

# VHDL

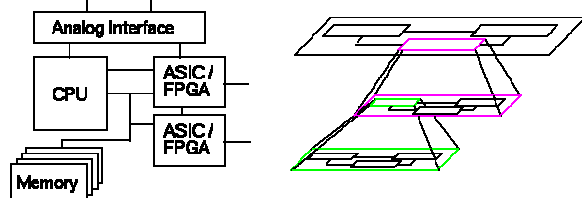
Ece Olcay Güneş, Berna Örs Yalçın

## Giriş

- VHDL "VHSIC Hardware Description Language" in kısaltmasıdır.
- VHSIC "Very High Speed Integrated Circuits" in kısaltmasıdır.
- Devre çizimi yapmak yerine bir programlama dili ile devreyi tanımlamada kullanılır.
- Büyük sayısal devre tasarımında pek çok kullanım şekli vardır.
  - Açıklama (Documentation)
  - Doğrulama (Verification)
  - Gerçekleme (Synthesis)
- Üç farklı devre tanımlama yöntemi kullanılır:
  - Yapısal (Structural)
  - Veri Akışı (Data flow)
  - Davranışsal (Behavioral)
- Çoğu zaman bu üç yöntemin karışımı kullanılır.
- VHDL 1976 yılında IEEE tarafından geliştirilmiş bir standarttır.
- Dil pek çok yenilemeden geçmiştir. Şu anda 1993 yılı üretimi en çok kullanılan halidir.

## Alt Bloklar

- Bir tasarımı daha anlaşılabilir ve değişikliklere açık hale getirebilmek için devreler alt bloklara ayrılır.
- Daha sonra bu bloklar bütün devreyi oluşturmak üzere uygun şekilde bağlanır.



## Entity

VHDL'de her blok kendi başına bir devre olarak düşünülür ve **entity** olarak adlandırılır.

**Entity:** Verilen bir lojik fonksiyon için bütün giriş ve çıkışları tanımlar. Yani lojik fonksiyonun dış dünyaya bağlantısını tanımlar. Her VHDL tasarım mutlaka en az bir entity içerir.

**entity** entity tanımlayıcı is

**port** (işaret tanımlama);

**end** entity tanımlayıcı;

**Port:** Port giriş ve çıkış işaretidir. Port ifadesi, her işaret için, port tanımlayıcı, port yönü ve port veri tipini belirlemelidir. Port da 3 yön kullanılır. Giriş için **in**, çıkış için **out**, çift yönlü portlar için **inout** kullanılır. Pek çok değişik veri tipi kullanılabilir.

**Veri tipleri:**

**bit:** 0 ve 1 değerini alabilir

**bit-vector:** aynı isim altında bir dizi 0 ve 1'li göstermek için kullanılır.

**integer:** pozitif ya da negatif tamsayılarla karşı düşer

**natural:** 0'dan başlayıp istenen bir limit değere kadar tamsayılar için kullanılır.

**positive:** 1'den başlayıp istenen bir limit değere kadar tamsayılar için kullanılır.

**Boolean:** Doğru ve yanlış olmak üzere 2 değerli bir veri tipidir.

## Entity Örneği

**entity** Ornek **is**

**port** (A,B: **in bit**; Z: **out naturel range 0 to 31**);

**end** Ornek;

- İlk satır yeni tanımlanacak devrenin ismini bildirir: Ornek
- Son satır tanımlamanın bittiğini gösterir.
- Aradaki satırlar devrenin giriş çıkışlarını tanımlar.
- Port tanımında her satır bir giriş-çıkış listesi, modunu ve veri tipini gösterir.
- Son satırın dışındaki her satır ; ile bitirilir. Port tanımının tamamı da ; ile bitirilir.

## Architecture

Lojik fonksiyonun işlevi **architecture** tarafından belirlenir. Her **entity** için bir **architecture** tanımlanır.

**architecture** mimari ismi **of** tanımlayıcı ismi **is begin**

lojik fonksiyonun yapacağı iş burada tanımlanır;  
**end** mimari ismi;

## Architecture Örneği

**entity** AND\_Gate **is**

**port**(A, B: **in bit**; X:**out bit**);

**end entity** AND\_Gate;

**Architecture** dataflow **of** AND\_Gate **is**

**begin**

X <= A **and** B;

**end** dataflow;

- İlk satır dataflow isimli mimarinin AND\_Gate isimli devreye ait olduğunu gösteriyor.
- **begin** ve **end** arasındaki satırlar AND\_Gate in nasıl bir işlem gerçekleştireceğini belirtir.

## Yapısal Tanımlamalar

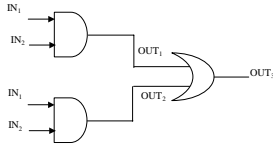
- Tasarımımızdaki temel bloklar **entity** ve karşılık gelen **architecture** kullanılarak tanımlandıktan sonra diğer tasarımlarda kullanılmak üzere birleştirilebilirler.
- Bu işlem yapılırken alt bloklar üst bloğun içinde **component** olarak tanımlanırlar.

**component** elemanın-adı **is**

**port** (uç tanımları)

**end component**;

## Component Örneği



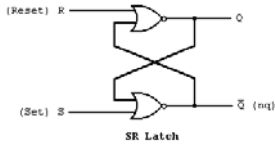
Ara bağlantılara karşı düşer. Entegre içinde dış bağlantısı olmayan elemanlar arası bağlantılardır.

```
entity AND-OR-logic is
  port(IN1, IN2, IN3, IN4: in bit; OUT3: out bit);
end AND-OR-logic;
architecture LogicOperation of AND-OR-logic is
  component AND_Gate is
    port(A, B: in bit; X: out bit);
  end component;
  component OR_Gate is
    port(A, B: in bit; X: out bit);
  end component;
  signal OUT1, OUT2: bit;
begin
  G1: AND_Gate
    port map (A=>IN1, B=>IN2,
              X=>OUT1);
  G2: AND_Gate
    port map (A=>IN3, B=>IN4,
              X=>OUT2);
  G3: OR_Gate
    port map (A=>OUT1, B=>OUT2,
              X=>OUT3);
end LogicOperation;
```

## Veri Akışı Tanımlama

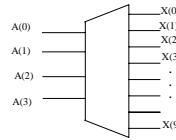
- Veri akışı tanımlamada temel blokların (örneğin *and* kapısı) girişlerinin ve çıkışlarının devre içinde nasıl bağlanacağı tanımlanır.
- İşaretlerin devre içinde nasıl akacağı tanımlanır.

## Veri Akışı Tanımlama Örneği 1



```
entity latch is
  port (s,r : in bit; q,nq :
  out bit);
end latch;
architecture dataflow of
  latch is
begin
  q<=r nor nq;
  nq<=s nor q;
end dataflow;
```

## Sayıları BCD'den Decimal'e çeviren decoder tasarımı



```
entity decoder is
  port (A: in bit-vector(0 to3); X:out bit-vector (0 to 9));
end decoder;
architecture BCD-to-decimal of decoder is
begin
  X(0)=not A(3) and not A(2) and not A(1) and not A(0);
  X(1)=not A(3) and not A(2) and not A(1) and A(0);
  X(2)=not A(3) and not A(2) and A(1) and not A(0);
  X(3)=not A(3) and not A(2) and A(1) and A(0);
  X(4)=not A(3) and A(2) and not A(1) and not A(0);
  X(5)=not A(3) and A(2) and not A(1) and A(0);
  X(6)=not A(3) and A(2) and A(1) and not A(0);
  X(7)=not A(3) and A(2) and A(1) and A(0);
  X(8)=A(3) and not A(2) and not A(1) and not A(0);
  X(9)=A(3) and not A(2) and not A(1) and A(0);
end BCD-to-decimal;
```

## Not

- Bu örnekte birden fazla giriş çıkış söz konusu olduğu için bunlar VHDL'de bir dizi (bit-vector) veri tipi ile gösterilebilir.
- Genel olarak bit dizilerinin büyüklüğü 0'dan n'ye seçilir.
- 0 her zaman dizinin ilk elemanına karşı düşer.
- Bu nedenle n dizideki eleman sayısından 1 eksik olarak seçilmelidir.
- n+1 giriş veya çıkış dizi olarak bit-vector (0 to n) olarak gösterilir.
- Örnekte giriş bit-vector (0 to 3)
- Çıkış bit-vector (0 to 9) biçimindedir.

## Kombinezonsal Devre Tasarımı Örneği

Elemanları iki bitlik olan iki matrisin çarpımını bulan devre tasarlanacaktır.

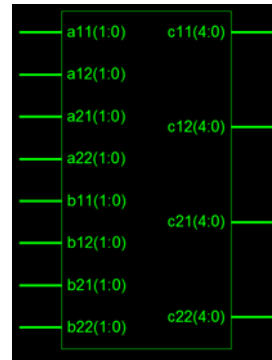
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} \quad c_{12} = a_{11}b_{12} + a_{12}b_{22}$$

$$c_{21} = a_{21}b_{11} + a_{22}b_{21} \quad c_{22} = a_{21}b_{12} + a_{22}b_{22}$$

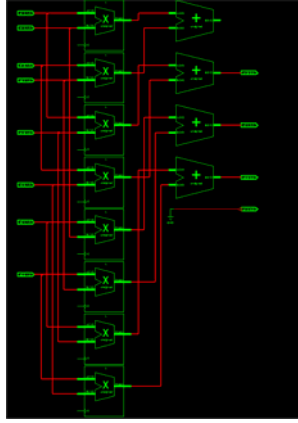
```
entity Matris_Carpici is
  port(a11 : in std_logic_vector(1 downto 0);
        a12 : in std_logic_vector(1 downto 0);
        a21 : in std_logic_vector(1 downto 0);
        a22 : in std_logic_vector(1 downto 0);
        b11 : in std_logic_vector(1 downto 0);
        b12 : in std_logic_vector(1 downto 0);
        b21 : in std_logic_vector(1 downto 0);
        b22 : in std_logic_vector(1 downto 0);
        c11 : out std_logic_vector(4 downto 0);
        c12 : out std_logic_vector(4 downto 0);
        c21 : out std_logic_vector(4 downto 0);
        c22 : out std_logic_vector(4 downto 0));
end Matris_Carpici;
```

$a_{ij}b_{kl}$  Çarpımı 4-bitliktir.  
 $c_{ij}$  5-bitlik olmalıdır.



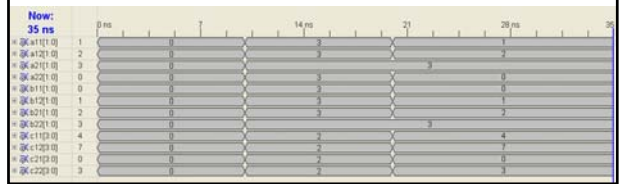
**architecture dataflow of  
Matris\_Carpici is**

```
begin
c11 <= (a11 * b11) + (a12 * b21);
c12 <= (a11 * b12) + (a12 * b22);
c21 <= (a21 * b11) + (a22 * b21);
c22 <= (a21 * b12) + (a22 * b22);
end dataflow;
```



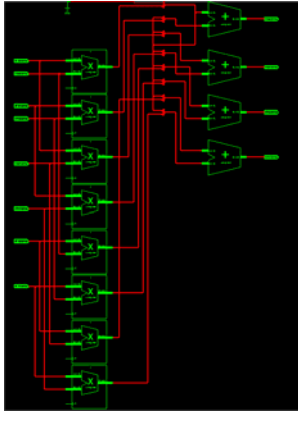
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
ENTITY matris_5bitlik_tb IS
END matris_5bitlik_tb;
ARCHITECTURE behavior OF matris_5bitlik_tb
IS
COMPONENT Matris_Carpici
PORT(a11,a12,a21,a22,b11,b12,b21,
b22 : IN std_logic_vector(1 downto 0);
c11,c12,c21,
c22 : OUT std_logic_vector(3 downto 0));
END COMPONENT;
SIGNAL a11,a12,a21,a22,b11,b12,b21,b22:
std_logic_vector(1 downto 0) := (others=>'0');
SIGNAL c11,c12,c21,
c22:std_logic_vector(3 downto 0);
```

```
BEGIN
uit: Matris_Carpici PORT MAP(
a11 => a11, a12 => a12, a21 => a21, a22 => a22,
b11 => b11, b12 => b12, b21 => b21, b22 => b22,
c11 => c11, c12 => c12, c21 => c21, c22 => c22);
tb : PROCESS
BEGIN
wait for 10 ns;
a11 <= "11"; a12 <= "11"; a21 <= "11"; a22 <= "11";
b11 <= "11"; b12 <= "11"; b21 <= "11"; b22 <= "11";
wait for 10 ns;
a11 <= "01"; a12 <= "10"; a21 <= "11"; a22 <= "00";
b11 <= "00"; b12 <= "01"; b21 <= "10"; b22 <= "11";
wait;
END PROCESS;
END;
```



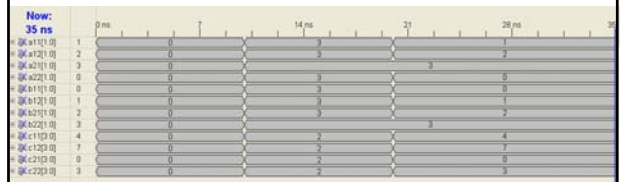
**architecture dataflow of Matris\_Carpici is**  
signal ara1, ara2, ara3, ara4, ara5, ara6,  
ara7, ara8: std\_logic\_vector(4 downto 0);

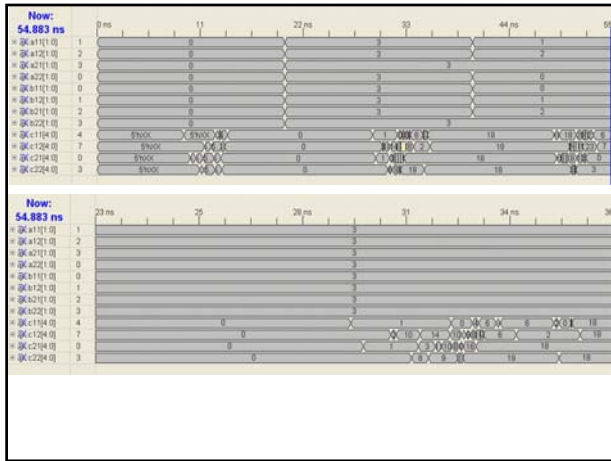
```
begin
ara1 <= (a11 * b11);
ara2 <= (a12 * b21);
ara3 <= (a11 * b12);
ara4 <= (a12 * b22);
ara5 <= (a21 * b11);
ara6 <= (a22 * b21);
ara7 <= (a21 * b12);
ara8 <= (a22 * b22);
c11 <= ara1 + ara2;
c12 <= ara3 + ara4;
c21 <= ara5 + ara6;
c22 <= ara7 + ara8;
end dataflow;
```



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
ENTITY matris_5bitlik_aracarpim_tb IS
END matris_5bitlik_aracarpim_tb;
ARCHITECTURE behavior OF
matris_5bitlik_aracarpim_tb
IS
COMPONENT Matris_Carpici
PORT(a11,a12,a21,a22,b11,b12,b21,
b22 : IN std_logic_vector(1 downto 0);
c11,c12,c21,
c22 : OUT std_logic_vector(3 downto 0));
END COMPONENT;
SIGNAL a11,a12,a21,a22,b11,b12,b21,b22:
std_logic_vector(1 downto 0) := (others=>'0');
SIGNAL c11,c12,c21,
c22:std_logic_vector(3 downto 0);
```

```
BEGIN
uit: Matris_Carpici PORT MAP(
a11 => a11, a12 => a12, a21 => a21, a22 => a22,
b11 => b11, b12 => b12, b21 => b21, b22 => b22,
c11 => c11, c12 => c12, c21 => c21, c22 => c22);
tb : PROCESS
BEGIN
wait for 10 ns;
a11 <= "11"; a12 <= "11"; a21 <= "11"; a22 <= "11";
b11 <= "11"; b12 <= "11"; b21 <= "11"; b22 <= "11";
wait for 10 ns;
a11 <= "01"; a12 <= "10"; a21 <= "11"; a22 <= "00";
b11 <= "00"; b12 <= "01"; b21 <= "10"; b22 <= "11";
wait;
END PROCESS;
END;
```





```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

package matris_tanimi is
type satir_giris is array (integer range 1 to 2) of
std_logic_vector(1 downto 0);
type satir_cikis is array (integer range 1 to 2) of
std_logic_vector(4 downto 0);
type matris_giris is array (integer range 1 to 2) of satir_giris;
type matris_cikis is array (integer range 1 to 2) of satir_cikis;
type ara_carpim is array (integer range 1 to 8) of
std_logic_vector(4 downto 0);
end matris_tanimi;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use matris_tanimi.all;

entity Matris_Carpici is
port(a : in matris_giris;
b : in matris_giris;
c : out matris_cikis);
end Matris_Carpici;

architecture dataflow of Matris_Carpici is
begin
process(a,b)
variable ara: satir_cikis;
begin
for i in 1 to 2 loop
for j in 1 to 2 loop
for k in 1 to 2 loop
ara(k) := a(i)(k) * b(k)(j);
end loop;
c(i)(j) <= ara(1) + ara(2);
end loop;
end process;
end dataflow;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use matris_tanimi.all;

entity Satir_Sutun is
port(a : in satir_giris;
b : in satir_giris;
c : out std_logic_vector(4 downto 0));
end Satir_Sutun;

architecture dataflow of Satir_Sutun is
begin
process(a,b)
variable ara: satir_cikis;
begin
for k in 1 to 2 loop
ara(k) := a(k) * b(k);
end loop;
c <= ara(1) + ara(2);
end process;
end dataflow;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use matris_tanimi.all;

entity Matris_Carpici is
port(a : in matris_giris;
b : in matris_giris;
c : out matris_cikis);
end Matris_Carpici;

architecture structure of Matris_Carpici is
component Satir_Sutun is
port(a : in satir_giris;
b : in satir_giris;
c : out std_logic_vector(4 downto 0));
end component;
begin
SS1: Satir_Sutun port map(a => a(1), b => b(1), c => c(1)(1));
SS2: Satir_Sutun port map(a => a(1), b => b(2), c => c(1)(2));
SS3: Satir_Sutun port map(a => a(2), b => b(1), c => c(2)(1));
SS4: Satir_Sutun port map(a => a(2), b => b(2), c => c(2)(2));
end structure;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use matris_tanimi.all;

entity Matris_Carpici is
port(a : in matris_giris;
b : in matris_giris;
c : out matris_cikis);
end Matris_Carpici;

architecture structure of Matris_Carpici is
component Satir_Sutun is
port(a : in satir_giris;
b : in satir_giris;
c : out std_logic_vector(4 downto 0));
end component;
begin
satir: for i in 1 to 2 generate
sutun: for j in 1 to 2 generate
SS: Satir_Sutun port map(a => a(i), b => b(j),
c => c(i)(j));
end generate sutun;
end generate satir;
end structure;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use matris_tanimi.all;

entity Matris_Carpici is
port(a : in matris_giris;
      b : in matris_giris;
      c : out matris_cikis;
      saat, reset, basla : in std_logic;
      bitti : out std_logic);
end Matris_Carpici;

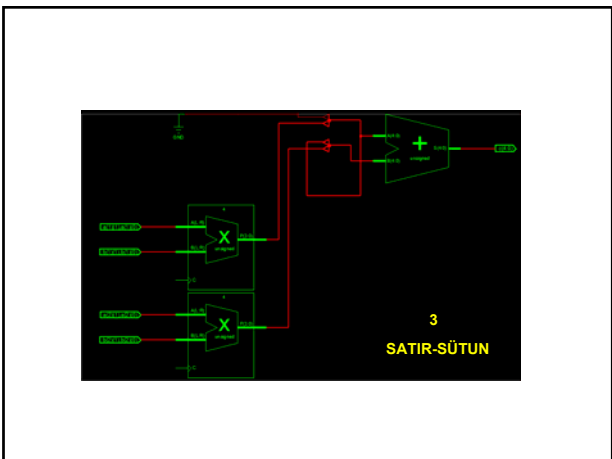
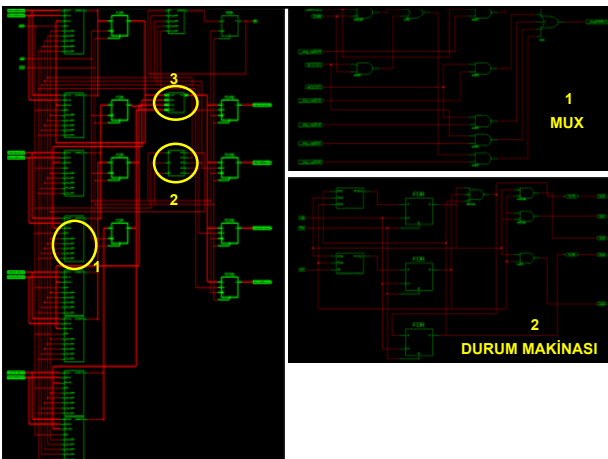
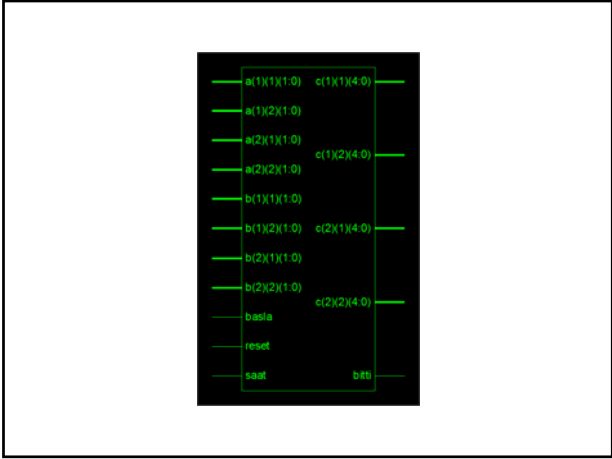
architecture structure of Matris_Carpici is
component Satir_Sutun is
port(a,b : in satir_giris;
      c : out std_logic_vector(4 downto 0));
end component;
type state_type is (baslangic, durum1, durum2,
                    durum3, durum4);
signal durum : state_type;
signal satir, sutun : satir_giris;
signal carpim : std_logic_vector(4 downto 0);
begin
SS: Satir_Sutun port map(a => satir, b => sutun,
                        c => carpim);
process(saat)
begin
if rising_edge(saat) then
if reset='1' then

```

```

for i in 1 to 2 loop
satir(i) <= (others => '0'); sutun(i) <= (others => '0');
for j in 1 to 2 loop
c(i)(j) <= (others => '0');
end loop;
end loop;
bitti <= '0'; durum <= baslangic;
else
case durum is
when baslangic =>
bitti <= '0';
if basla='1' then
durum <= durum1; satir <= a(1); sutun <= b(1);
else
durum <= baslangic;
end if;
when durum1 =>
durum <= durum2; satir <= a(1); sutun <= b(2);
c(1)(1) <= carpim;
when durum2 =>
durum <= durum3; satir <= a(2); sutun <= b(1);
c(1)(2) <= carpim;
when durum3 =>
durum <= durum4; satir <= a(2); sutun <= b(2);
c(2)(1) <= carpim;
when durum4 =>
durum <= baslangic; c(2)(2) <= carpim; bitti <= '1';
end case;
end if;
end if;
end process;
end structure;

```



```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
use matris_tanimi.all;

ENTITY matris_state_tb_vhd IS
END matris_state_tb_vhd;

ARCHITECTURE behavior OF matris_state_tb_vhd IS
  COMPONENT Matris_Carpici
  PORT(a,b : IN matris_giris;
        saat,reset,basla : IN std_logic;
        c : OUT matris_cikis;
        bitti : OUT std_logic);
  END COMPONENT;

  SIGNAL saat : std_logic := '0';
  SIGNAL reset : std_logic := '0';
  SIGNAL basla : std_logic := '0';
  SIGNAL a : matris_giris;
  SIGNAL b : matris_giris;

  SIGNAL c : matris_cikis;
  SIGNAL bitti : std_logic;

  constant cycle : time := 10 ns;

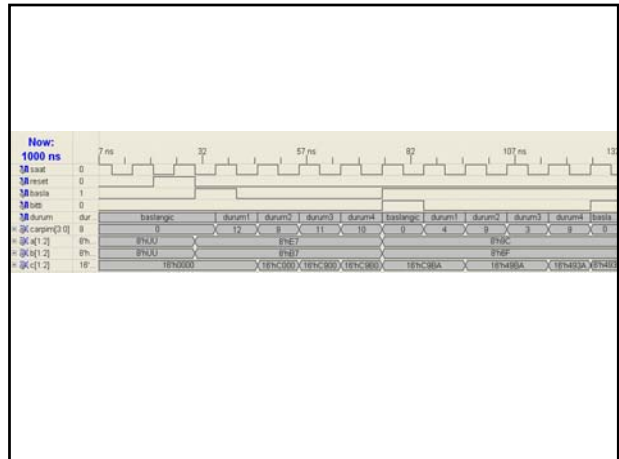
BEGIN
  uut: Matris_Carpici PORT MAP(a => a,b => b,c => c,
    saat => saat,reset => reset,basla => basla,bitti =>
    bitti);

  saat <= not saat after cycle/2;

  tb : PROCESS
  BEGIN
    wait for 20 ns;
    reset <= '1';
    wait for cycle;
    reset <= '0';
    basla <= '1';

    a(1)(1) <= "11"; a(1)(2) <= "10";
    a(2)(1) <= "01"; a(2)(2) <= "11";
    b(1)(1) <= "10"; b(1)(2) <= "11";
    b(2)(1) <= "01"; b(2)(2) <= "11";
    wait for cycle;
    basla <= '0';
    wait until bitti='1';
    basla <= '1';
    a(1)(1) <= "10"; a(1)(2) <= "01";
    a(2)(1) <= "11"; a(2)(2) <= "00";
    b(1)(1) <= "01"; b(1)(2) <= "10";
    b(2)(1) <= "11"; b(2)(2) <= "11";
    wait until bitti='1';
    wait;
  END PROCESS;
END;

```



```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
use matris_tanimi.all;

entity Satir_Sutun is
port(a : in satir_giris;
      b : in satir_giris;
      c : out std_logic_vector(4 downto 0));
end entity Satir_Sutun;

architecture structure of Satir_Sutun is
component Toplama is
port(a : in std_logic_vector(3 downto 0);
      b : in std_logic_vector(3 downto 0);
      toplam : out std_logic_vector(4 downto 0));
end component;
component Carpma is
port(a : in std_logic_vector(1 downto 0);
      b : in std_logic_vector(1 downto 0);
      carpim : out std_logic_vector(3 downto 0));
end component;
signal ara : ara_carpim;
begin
  Carp: for i in 1 to 2 generate
  C1: Carpma port map (a => a(i),b => b(i),carpim =>
    ara(i));
  end generate;
  T1: Toplama port map (a => ara(1),b => ara(2),toplam
    => c);
end architecture structure;

```

$$\begin{array}{r}
 \begin{array}{cccc}
 & a_1 & a_0 & & \\
 & & & x & \\
 & & & b_1 & b_0 \\
 \hline
 c_3 & c_2 & c_1 & c_0 & \\
 \end{array} \\
 \\
 \begin{array}{cccc}
 a_3 & a_2 & a_1 & a_0 \\
 + & b_3 & b_2 & b_1 & b_0 \\
 \hline
 c_4 & c_3 & c_2 & c_1 & c_0
 \end{array}
 \end{array}$$

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity Carpma is
port(a : in std_logic_vector(1 downto 0);
      b : in std_logic_vector(1 downto 0);
      carpim : out std_logic_vector(3 downto 0));
end Carpma;

architecture structure of Carpma is
component Yari_Toplayici is
port(a,b : in std_logic;
      toplam, elde : out std_logic);
end component;
signal ara_carpim : std_logic_vector(3 downto 0);
signal elde : std_logic;
begin
  carpim(0) <= a(0) and b(0);
  ara_carpim(0) <= a(1) and b(0);
  ara_carpim(1) <= a(0) and b(1);
  ara_carpim(2) <= a(1) and b(1);
  YT1: Yari_Toplayici
  port map (a => ara_carpim(0),b => ara_carpim(1),
    toplam => carpim(1),elde => elde);
  YT2: Yari_Toplayici
  port map (a => ara_carpim(2),b => elde,
    toplam => carpim(2),elde => carpim(3));
end architecture structure;

```

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity Toplama is
port(a : in std_logic_vector(3 downto 0);
      b : in std_logic_vector(3 downto 0);
      toplam : out std_logic_vector(4 downto 0));
end Toplama;

architecture structure of Toplama is
component Yari_Toplayici is
port(a,b : in std_logic;
      toplam, elde : out std_logic);
end component;
signal elde : std_logic_vector(2 downto 0);
begin
  YT: Yari_Toplayici
  port map (a => a(0),b => b(0),
    toplam => toplam(0),elde => elde(0));
  TT: for i in 1 to 2 generate
  TT1: Tam_Toplayici
  port map (a => a(i),b => b(i),elde_giris => elde(i-1),
    toplam => toplam(i),elde_cikis => elde(i));
  end generate;
  TT3: Tam_Toplayici
  port map (a => a(3),b => b(3),elde_giris => elde(2),
    toplam => toplam(3),elde_cikis => toplam(4));
end architecture structure;

```

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity Yari_Toplayici is
port(a : in std_logic;
     b : in std_logic;
     toplam : out std_logic;
     elde : out std_logic);
end Yari_Toplayici;

architecture dataflow of Yari_Toplayici is
begin
toplam <= a xor b;
elde <= a and b;
end architecture dataflow;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

entity Tam_Toplayici is
port(a : in std_logic;
     b : in std_logic;
     elde_giris : in std_logic;
     toplam : out std_logic;
     elde_cikis : out std_logic);
end Tam_Toplayici;

architecture dataflow of Tam_Toplayici is
begin
toplam <= a xor b xor elde_giris;
elde_cikis <= (a and b) or (a and elde_giris) or (b and elde_giris);
end architecture dataflow;

```

