**(
5. in\_giris : **in** std\_logic\_vector(**7** downto **0**);
6. out\_cikis : **out** std\_logic
7. );
8. **end** nand\_kapi\_signal;
9. **architecture** Behavioral **of** nand\_kapi\_signal **is**
10. **signal** r\_nand\_sonuc : std\_logic := '0';
11. **begin**
13. out\_cikis <= r\_nand\_sonuc;
15. **process**(in\_giris)
16. **variable** v\_nand\_sonuc : std\_logic := '0';
17. **begin**
18. r\_nand\_sonuc <= in\_giris(0);
19. **for** n\_i **in** **1** to **7** **loop**
20. r\_nand\_sonuc <= not(r\_nand\_sonuc and in\_giris(n\_i));
21. **end** **loop**;
22. **end** **process;**
23. **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 ‑ 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))

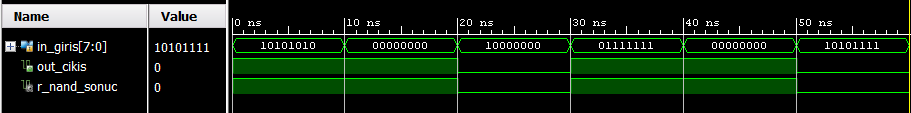
<= **not**('0' **and** '0') <= '1';

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 ‑ 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.

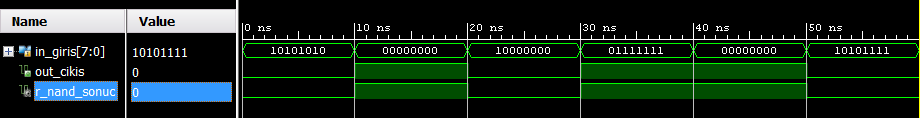
Kodda 19. satırda hassasiyet listesinde **in\_giris** girişi mevcuttur. Bunun anlamı **in\_giris** değerinde meydana gelen değişikliklerde **process** aktif hale gelecektir ve 23. satırda tanımlanan atama işlemi gerçekleşmektedir. Daha sonra 24. satırda tanımlanan **n\_i** değişkeni ile tanımlanmış döngü 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. **entity** nand\_kapi\_variable **is**
4. **port** (
5. in\_giris : **in** std\_logic\_vector(**7** downto **0**);
6. out\_cikis : **out** std\_logic
7. );
8. **end** nand\_kapi\_variable;
9. **architecture** Behavioral **of** nand\_kapi\_variable is
10. **signal** r\_nand\_sonuc : std\_logic := '0';
11. **begin**
12. out\_cikis <= r\_nand\_sonuc;
13. **process**(in\_giris)
14. **variable** v\_nand\_sonuc : std\_logic := '0';
15. **begin**
16. v\_nand\_sonuc := in\_giris(0);
17. **for** n\_i **in** **1** to **7** **loop**
18. v\_nand\_sonuc := **not**(v\_nand\_sonuc **and** in\_giris(n\_i));
19. **end** **loop**;
20. r\_nand\_sonuc <= v\_nand\_sonuc;
21. **end** **process;**
22. **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 ‑ 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 ‑ nand\_kapi\_variable varlığı benzetim çıktısı

## 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. **entity** if\_ornek\_process **is**
4. **Port** (
5. in\_giris\_1 : **in** std\_logic;
6. in\_giris\_2 : **in** std\_logic;
7. in\_giris\_secme : **in** std\_logic;
   1. out\_cikis : **out** std\_logic
8. );
9. **end** if\_ornek\_process;
10. **architecture** Behavioral **of** if\_ornek\_process **is**
11. **begin**
12. **process**(in\_giris\_1, in\_giris\_2, in\_giris\_secme)
13. **begin**
14. **if** in\_giris\_secme = '0' **then**
15. out\_cikis <= in\_giris\_1;
16. **elsif** in\_giris\_secme = '1' **then**
17. out\_cikis <= in\_giris\_2;
18. **else**
19. out\_cikis <= '0';
20. **end** **if**;
21. **end** **process**;
22. **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. **entity** if\_ornek\_function **is**
4. **Port** (
5. in\_giris\_1 : **in** std\_logic;
6. in\_giris\_2 : **in** std\_logic;
7. in\_giris\_secme : **in** std\_logic;
8. out\_cikis : **out** std\_logic
9. );
10. **end** if\_ornek\_function;
11. **architecture** Behavioral **of** if\_ornek\_function **is**
12. **function** secme\_if(sinyal\_1, sinyal\_2, sinyal\_secme : std\_logic) return std\_logic is
13. **variable** sinyal\_cikis : std\_logic;
14. **begin**
15. **if** sinyal\_secme = '0' **then**
16. sinyal\_cikis := sinyal\_1;
17. **elsif** sinyal\_secme = '1' **then**
18. sinyal\_cikis := sinyal\_2;
19. **else**
20. sinyal\_cikis := '0';
21. **end** **if**;
22. **return** sinyal\_cikis;
23. **end** secme\_if;
24. **begin**
25. out\_cikis <= secme\_if(in\_giris\_1, in\_giris\_2, in\_giris\_secme);
26. **end** Behavioral;

## 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. **entity** case\_ornek\_process **is**
4. **Port** (
5. in\_giris\_1 : **in** std\_logic;
6. in\_giris\_2 : **in** std\_logic;
7. in\_giris\_secme : **in** std\_logic;
8. out\_cikis : **out** std\_logic
9. );
10. **end** case\_ornek\_process;
11. **architecture** Behavioral **of** case\_ornek\_process **is**
12. **begin**
13. **process**(in\_giris\_1, in\_giris\_2, in\_giris\_secme)
14. **begin**
15. **case** in\_giris\_secme **is**
16. **when** '0' =>
17. out\_cikis <= in\_giris\_1;
18. **when** '1' =>
19. out\_cikis <= in\_giris\_2;
20. **when** **others** =>
21. out\_cikis <= '0';
22. **end** **case**;
23. **end** **process**;
24. **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. **entity** case\_ornek\_function **is**
4. **Port** (
5. in\_giris\_1 : **in** std\_logic;
6. in\_giris\_2 : **in** std\_logic;
7. in\_giris\_secme : **in** std\_logic;
8. out\_cikis : **out** std\_logic
9. );
10. **end** case\_ornek\_function;
11. **architecture** Behavioral **of** case\_ornek\_function **is**
12. **function** secme\_case(sinyal\_1, sinyal\_2, sinyal\_secme : std\_logic) **return** std\_logic **is**
13. **variable** sinyal\_cikis : std\_logic;
14. **begin**
15. **case** sinyal\_secme **is**
16. **when** '0' =>
17. sinyal\_cikis := sinyal\_1;
18. **when** '1' =>
19. sinyal\_cikis := sinyal\_2;
20. **when** **others** =>
21. sinyal\_cikis := '0';
22. **end** **case**;
23. **return** sinyal\_cikis;
24. **end** secme\_case;
25. **begin**
26. out\_cikis <= secme\_case(in\_giris\_1, in\_giris\_2, in\_giris\_secme);
27. **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. **entity** case\_ornek\_type **is**
5. **Port** (
6. in\_giris\_1 : **in** std\_logic\_vector(**3** downto **0**);
7. in\_giris\_2 : **in** std\_logic\_vector(**3** downto **0**);
8. out\_cikis : **out** std\_logic\_vector(**3** downto **0**)
9. );
10. **end** case\_ornek\_type;
11. **architecture** Behavioral **of** case\_ornek\_type **is**
12. **type** t\_MATH\_ISLEM **is** (TOPLA, CIKAR);
13. **signal** r\_MATH\_ISLEM : t\_MATH\_ISLEM := TOPLA;
14. **begin**
15. **process**(in\_giris\_1, in\_giris\_2)
16. **begin**
17. **case** r\_MATH\_ISLEM i**s**
18. **when** TOPLA =>
19. r\_MATH\_ISLEM <= CIKAR;
20. out\_cikis <= in\_giris\_1 + in\_giris\_2;
21. **when** CIKAR =>
22. r\_MATH\_ISLEM <= TOPLA;
23. out\_cikis <= in\_giris\_1 - in\_giris\_2;
24. **when** others =>
25. out\_cikis <= (others => '0');
26. **end** **case**;
27. **end** **process**;
28. **end** Behavioral;

## 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.

dongu\_etiketi : **for** degisken\_adi **in** aralik **loop**

sözdizimi;

{sözdizimi;}

**end** loop dongu\_etiketi ;

**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. **entity** for\_ornek\_process **is**
4. **Port**(
5. in\_giris : **in** std\_logic\_vector(7 downto 0);
6. out\_cikis : **out** std\_logic\_vector(7 downto 0)
7. );
8. **end** for\_ornek\_process;
9. **architecture** Behavioral **of** for\_ornek\_process **is**
10. **signal** r\_sinyal\_sonuc : std\_logic\_vector(7 downto 0);
11. **begin**
12. out\_cikis <= r\_sinyal\_sonuc;
13. **process**(in\_giris)
14. **variable** v\_sinyal\_sonuc : std\_logic\_vector(8 downto 0);
15. **begin**
17. v\_sinyal\_sonuc(0) := '0';
18. **for** n\_i **in** 0 to 7 **loop**
19. v\_sinyal\_sonuc (n\_i + 1) := v\_sinyal\_sonuc (n\_i) **or** in\_giris(n\_i);
20. **end loop**;
21. r\_sinyal\_sonuc <= v\_sinyal\_sonuc(8 downto 1);
22. **end** **process**;
23. **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. **entity** for\_ornek\_function **is**
4. Port (
5. in\_giris : **in** std\_logic\_vector(**7** downto **0**);
6. out\_cikis : **out** std\_logic\_vector(**7** downto **0**)
7. );
8. **end** for\_ornek\_function;
9. **architecture** Behavioral **of** for\_ornek\_function **is**
10. **function** fonksiyon\_for(sinyal\_giris : std\_logic\_vector(**7** downto **0**)) **return** std\_logic\_vector **is**
11. **variable** sinyal\_cikis : std\_logic\_vector(**8** downto **0**);
12. **begin**
13. sinyal\_cikis(0) := '0';
14. **for** n\_i **in** **0** to **7** **loop**
15. sinyal\_cikis(n\_i + 1) := sinyal\_cikis(n\_i) **or** sinyal\_giris(n\_i);
16. **end** **loop**;
17. **return** sinyal\_cikis(**8** downto **1**);
18. **end** fonksiyon\_for;
19. **begin**
20. out\_cikis <= fonksiyon\_for(in\_giris);
21. **end** Behavioral;

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

dongu\_etiketi : **while** kosul **loop**

sözdizimi;

{sözdizimi;}

**end** **loop** dongu\_etiketi ;

**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. **entity** while\_ornek\_process **is**
4. **Port** (
5. in\_giris : **in** std\_logic\_vector(**7** downto **0**);
6. out\_cikis : **out** std\_logic\_vector(**7** downto **0**)
7. );
8. **end** while\_ornek\_process;
9. **architecture** Behavioral **of** while\_ornek\_process **is**
10. **signal** r\_sinyal\_sonuc : std\_logic\_vector(**7** downto **0**);
11. **begin**
12. out\_cikis <= r\_sinyal\_sonuc;
13. **process**(in\_giris)
14. **variable** v\_sinyal\_sonuc : std\_logic\_vector(**8** downto **0**);
15. **variable** n\_i : integer := 0;
16. **begin**
18. n\_i := 0;
19. v\_sinyal\_sonuc(0) := '0';
20. **while** n\_i < 8 **loop**
21. v\_sinyal\_sonuc(n\_i + 1) := v\_sinyal\_sonuc(n\_i) **or** in\_giris(n\_i);
22. n\_i := n\_i + 1;
23. **end** **loop**;
24. r\_sinyal\_sonuc <= v\_sinyal\_sonuc(**8** downto **1**);
25. **end** **process**;
26. **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.

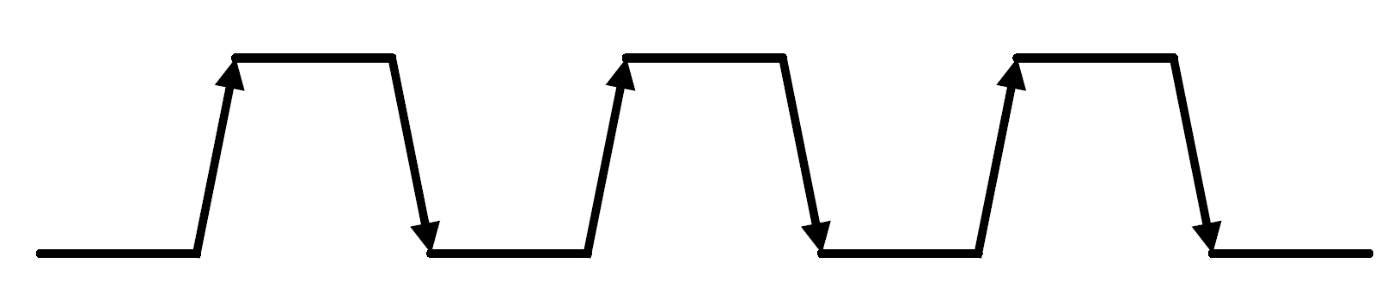
1. **library** IEEE;
2. **use** IEEE.STD\_LOGIC\_1164.ALL;
3. **entity** while\_ornek\_function **is**
4. **Port** (
5. in\_giris : **in** std\_logic\_vector(**7** downto **0**);
6. out\_cikis : **out** std\_logic\_vector(**7** downto **0**)
7. );
8. **end** while\_ornek\_function;
9. **architecture** Behavioral **of** while\_ornek\_function **is**
10. **function** fonksiyon\_while(sinyal\_giris : std\_logic\_vector(**7** downto **0**)) **return** std\_logic\_vector **is**
11. **variable** sinyal\_cikis : std\_logic\_vector(**8** downto **0**);
12. **variable** n\_i : integer := 0;
13. **begin**
14. sinyal\_cikis(0) := '0';
15. **while** n\_i < 8 **loop**
16. sinyal\_cikis(n\_i + 1) := sinyal\_cikis(n\_i) **or** sinyal\_giris(n\_i);
17. n\_i := n\_i + 1;
18. **end** **loop**;
19. **return** sinyal\_cikis(**8** downto **1**);
20. **end** fonksiyon\_while;
21. **begin**
22. out\_cikis <= fonksiyon\_while(in\_giris);
23. **end** Behavioral;

## 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 ‑ 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. **entity** saat\_darbesi\_ornek **is**
      4. **Port** (
      5. in\_clk : **in** std\_logic;
      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** saat\_darbesi\_ornek;
      12. **architecture** Behavioral **of** saat\_darbesi\_ornek **is**
      13. **begin**
      15. **process**(in\_clk)
      16. **begin**
      18. **if** rising\_edge(in\_clk) **then**
      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. **end** **if**;
      27. **end** **process**;
      28. **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.

## 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. **entity** wait\_until\_ornek **is**
4. **Port** (
5. in\_clk : **in** std\_logic;
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** wait\_until\_ornek;
12. **architecture** Behavioral **of** wait\_until\_ornek **is**
13. **begin**
14. **process**
15. **begin**
17. **wait** **until** in\_clk'event **and** in\_clk = '1';
19. **case** in\_giris\_secme **is**
20. **when** '0' =>
21. out\_cikis <= in\_giris\_1;
22. **when** '1' =>
23. out\_cikis <= in\_giris\_2;
24. **when** **others** =>
25. out\_cikis <= '0';
26. **end** **case**;
27. **end** **process**;
28. **end** Behavioral;

## 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;

..

..