Instituto Politécnico Nacional
Escuela Superior de Cómputo
Academia de Sistemas Digitales

Arreglos en VHDL

Autor: Victor Hugo García Ortega

CLASIFICACION

Los arreglos son colecciones de objetos del mismo tipo.

- Los arreglos pueden ser de 1 dimensión (1D)
- Los arreglos pueden ser de 2 dimensiones (2D)
- Los arreglos pueden ser de 1x1 dimensión (1Dx1D)
- Pueden ser de dimensiones mayores pero generalmente no son sintetizables.

DECLARACION

Los arreglos se tienen que especificar con TYPE:

```
TYPE nombre_tipo IS ARRAY (especificacion) OF data_type;
```

Después podemos declarar señales, constantes o variables de ese nuevo tipo de dato:

```
SIGNAL nombre_senial: nombre_tipo [:= valor_inicial];
```

Arreglo de 1D

```
TYPE ARREGLO IS ARRAY ( 3 DOWNTO 0 ) OF STD_LOGIC;
SIGNAL A1 : ARREGLO := "1101";
```

```
A1 <= ('1','0','0','0');
A1(0) <= '0';
```

EQUIVALENTES

```
SIGNAL A2 : STD_LOGIC_VECTOR( 3 DOWNTO 0 ) := "1101";
```

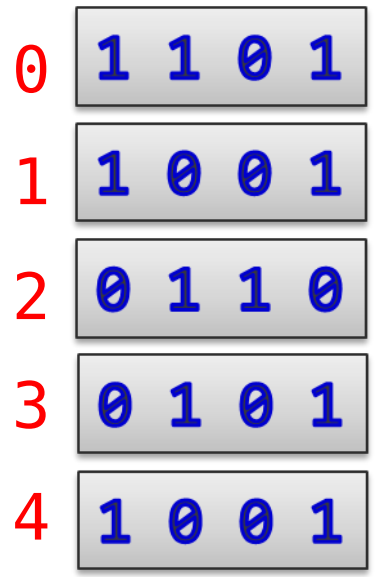
```
A2 <= ('0','1','1','0');
A2(2) <= '1';
```

1 1 0 1

Arreglo de 1D X ID

```
TYPE renglon IS ARRAY (3 DOWNTO 0) of STD_LOGIC;
TYPE matriz IS ARRAY (0 TO 4) OF renglon;
SIGNAL A1: matriz := ("1101","1001","0110","0101","1001");

TYPE matriz2 IS ARRAY (0 TO 4) OF STD_LOGIC_VECTOR (3 downto 0);
SIGNAL A2: matriz2 := ("1101","1001","0110","0101","1001");
```

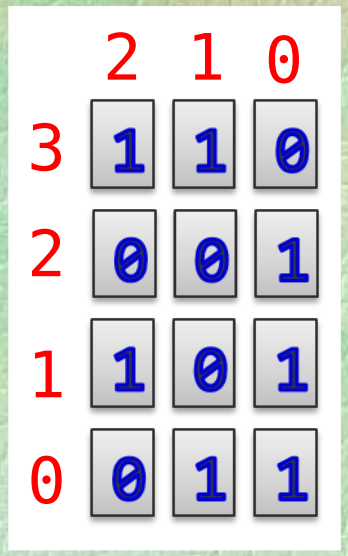
```
A1 <=( ('1','1','0','1'), ('1','0','0','1'),
       ('0','1','1','0'), ('0','1','0','1'),
       ('0','1','0','1'));
A1(0) <= "1010";
A1(1)(2) <= '1';

A2 <=( ('1','1','0','1'), ('1','0','0','1'),
       ('0','1','1','0'), ('0','1','0','1'),
       ('0','1','0','1'));
A2(2) <= "1010";
A2(2)(3) <= '1';
```



Arreglo de 2D

```
16
17 TYPE MATRIZ IS ARRAY ( 3 DOWNTO 0, 2 DOWNTO 0 ) OF STD_LOGIC;
18 SIGNAL A1 : MATRIZ := ("110","001","101","011");
19
```

```
24          A1 <= ( ('1','0','0'),('0','0','0'),
25                  ('0','0','0'),('0','1','0'));
26          A1(1,2) <= '1';
```



En la inicialización si se puede colocar los valores como un vector, pero no en el acceso individual a los elementos

Arreglo de 2Dx1D

```
16
17 TYPE MATRIZ IS ARRAY ( 3 DOWNTO 0, 2 DOWNTO 0 ) OF STD_LOGIC_VECTOR(4 DOWNTO 0);
18 SIGNAL A1 : MATRIZ := ( ("11011","00001","01101"),
19                         ("11011","00001","01101"),
20                         ("11011","00001","01101"),
21                         ("11011","00001","01101"));
22
```

```
26
27  A1 <= ( ("10010","10001","01001"),
28           ("11011","01001","01001"),
29           ("11111","01001","01111"),
30           ("10011","00101","01001"));
31
32  A1(1,2) <= "01010";
33
```

11011	00001	01101
11011	00001	01101
11011	00001	01101
11011	00001	01101

Arreglo de 3D

```
16
17 TYPE MATRIZ IS ARRAY ( 3 DOWNTO 0, 2 DOWNTO 0, 4 DOWNTO 0 ) OF STD_LOGIC;
18 SIGNAL A1 : MATRIZ := ( ("11011","00001","01101"),
19                        ("11011","00001","01101"),
20                        ("11011","00001","01101"),
21                        ("11011","00001","01101"));
22
```

```
26
27    A1 <= ( ("10010","10001","01001"),
28            ("11011","01001","01001"),
29            ("11111","01001","01111"),
30            ("10011","00101","01001"));
31
32    A1(1,2,0) <= '0';
33
```

MEMORIA ROM

Para declarar una memoria ROM en VHDL usamos un arreglo constante. Organización de 4 x 7.

```
17 CONSTANT LH : STD_LOGIC_VECTOR(6 DOWNTO 0):= "1001000";
18 CONSTANT LO : STD_LOGIC_VECTOR(6 DOWNTO 0):= "0000001";
19 CONSTANT LL : STD_LOGIC_VECTOR(6 DOWNTO 0):= "1110001";
20 CONSTANT LA : STD_LOGIC_VECTOR(6 DOWNTO 0):= "0001000";
21
22 TYPE MEMORIA IS ARRAY ( 3 DOWNTO 0 ) OF STD_LOGIC_VECTOR(6 DOWNTO 0);
23 CONSTANT ROM : MEMORIA := ( LH, LO, LL, LA );
24
```

Para accesar a sus elementos:

```
28     DATO <= ROM( CONV_INTEGER(DIRECCION) );
```

MEMORIA ROM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5  entity ROM is
6      GENERIC( BITS_BUS_DIR : INTEGER := 3;
7               BITS_BUS_DATOS : INTEGER := 7  );
8      Port (
9             DIRECCION : IN  STD_LOGIC_VECTOR( BITS_BUS_DIR-1 DOWNTO 0 );
10            BUS_DATOS : OUT STD_LOGIC_VECTOR( BITS_BUS_DATOS-1 DOWNTO 0 );
11            CS : IN STD_LOGIC
12     );
13 end ROM;
14
15 architecture MEMORIA of ROM is
16 CONSTANT H: STD_LOGIC_VECTOR(6 DOWNTO 0) := "1001000";--H
17 CONSTANT O: STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000001";--O
18 CONSTANT L: STD_LOGIC_VECTOR(6 DOWNTO 0) := "1110001";--L
19 CONSTANT A: STD_LOGIC_VECTOR(6 DOWNTO 0) := "0001000";--A
20 CONSTANT C: STD_LOGIC_VECTOR(6 DOWNTO 0) := "1011101";--"
21
22 TYPE MEMORIA IS ARRAY ( 5 DOWNTO 0 ) OF STD_LOGIC_VECTOR(BUS_DATOS'RANGE);
23 CONSTANT ROM : MEMORIA := ( C, H, O, L, A, C );
24 SIGNAL DATO : STD_LOGIC_VECTOR( BITS_BUS_DATOS-1 DOWNTO 0 );
25 begin
26
27     DATO <= ROM( CONV_INTEGER(DIRECCION) );
28
29     PBUF : PROCESS( CS, DATO )
30     BEGIN
31         IF( CS = '1' )THEN
32             BUS_DATOS <= DATO;
33         ELSE
34             BUS_DATOS <= (OTHERS => 'Z');
35         END IF;
36     END PROCESS PBUF;
37 end MEMORIA;
```

Organización de 6x7

MEMORIA ROM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5  entity ROM is
6      GENERIC( BITS_BUS_DIR : INTEGER := 7;
7               BITS_BUS_DATOS : INTEGER := 8  );
8      Port (
9             DIRECCION : IN  STD_LOGIC_VECTOR( BITS_BUS_DIR-1 DOWNTO 0 );
10            BUS_DATOS : OUT STD_LOGIC_VECTOR( BITS_BUS_DATOS-1 DOWNTO 0 );
11            CS : IN STD_LOGIC
12     );
13 end ROM;
14
15 architecture MEMORIA of ROM is
16 TYPE MEMORIA IS ARRAY ( 0 TO 2**BITS_BUS_DIR-1 ) OF
17                         STD_LOGIC_VECTOR(BUS_DATOS'RANGE);
18 CONSTANT ROM : MEMORIA := ( X"03",
19                            X"45",
20                            X"A3",
21                            X"B2",
22                            OTHERS => X"00"
23                          );
24 SIGNAL DATO : STD_LOGIC_VECTOR( BITS_BUS_DATOS-1 DOWNTO 0 );
25 begin
26
27     DATO <= ROM( CONV_INTEGER(DIRECCION) );
28
29     PBUF : PROCESS( CS, DATO )
30     BEGIN
31         IF( CS = '1' )THEN
32             BUS_DATOS <= DATO;
33         ELSE
34             BUS_DATOS <= (OTHERS => 'Z');
35         END IF;
36     END PROCESS PBUF;
37 end MEMORIA;
```

Organización de 128x8

MEMORIA RAM

```
entity RAM2B_ASINC is
    Port (
           RW : in  STD_LOGIC;
           CS : in  STD_LOGIC;
           DIRECCION : in STD_LOGIC_VECTOR (4 downto 0);
           DATAIN : in  STD_LOGIC_VECTOR (3 downto 0);
           DATAOUT : out  STD_LOGIC_VECTOR (3 downto 0));
end RAM2B_ASINC;

architecture PROGRAMA of RAM2B_ASINC is
TYPE MEMORIA IS ARRAY (0 TO 31) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL RAM : MEMORIA;
begin
   PRAM : PROCESS( CS, RW, RAM, DATAIN )
   BEGIN
      DATAOUT <= (OTHERS => 'Z');
      IF( CS = '1' )THEN
         IF( RW = '0' )THEN
            RAM( CONV_INTEGER(DIRECCION) ) <= DATAIN;
         ELSE
            DATAOUT <= RAM( CONV_INTEGER(DIRECCION) );
         END IF;
      END IF;
   END PROCESS PRAM;
end PROGRAMA;
```

Memoria con escritura y lectura asíncrona. Organización de 32 x 4.

MEMORIA RAM

```
entity RAM2B is
    Port ( CLK, RW, CS : IN  STD_LOGIC;
           DIRECCION : IN STD_LOGIC_VECTOR (5 downto 0);
           DATAIN : IN STD_LOGIC_VECTOR (7 downto 0);
           DATAOUT : OUT STD_LOGIC_VECTOR (7 downto 0));
end RAM2B;

architecture PROGRAMA of RAM2B is
TYPE MEMORIA IS ARRAY (0 TO 63) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RAM : MEMORIA;
begin
   PRAM : PROCESS( CS, RW, CLK, RAM, DATAIN )
   BEGIN
      DATAOUT <= (OTHERS => 'Z');
      IF( CS = '1' )THEN
         IF( RW = '0' )THEN
            IF( CLK'EVENT AND CLK = '1' )THEN
               RAM( CONV_INTEGER(DIRECCION) ) <= DATAIN;
            END IF;
         ELSE
            DATAOUT <= RAM( CONV_INTEGER(DIRECCION) );
         END IF;
      END IF;
   END PROCESS PRAM;
end PROGRAMA;
```

Memoria con escritura síncrona y lectura asíncrona. Organización de 64 x 8.

MEMORIA RAM

```
entity RAM2B_SINC is
    Port ( CLK, RW, CS : in  STD_LOGIC;
           DIRECCION : in STD_LOGIC_VECTOR (6 downto 0);
           DATAIN : in  STD_LOGIC_VECTOR (7 downto 0);
           DATAOUT : out  STD_LOGIC_VECTOR (7 downto 0));
end RAM2B_SINC;

architecture PROGRAMA of RAM2B_SINC is
TYPE MEMORIA IS ARRAY (0 TO 127) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL RAM : MEMORIA;
begin
   PRAM : PROCESS( CS, RW, CLK, RAM, DATAIN )
   BEGIN
      IF( CLK'EVENT AND CLK = '1' )THEN
         DATAOUT <= (OTHERS => 'Z');
         IF( CS = '1' )THEN
            IF( RW = '0' )THEN
                  RAM( CONV_INTEGER(DIRECCION) ) <= DATAIN;
            ELSE
               DATAOUT <= RAM( CONV_INTEGER(DIRECCION) );
            END IF;
         END IF;
      END IF;
   END PROCESS PRAM;
end PROGRAMA;
```

Memoria con escritura Y lectura síncrona. Organización de 128 x 8.

GRACIAS POR SU ATENCIÓN

vgarciaortega@yahoo.com.mx