7. Eşzamanlı Atama İfadeleri

Eşzamanlı atama ifadeleri mimari bölgesinde bir değerin bir sinyale atama işleminde kullanılır. VHDL 4 çeşit eşzamanlı atama ifadesi mevcuttur. Bu atama ifadeleri aşağıda tanıtılacaktır.

7.1. Basit Sinyal Atamaları

Basit sinyal atamaları lojik veya aritmetik ifadelerde kullanılır. Atama işlemine ait genel form aşağıda verilmiştir.

```
sinyal_adi <= ifade;
```

<= operatörü VHDL'de atama operatörüdür. Aşağıdaki verilen örnekte bu operatörün kullanımını daha detaylı olarak göstermektedir. **sinyal_sonuc** sinyaline, **sinyal_1** ve **sinyal_2** sinyallerinin **xor** sonucunun **sinyal_3** ile **and** işlemine tabi tutulmasında elde edilen sonuç atanmaktadır. Bu tanımlamada **sinyal_sonuc** ifadesi, bir bitlik sonucu içermektedir.

```
..
..
signal sinyal_1 : std_logic;
signal sinyal_2 : std_logic;
signal sinyal_3 : std_logic;
signal sinyal_sonuc : std_logic;
..
..
sinyal_sonuc <= (sinyal_1 xor sinyal_2) and sinyal_3;
..
..
```

VHDL'de aynı zamanda çoklu bit atamaları da yapılabilmektedir. Aşağıda verilen örnekte **sinyal_1** ve **sinyal_2** sinyalleri 3 bitlik **std_logic_vector** tipinde tanımlanmıştır. **sinyal_1** ve **sinyal_2** sinyallerinin **or** işleminin sonucu ise yine 3 bitlik **std_logic_vector** tipinde tanımlanan **sinyal_sonuc** sinyaline atanmaktadır.

```
..
..
signal sinyal_1 : std_logic_vector(2 downto 0);
signal sinyal_2 : std_logic_vector(2 downto 0);
signal sinyal_sonuc : std_logic_vector(2 downto 0);
..
..
```

```
sinyal_sonuc <= sinyal_1 or sinyal_2;
..
..
```

Bu tanımlamada tek bitli olarak aslında aşağıdaki işlemler yapılmaktadır. **sinyal_sonuc** ifadesinin 0. bitine **sinyal_1** ile **sinyal_2**'nin 0. bitlerinin **or** işlemlerinin sonucu atanmaktadır. Aynı şekilde **sinyal_sonuc** ifadesinin 1. bitine **sinyal_1** ile **sinyal_2**'nin 1. bitlerinin **or** işlemlerinin sonucu ve **sinyal_sonuc** ifadesinin 2. bitine **sinyal_1** ile **sinyal_2**'nin 2. bitlerinin **or** işlemlerinin sonucu atanmaktadır.

```
sinyal_sonuc(0) <= sinyal_1(0) or sinyal_2(0);
sinyal_sonuc(1) <= sinyal_1(1) or sinyal_2(1);
sinyal_sonuc(2) <= sinyal_1(2) or sinyal_2(2);
```

Aritmetik işlemlerde atama işlemlerine ilişkin örnek aşağıda verilmiştir. Verilen örnekte **sinyal_1** ve **sinyal_2** sinyalleri 4 bitlik **std_logic_vector** tipinde tanımlanmıştır. **sinyal_1** ve **sinyal_2** sinyallerinin **toplama** işleminin sonucu ise yine 4 bitlik **std_logic_vector** tipinde tanımlanan **sinyal_sonuc** sinyaline atanmaktadır.

```
..
..
signal sinyal_1 : std_logic_vector(3 downto 0);
signal sinyal_2 : std_logic_vector(3 downto 0);
signal sinyal_sonuc : std_logic_vector(3 downto 0);
..
..
sinyal_sonuc <= sinyal_1 + sinyal_2;
..
..
```

Yukarda belirtilen tanımlamaya alternatif olarak aşağıdaki tanımlamada kullanılabilmektedir. Verilen örnekte **sinyal_1** ve **sinyal_2** sinyalleri 4 bitlik **std_logic_vector** tipinde tanımlanmıştır. **sinyal_3** ise 1 bitlik **std_logic** tipinde tanımlanmıştır. **sinyal_1** sinyali **&** operatörü ile başına **'0'** eklenerek 5 bitlik hale getirilmiştir. Daha sonra elde edilen bu değer **sinyal_2** ve **sinyal_3** ile toplanarak işleminin sonucu ise 5 bitlik **std_logic_vector** tipinde tanımlanan **sinyal_sonuc** sinyaline atanmaktadır.

```
..
..
signal sinyal_1 : std_logic_vector(3 downto 0);
signal sinyal_2 : std_logic_vector(3 downto 0);
signal sinyal_3 : std_logic;
signal sinyal_sonuc : std_logic_vector(4 downto 0);
..
..
```

```
sinyal_sonuc<= ('0' & sinyal_1) + sinyal_2  + sinyal_3 ;
..
..
```

Örnek 7.1 : Aşağıda verilen **tam_toplayici.vhd** VHDL kodunda 21. satırda yukarda anlatılan ifadenin kullanımıyla, 4 bitlik tam toplayıcı devresinde kullanılması gösterilmektedir. Toplam değerinin, en anlamlı biti elde değerini tutmaktadır ve **out_cikis_elde** değerine atanmaktadır. Geri kalan 4 bit ise toplam sonucu olarak **out_cikis** değerine atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3. use IEEE.STD_LOGIC_SIGNED.all;
4.
5. entity tam_toplayici is
6.   port (
7.     in_giris_elde : in std_logic;
8.     in_giris_1 : in std_logic_vector(3 downto 0);
9.     in_giris_2 : in std_logic_vector(3 downto 0);
10.    out_cikis : out std_logic_vector(3 downto 0);
11.    out_cikis_elde : out std_logic
12.  );
13.end tam_toplayici;
14.
15.architecture Behavioral of tam_toplayici is
16.
17.  signal r_Toplam : std_logic_vector(4 downto 0);
18.
19.begin
20.
21.  r_Toplam<= ('0' & in_giris_1) + in_giris_2 + in_giris_elde;
22.  out_cikis_elde<= r_Toplam(4);
23.  out_cikis<= r_Toplam(3 downto 0);
24.
25.end Behavioral;
```

7.1.1. OTHERS Kullanarak Sinyal Değeri Atama

Bir sinyal değerinin belli bir kısmına aynı bitler yazılacaksa **others** kullanılarak bu işlem yapılabilmektedir. Aşağıda verilen tanımlamada **sinyal_1** sinyalinin tüm bitlerine **0**, **sinyal_2** sinyalinin tüm bitlerine ise **1** atanmaktadır.

```
..
```

```
..
signal sinyal_1 : std_logic_vector(7 downto 0)
signal sinyal_2 : std_logic_vector(3 downto 0);
..
..
sinyal_1<= (others => '0');
sinyal_2<= (others => '1');
..
..
```

7.2. Seçilmiş Sinyal Atama

Seçilmiş sinyal ifadesi, seçim koşullarına uygun birkaç alternatif değerden bir sinyalin değer olarak atanmasında kullanılır. Genel tanımlama ifadesi aşağıdaki gibidir.

```
with ifade select
     sinyal_adi<= ifade when sabit_deger
     {,ifade when sabit_deger } ;
```

Aşağıda verilen tanımlamada **sinyal_cikis** ifadesine yapılacak atama değeri **sinyal_secme** sinyalinin değerine bağlı olarak yapılmaktadır. Eğer **sinyal_secme** sinyali **'0'** ise **sinyal_cikis** değerine **sinyal_1** sinyali atanmaktadır. **sinyal_secme** sinyali 0 haricinde başka değerler alması durumunda ise **sinyal_cikis** değerine **sinyal_2** sinyali atanmaktadır.

```
..
..
signal sinyal_1 : std_logic;
signal sinyal_2 : std_logic;
signal sinyal_secme : std_logic;
signal sinyal_sonuc : std_logic;
..
..
with sinyal_secme select
    sinyal_sonuc <= sinyal_1 when '0',
    sinyal_2 when others;
..
..
```

7.3. Şartlı Sinyal Atamaları

Seçilmiş sinyal atamalarına benzer olarak; şartlı sinyal atamaları da birkaç alternatif değerden bir sinyal değerinin atanması için kullanılır. Genel gösterim aşağıdaki gibidir:

```
sinyal_adi<= ifade when lojik_ifade else
             {ifade when lojik_ifade else}
              ifade ;
```

Aşağıda verilen tanımlamada **sinyal_cikis** ifadesine yapılacak atama değeri 3 farklı koşulda belirlenmektedir. Eğer **sinyal_secme** sinyali **'0'** ise **sinyal_cikis** değerine **sinyal_1** sinyali atanmaktadır. Eğer **sinyal_secme** sinyali **'1'** ise **sinyal_cikis** değerine **sinyal_2** sinyali atanmaktadır. **sinyal_secme** sinyali **'0'** ve **'1'** haricinde başka değerler alması durumunda ise **sinyal_cikis** değerine 0 atanmaktadır.

```
..
..
signal sinyal_1 : std_logic;
signal sinyal_2 : std_logic;
signal sinyal_secme : std_logic;
signal sinyal_sonuc : std_logic;
..
..

sinyal_cikis <= sinyal_1 when sinyal_secme = '0'
             else sinyal_2 when sinyal_secme = '1'
             else '0';
..
..
```

Örnek 7.2: Aşağıda verilen **oncelikli_atama.vhd** VHDL kodunda şartlı sinyal atama kullanılarak tasarım yapılmıştır. Kodda **in_giris_1**, **in_giris_2** ve **in_giris_3** girişlerini 1 olması durumunda **out_cikis** değerine sırası ile **"01"**, **"10"** ve **"11"** atanmaktadır. Bu şartların dışında meydana gelebilecek durumlarda ise çıkışa **"00"** atanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity oncelikli_atama 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_cikis : out std_logic_vector(1 downto 0)
```

```
10.  );
11.end oncelikli_atama;
12.
13.architecture Behavioral of oncelikli_atama is
14.
15.begin
16.
17.  out_cikis<= "01" when in_giris_1 = '1'
18.      else "10" when in_giris_2 = '1'
19.      else "11" when in_giris_3 = '1'
20.      else "00" ;
21.
22.end Behavioral;
```

7.4. GENERATE İfadeleri

generate ifadesi, VHDL'de tekrarlanan lojikler eşitlikler veya **component** örneklerinde kullanılır. İki tip **generate** ifadesi mevcuttur :

- **if generate**
- **for generate**

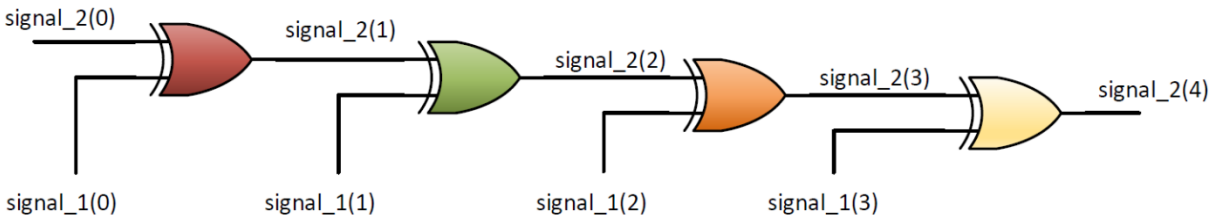
Bu ifadelerden **if generate** çok nadir kullanılır. Fakat **for generate** sıklıkla kullanılan bir ifadedir ve genel gösterimi aşağıda verilmiştir.

```
for değer in aralik generate
    ifade ;
    {ifade ;}
end generate;
```

Aşağıda verilen örnekte 4 bitlik **std_logic_vector** tipinde **sinyal_1** sinyaline başlangıç değeri olarak **"0001"** atanmıştır. **sinyal_2** sinyali de 5 bitlik **std_logic_vector** tipinde tanımlanmıştır. **sinyal_2** sinyalinin en anlamsız bitine **'1'** değeri atanmaktadır. Daha sonra **for generate** döngüsü içerisinde **sinyal_1** ve **sinyal_2**'ye ait **n_i.** bitler **xor** işlemine tabi tutulduktan sonra **sinyal_2**'nin **n_i + 1.** bitine atamaktadır. Koda ilişkin devre benzetimi Şekil 6.1'de verilmiştir.

```
..
..
signal sinyal_1 : std_logic_vector(3 downto 0) := "0001";
signal sinyal_2 : std_logic_vector(4 downto 0);
..
..
sinyal_2(0) <= '1';
```

```
for_kontrol: for n_i 0 to 3 generate
    sinyal_2(n_i + 1) <= sinyal_2(n_i) xor sinyal_1(n_i)
end generate for_kontrol;
..
..
```



Şekil 6-1 for generate için verilen örneğe ilişkin lojik gösterim

if generate ifadesinin genel gösterimi aşağıda verilmiştir.

```
if koşul generate
    ifade ;
    {ifade ;}
end generate;
```

Örnek 7.3: Aşağıda verilen **for_if_generate.vhd** VHDL kodunda **for generate** ve **if generate** ifadeleri kullanılarak 8 bitlik toplayıcı tasarımı yapılmıştır. **for_kontrol** etiketli **for generate** döngüsü ile ardışık olarak oluşturulan toplayıcı devreleri ile toplama sonucu elde edilmektedir. Döngü içerisinde bulunan **if_kontrol_EAB** etiketli **if generate** söz dizimi ile sadece **n_i** değerinin sıfır olduğu durumda söz dizimi içerisinde bulunan yarı toplayıcı lojik eşitlikleri aktif hale gelmektedir. **n_i**'nin diğer durumlarında ise bu blok pasif durumda olacaktır. **if_kontrol_DB** etiketli **if generate** söz dizimi ile sadece **n_i** değerinin sıfırdan farklı olduğu durumlarda söz dizimi içerisinde bulunan tam toplayıcı lojik eşitlikleri aktif hale getirmektedir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity for_if_generate is
5.   port (
6.     in_giris_1 : in std_logic_vector(7 downto 0);
7.     in_giris_2 : in std_logic_vector(7 downto 0);
8.     out_cikis : out std_logic_vector(7 downto 0);
9.     out_cikis_elde : out std_logic
10.  );
11.end for_if_generate;
12.
13.architecture Behavioral of for_if_generate is
```

```
14.
15.  signal r_toplam : std_logic_vector(8 downto 1);
16.
17.begin
18.
19.  for_kontrol : for n_i in 0 to 7 generate
20.    if_kontrol_EAB : if n_i = 0 generate
21.      out_cikis(n_i)<= in_giris_1(n_i) xor in_giris_2(n_i);
22.      r_toplam(n_i + 1)<= in_giris_1(n_i) and in_giris_2(n_i);
23.    end generate if_kontrol_EAB;
24.
25.    if_kontrol_DB : if n_i> 0 generate
26.      out_cikis(n_i) <= r_toplam(n_i) xor
27.      in_giris_1(n_i) xor in_giris_2(n_i);
28.
29.      r_toplam(n_i + 1)<= (r_toplam(n_i) and
30.      in_giris_1(n_i)) or (in_giris_1(n_i) and
31.      in_giris_2(n_i)) or (in_giris_2(n_i) and
32.      r_toplam(n_i));
33.
34.    end generate if_kontrol_DB;
35.  end generate for_kontrol;
36.
37.  out_cikis_elde <= r_toplam(8);
38.
39.end Behavioral;
```

Örnek 7.4: Yukarıda verilen **for_if_generate.vhd** VHDL kodunda yarı toplayıcı ve tam toplayıcı lojik eşitlikleri yerine **yari_toplayici** ve **tam_toplayici** alt devrelerinin kullanıldığı **port_map_for_if_generate.vhd** VHDL kodu aşağıda verilmiştir. Kodda 40-46. satırlarda **yari_toplayici** alt devre tasarımı yapılmıştır. Kodda 50-47. satırlarda **tam_toplayici** alt devre tasarımı yapılmıştır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity port_map_for_if_generate is
5.   Port (
6.     in_giris_1 : in std_logic_vector(7 downto 0);
7.     in_giris_2 : in std_logic_vector(7 downto 0);
8.     out_cikis_elde : out std_logic;
```

```
9.      out_cikis : out std_logic_vector(7 downto 0)
10.   );
11.end port_map_for_if_generate;
12.
13.architecture Behavioral of port_map_for_if_generate is
14.
15.   component yari_toplayici
16.   port(
17.     in_giris_1 : in std_logic;
18.     in_giris_2 : in std_logic;
19.     out_cikis : out std_logic;
20.     out_cikis_elde : out std_logic
21.   );
22.   end component;
23.
24.   component tam_toplayici
25.   port(
26.     in_giris_elde : in std_logic;
27.     in_giris_1 : in std_logic;
28.     in_giris_2 : in std_logic;
29.     out_cikis : out std_logic;
30.     out_cikis_elde : out std_logic
31.   );
32.   end component;
33.
34.   signal r_toplam : std_logic_vector(8 downto 1);
35.
36.begin
37.
38.   for_kontrol : for n_i in 0 to 7 generate
39.     if_kontrol_EAB : if n_i = 0 generate
40.       yari_toplayici_map : yari_toplayici
41.       port map(
42.         in_giris_1 => in_giris_1(n_i),
43.         in_giris_2 => in_giris_2(n_i),
44.         out_cikis => out_cikis(n_i),
45.         out_cikis_elde => r_toplam(n_i + 1)
46.       );
47.     end generate if_kontrol_EAB;
48.
```

```
49.    if_kontrol_DB : if n_i > 0 generate
50.      tam_toplayici_map : tam_toplayici
51.      port map(
52.        in_giris_elde => r_toplam(n_i),
53.        in_giris_1 => in_giris_1(n_i),
54.        in_giris_2 => in_giris_2(n_i),
55.        out_cikis => out_cikis(n_i),
56.        out_cikis_elde => r_toplam(n_i + 1)
57.      );
58.    end generate if_kontrol_DB;
59.  end generate for_kontrol;
60.
61.  out_cikis_elde <= r_toplam(8);
62.
63.end Behavioral;
```