DESCRIPCIÓN DE CIRCUITOS DIGITALES

Circuitos combinacionales

Circuitos secuenciales

Organización del diseño. Diseño genérico

Operaciones iterativas

Autores: Celia López, Luis Entrena, Mario García, Enrique San Millán, Marta Portela, Almudena Lindoso

Entidades y Arquitecturas

☞ **Vistas anteriormente**

```
ENTITY nombre_entidad IS
        [ GENERIC (...); ]
        [ PORT (...); ]
        [ Declaraciones de tipos, variables, componentes, etc... ; ]
BEGIN
        [ Instrucciones concurrentes pasivas ; ]
END [ nombre ] ;
```

```
ARCHITECTURE nombre_arquitectura OF nombre_entidad IS
        [ Declaraciones de tipos, variables, componentes, etc... ; ]
BEGIN
        [ Instrucciones concurrentes ; ]
END [ nombre_arquitectura ] ;
```

Puertos y parámetros genéricos

- Los puertos son el interfaz de la entidad
- Modos
  - IN: Puerto de entrada. Su valor sólo puede ser leído
  - OUT: Puerto de salida. Su valor sólo puede ser escrito
  - INOUT: Puerto de entrada y salida. Su valor puede ser leído y escrito
- Los parámetros genéricos son constantes que sirven para parametrizar los diseños. Su utilización más habitual es el modelado de constantes y tiempos de retraso

Parámetros genéricos: sumador N-bits

- Al describir el componente se declara y se usa el parámetro genérico

```
ENTITY sumador IS
    GENERIC (
        n: INTEGER := 8 );  -- Por defecto de 8 bits
    PORT (
        a,b: IN STD_LOGIC_VECTOR (n-1 downto 0);
        s: OUT STD_LOGIC_VECTOR (n-1 downto 0);
        Cout: OUT STD_LOGIC );
END sumador;

ARCHITECTURE funcional OF sumador IS
    SIGNAL suma: STD_LOGIC_VECTOR (n downto 0);
BEGIN
    suma <= '0'&a + '0'&b;                  -- Suma de
n+1 bits
    s <= suma(n-1 downto 0);    -- Resultado de n bits
    Cout <= suma(n);            -- Acarreo
END funcional;
```

- Al instanciar, se da valor al genérico
- Si no se especifica un valor, se toma el valor por defecto (n=8)
- Se puede dar otro valor al genérico:

```
ARCHITECTURE … OF … IS
    ……
BEGIN
    sum1: sumador
        GENERIC MAP ( n => 16 );
        PORT MAP ( a => a, ….. );

END …;
```

Puertos abiertos

- Los puertos se pueden dejar abiertos. Esto se indica con la palabra clave OPEN

```
COMPONENT semisumador
                PORT ( a: IN BIT;  b: IN BIT; s: OUT BIT; c: OUT BIT);
END COMPONENT;
        ......
SS0: semisumador PORT MAP (x, y, s_parcial, OPEN);
```

No se considera el acarreo

- Cuando un puerto de entrada está abierto, se toma el valor inicial por defecto

Organización del diseño

- El lenguaje VHDL soporta el uso de bibliotecas de diseño que organizan componentes o utilidades
- La creación y el mantenimiento de las bibliotecas depende de cada herramienta en particular. No obstante, el estándar VHDL sí define cómo utilizar las bibliotecas.
- Bibliotecas predefinidas:
  - Estándar (STD): contiene las definiciones de los tipos estándar BIT, BIT_VECTOR, TIME, así como las funciones de acceso a ficheros ASCII
  - Trabajo (WORK): La biblioteca de trabajo actual

Uso de bibliotecas

- La selección de componentes o utilidades se realiza mediante las denominadas claúsulas de contexto:
  - Clausúla LIBRARY: se utiliza para seleccionar una biblioteca
  - Claúsula USE: se utiliza para seleccionar una entidad, objeto, etc... definido en una biblioteca
- Las claúsulas de contexto se ponen inmediatamente antes del componente
- Se asume que todo diseño contiene implícitamente las siguientes sentencias
  - LIBRARY STD
  - USE STD.STANDARD.ALL
  - LIBRARY WORK

Ejemplo

```
LIBRARY UNISIM;                    -- Hace visible la librería UNISIM
use UNISIM.vcomponents.ALL;        -- Hace visible el paquete Vcomponents
USE WORK.mi_paquete.mi_componente;
                                   -- Hace visible mi_componente definido dentro
                                      del paquete mi_paquete
ARCHITECTURE ... OF ....
```

Las claúsulas de contexto se colocan inmediatamente antes de la declaración (entidad, arquitectura, etc...) a la que afectan

Subrutinas: Funciones y procedimientos

- Como en otros lenguajes de programación, VHDL ofrece la posibilidad de utilizar subprogramas para estructurar y modularizar las descripciones
- Dos tipos de subprogramas:
  - Funciones: devuelven siempre un valor y no pueden alterar el valor de sus parámetros
  - Procedimientos: se utilizan como sentencias y sí pueden alterar el valor de sus parámetros
- Los subprogramas pueden ser recursivos, aunque en ese caso no son sintetizables

Ejemplo de función

```
FUNCTION par (a: IN STD_LOGIC_VECTOR) RETURN STD_LOGIC IS
        VARIABLE p: STD_LOGIC;
BEGIN
        p := '0';
        FOR i IN a'RANGE  loop
                p:= p XOR a(i);
        END LOOP;
        RETURN p;
END par;
```

```
-- Llamada a la función
        r <= par ( “00101101”);
```

Ejemplo de procedimiento

```
PROCEDURE par_proc (a: IN STD_LOGIC_VECTOR; s: OUT STD_LOGIC) IS
        VARIABLE p: STD_LOGIC;
BEGIN
        p := '0';
        FOR i IN a'RANGE  LOOP
                p:= p XOR a(i);
        END LOOP;
        s := p;
END par_proc;
```

```
-- Llamada al procedimiento
        par_proc( “00101101”, r);
```


Asociación de parámetros

- La asociación de parámetros se realiza igual que en las instanciaciones de componente (por posición o explícitamente)
- Los procedimientos sólo pueden tener parámetros de tipo IN, OUT o INOUT.
- Las funciones sólo pueden tener parámetros de tipo IN.
- La clase del parámetro ha de coincidir, excepto en el caso de un parámetro constante, en que el actual puede ser cualquier expresión.
- Si la clase de parámetro no se indica, se asume que es CONSTANT en parámetros de modo IN y VARIABLE en parámetros de modo OUT o INOUT

Paquetes

- Sirven para agrupar elementos que se pueden utilizar en más de un diseño.
- Las declaraciones de paquetes pueden contener:
  - Declaraciones de tipos, subtipos, constantes, señales, ficheros, alias, subrutinas, atributos y componentes
  - Claúsulas USE
  - Especificaciones de atributos y desconexiones
- Los paquetes pueden tener también un "cuerpo de paquete", que puede contener
  - Declaraciones de tipos, subtipos, constantes, señales, ficheros, alias y subrutinas
  - Claúsulas USE
  - Cuerpos de subrutinas

Ejemplo: declaración y utilización de paquetes

```
PACKAGE mi_paquete IS
        COMPONENT semisumador
                PORT ( a: IN BIT;  b: IN BIT; s: OUT BIT; c: OUT BIT);
        END COMPONENT;
        TYPE enteros  IS ARRAY ( 0 TO 12 ) OF INTEGER;
        CONSTANT retraso : TIME := 10 NS;
        PROCEDURE par_proc (a: IN STD_LOGIC_VECTOR; s: OUT STD_LOGIC);
END mi_paquete;
```

```
USE WORK.mi_paquete.ALL;
ARCHITECTURE usando_mi_paquete OF ... IS
...
END usando_mi_paquete;
```

Ejemplo: cuerpo de paquete

```
PACKAGE BODY mi_paquete IS
PROCEDURE par_proc (a: IN STD_LOGIC_VECTOR; s: OUT STD_LOGIC) IS
        VARIABLE p: STD_LOGIC;
BEGIN
        p := '0';
        FOR i IN a'RANGE  LOOP
                p:= p XOR a(i);
        END LOOP;
        s := p;
END par_proc;
END mi_paquete;
```


Bibliotecas estándar

```
library STD;
use STD.STANDARD.all;
library WORK;
```



```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_misc.all;
use IEEE.std_logic_textio.all;
…
```

STD

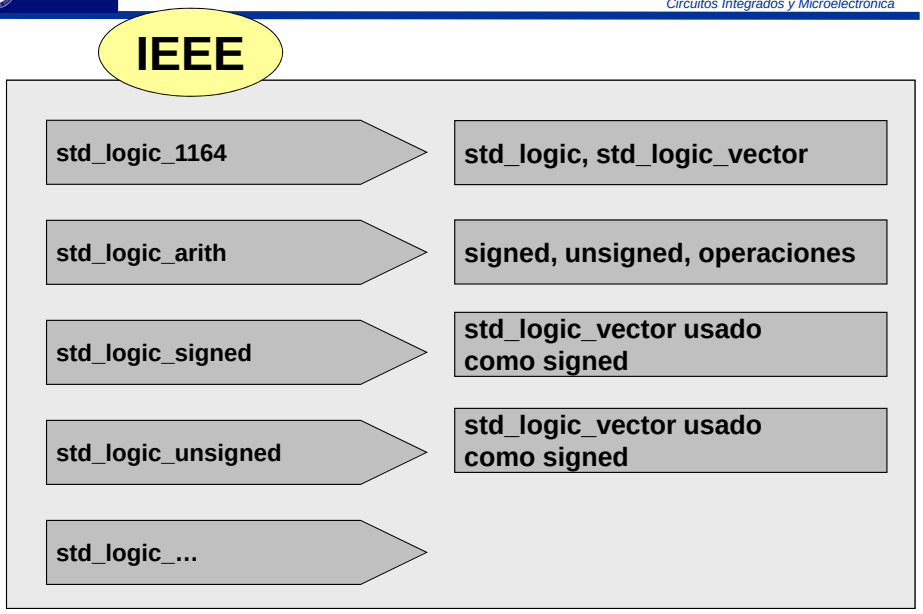
standard → **Tipos básicos**
Tipos *hardware*: time, unidades,etc.
Tipos *severity*
Operaciones

textio → **Tipos fichero y línea de fichero**
Operaciones lectura y escritura

package STANDARD is

Package

```
   TYPE BOOLEAN IS (FALSE, TRUE);
   TYPE BIT IS ('0', '1');
   TYPE CHARACTER IS
(NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, BS, HT, LF, VT,
FF, CR, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
CAN, EM, SUB, ESC, FSP, GSP, RSP, USP, ' ', '!', '"',
'#', '$', '%', '&', ''', '(', ')', '*', '+', ',', '-',
'.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8',
'9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C',
'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
'Z', '[', '\', ']', '^', '_', '`', 'a', 'b', 'c', 'd',
'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
'{', '|', '}', '~', DEL);
   TYPE SEVERITY_LEVEL IS
            (NOTE, WARNING, ERROR, FAILURE);
   TYPE INTEGER IS
            RANGE -2147483648 TO 2147483647;
   TYPE REAL IS
            RANGE -1.7014110e+038 TO 1.7014110e+038;
TYPE TIME IS
    RANGE -9223372036854775808 TO 9223372036854775807
     UNITS fs;
        ps = 1000 fs;
        ns = 1000 ps;
        us = 1000 ns;
        ms = 1000 us;
        sec = 1000 ms;
        min = 60 sec;
        hr = 60 min;
     END UNITS;
  FUNCTION NOW RETURN TIME;
  SUBTYPE NATURAL IS
            INTEGER RANGE 0 TO INTEGER'HIGH;
  SUBTYPE POSITIVE IS
             INTEGER RANGE 1 TO INTEGER'HIGH;
  TYPE STRING IS
            ARRAY (POSITIVE RANGE <>) OF CHARACTER;
  TYPE BIT_VECTOR IS
            ARRAY (NATURAL RANGE <>) OF BIT;
END STANDARD;
```



PACKAGE std_logic_1164 IS

Package

```
    ------------------------------------------------------
    -- logic state system  (unresolved)
    ------------------------------------------------------
    TYPE std_ulogic IS ( 'U',  -- Uninitialized
                         'X',  -- Forcing  Unknown
                         '0',  -- Forcing  0
                         '1',  -- Forcing  1
                         'Z',  -- High Impedance
                         'W',  -- Weak     Unknown
                         'L',  -- Weak     0
                         'H',  -- Weak     1
                        );'-'  -- Don't care
    attribute ENUM_ENCODING of std_ulogic : type is "U D 0 1 Z D 0 1 D";
    TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;
…
```

END std_logic_1164;

PACKAGE BODY std_logic_1164 IS

Package body

```
    -------------------------------------------------------------------
    -- local types
    -------------------------------------------------------------------
    --synopsys synthesis_off
    TYPE stdlogic_1d IS ARRAY (std_ulogic) OF std_ulogic;
    TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;
    -------------------------------------------------------------------
    -- resolution function
    -------------------------------------------------------------------
    CONSTANT resolution_table : stdlogic_table := (
    --      ---------------------------------------------------
    --      |  U    X    0    1    Z    W    L    H    -       |   |
    --      ---------------------------------------------------
            ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
            ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
            ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
            ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
…
```

END std_logic_1164;

Aritmética con signo y sin signo

- Paquete: IEEE.STD_LOGIC_ARITH
  - Tipos: UNSIGNED, SIGNED
  - Operaciones: +,-,*
  - Funciones: SHL, SHR, EXT, SEXT
  - Funciones de conversion:
    - STD_LOGIC_VECTOR(), SIGNED(), UNSIGNED()
    - CONV_INTEGER(), CONV_SIGNED(), CONV_UNSIGNED()
    - CONV_STD_LOGIC_VECTOR()
- Paquete: IEEE. STD_LOGIC_SIGNED
  - Operaciones con signo para STD_LOGIC_VECTOR
- Paquete: IEEE. STD_LOGIC_UNSIGNED
  - Operaciones sin signo para STD_LOGIC_VECTOR
- Opción 1:
  - Declarar STD_LOGIC_ARITH
  - Utilizar los tipos SIGNED Y UNSIGNED
- Opción 2:
  - Declarar STD_LOGIC_ARITH
  - Declarar STD_LOGIC_UNSIGNED o STD_LOGIC_SIGNED
  - Utilizar STD_LOGIC_VECTOR, que se comportarán según el paquete declarado.

Usos de bibliotecas típicos

```
-- Declaración mínima para
-- usar STD_LOGIC y STD_LOGIC_VECTOR
library IEEE;
use IEEE.std_logic_1164.all;
```

```
-- Operaciones aritméticas
-- con tipos SIGNED y UNSIGNED
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

```
-- Operaciones aritméticas que usan
-- STD_LOGIC_VECTOR como enteros sin signo
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

```
-- Operaciones aritméticas que usan
-- STD_LOGIC_VECTOR como enteros con signo
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
```


DESCRIPCIÓN DE CIRCUITOS DIGITALES

Circuitos combinacionales

Circuitos secuenciales

Organización del diseño. Diseño genérico

Operaciones iterativas

Autor: Celia López Ongil

Bucles

- Bucles

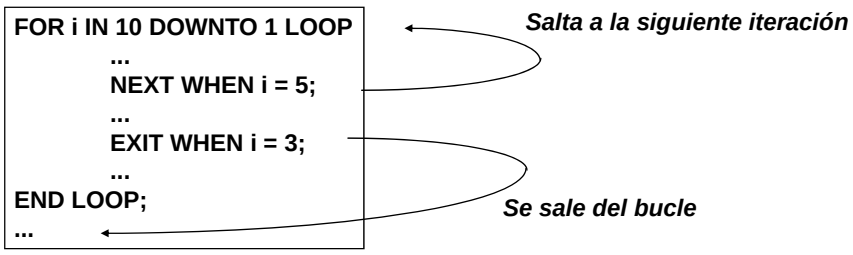
```
-- Bucle infinito
LOOP
        ...
END LOOP;
```

```
-- Bucle WHILE
WHILE a < b LOOP
        ...
END LOOP;
```

```
-- Bucle FOR
FOR i IN 10 DOWNTO 1 LOOP
        ...
END LOOP;
```

- Sentencia NEXT ... [WHEN]: salta a la siguiente iteración del bucle
- Sentencia EXIT ... [WHEN]: sale fuera del bucle

Ejemplo de utilización de NEXT y EXIT



Síntesis de bucles

- Un bucle se puede sintetizar solamente si se puede determinar estáticamente el número de iteraciones que va a realizar y por tanto el número de componentes que se van a inferir
- NEXT y EXIT no son sintetizables
- WHILE se admite en algunos sintetizadores, siempre que el número de iteraciones sea estático
- En síntesis, se deben utilizar siempre bucles FOR con un rango constante

Ejemplo

```
SIGNAL a: STD_LOGIC_VECTOR (0 TO 7);
SIGNAL z: STD_LOGIC;
...
z <= a(0) AND
     a(1) AND
     a(2) AND
     a(3) AND
     a(4) AND
     a(5) AND
     a(6) AND
     a(7);
```

```
SIGNAL a: STD_LOGIC_VECTOR (0 TO 7);
SIGNAL z: STD_LOGIC;
...
PROCESS (a)
     VARIABLE aux: STD_LOGIC;
BEGIN
     aux := a(0);
     FOR i IN 1 to 7 LOOP
          aux := aux AND a(i);
     END LOOP;
     z <= aux;
END PROCESS;
```

Sentencia GENERATE

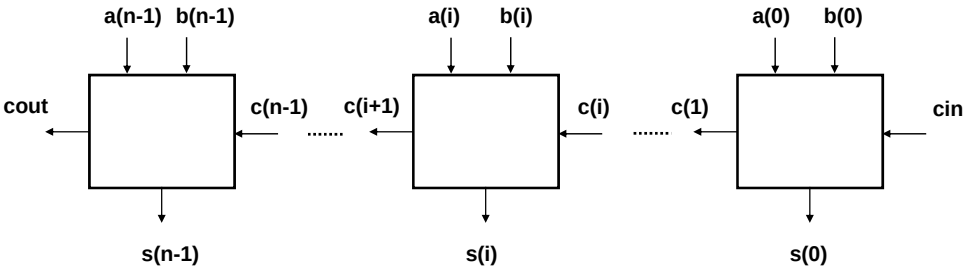
- Permite replicar estructuras de forma sencilla o dependiendo de parámetros
- Dos formas:
  - Genera las sentencias concurrentes varias veces

```
Etiqueta:  FOR nombre IN rango_discreto GENERATE
                ... -- sentencias concurrentes
           END GENERATE [ Etiqueta ];
```

  - Genera las sentencias concurrentes si se cumple una condición

```
Etiqueta:  IF condición GENERATE
                ... -- sentencias concurrentes
           END GENERATE [ Etiqueta ];
```


Ejemplo: un sumador de n bits



El sumador de n bits en VHDL

```
ENTITY sumador_n_bits IS
    GENERIC (n: INTEGER);
    PORT ( a:     IN STD_LOGIC_VECTOR ( n-1 DOWNTO 0);
           b:     IN STD_LOGIC_VECTOR ( n-1 DOWNTO 0);
           cin:   IN STD_LOGIC;
           s:     IN STD_LOGIC_VECTOR ( n-1 DOWNTO 0);
           cout:  OUT STD_LOGIC );
END sumador_n_bits;

ARCHITECTURE generada OF sumador_n_bits IS
    COMPONENT sumador_total
    PORT ( x: IN STD_LOGIC; y: IN STD_LOGIC; cin: IN STD_LOGIC;
                  s: OUT STD_LOGIC; cout: OUT STD_LOGIC);
    END COMPONENT;
    SIGNAL c : STD_LOGIC_VECTOR (n DOWNTO 0);
BEGIN
        c(0) <= cin;
fors:   FOR i IN 0 to n-1 GENERATE
sumx:           sumador_total PORT_MAP (a(i), b(i), c(i), s(i), c(i+1));
        END GENERATE;
        cout <= c(n);
END generada;
```