Tema 2. Funciones Lógicas

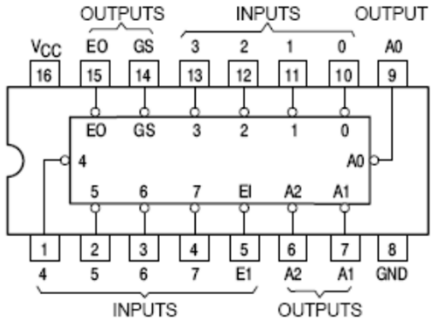
- Algebra de Conmutación.
- Representación de circuitos digitales.
- Minimización de funciones lógicas.

Representación de Circuitos Digitales

- Representación esquemática.
- Representación mediante HDL (Hardware Description Language): VHDL.

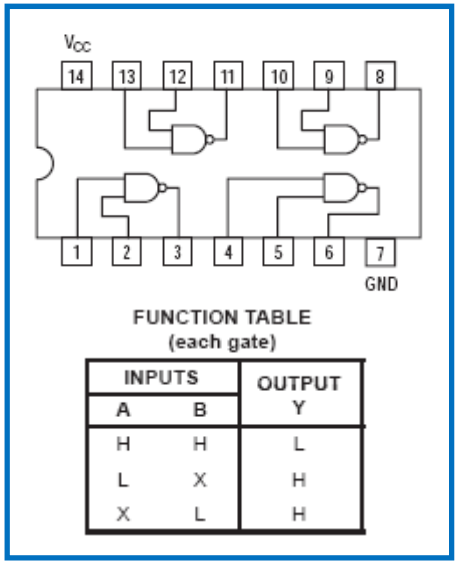
Representación Esquemática de Circuitos Digitales

- Representación esquemática tradicional: un diagrama de conexiones y una tabla de verdad asociada al dispositivo. La tabla de verdad se muestra en valores H (tensión alta, 1 lógico), L (tensión baja, 0 lógico), y X (cualquier valor).



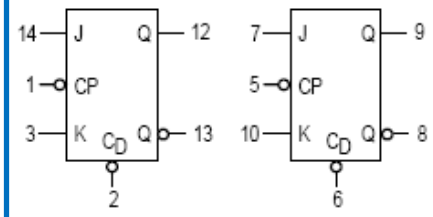
SN74LS148
FUNCTION TABLE

INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H



FUNCTION TABLE
(each gate)

INPUTS		OUTPUT
A	B	Y
H	H	L
L	X	H
X	L	H



MODE SELECT — TRUTH TABLE

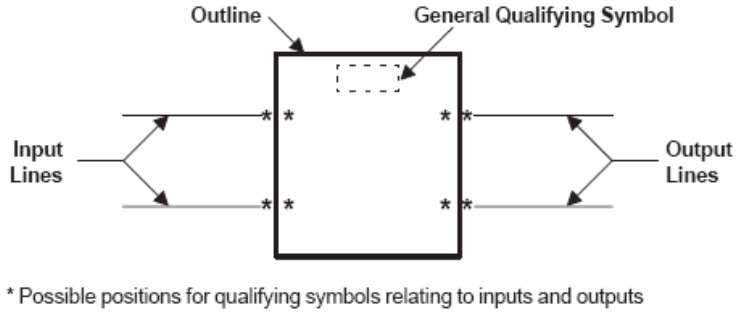
OPERATING MODE	INPUTS			OUTPUTS	
	$\overline{C}_D$	J	K	Q	$\overline{Q}$
Reset (Clear)	L	X	X	L	H
Toggle	H	h	h	$\overline{q}$	q
Load "0" (Reset)	H	l	h	L	H
Load "1" (Set)	H	h	l	H	L
Hold	H	l	l	q	$\overline{q}$

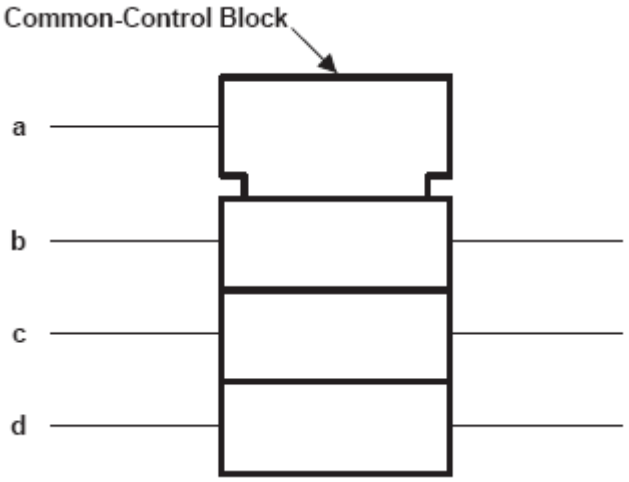
H, h = HIGH Voltage Level
L, l = LOW Voltage Level
X = Don't Care
l, h (q) = Lower case letters indicate the state of the referenced input (or output) one set-up time prior to the HIGH to LOW clock transition.

Representación Esquemática de Circuitos Digitales

- Representación esquemática estándar: ANSI/IEEE Std. 91-1984 (160 páginas aproximadamente). Incorpora al esquemático simbología que define completamente la operación del circuito.

- El esquema del circuito está basado en un rectángulo. El rectángulo puede ser dividido para representar distintas partes del circuito. Se puede incluir una zona indicando lógica de control común a varias partes iguales de un circuito.





Representación Esquemática de Circuitos Digitales

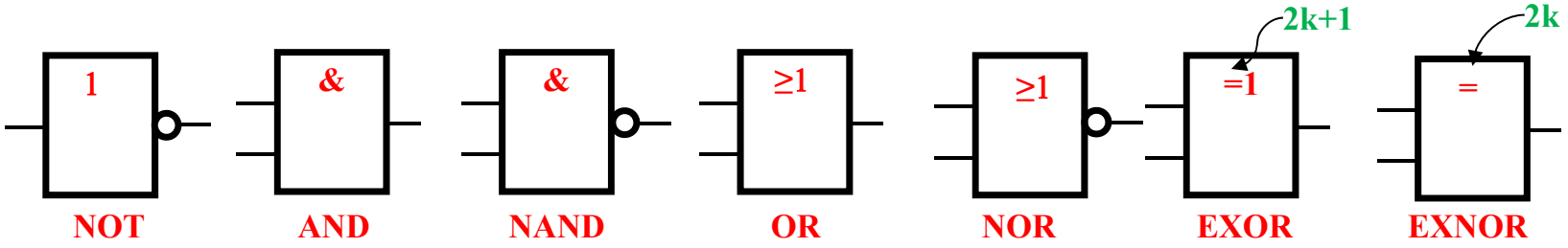
- En la parte superior del rectángulo se incluye un símbolo para indicar la operación que realiza.

SYMBOL	DESCRIPTION
&	AND gate or function
≥1	OR gate or function. The symbol was chosen to indicate that at least one active input is needed to activate the output.
=1	Exclusive OR. One and only one input must be active to activate the output.
=	Logic identity. All inputs must stand at the same state.
2k	An even number of inputs must be active.
2k + 1	An odd number of inputs must be active.
1	The one input must be active.
▷ or ◁	A buffer or element with more than usual output capability. Symbol is oriented in the direction of signal flow.
⨍	Schmitt trigger; element with hysteresis
X/Y	Coder, code converter (DEC/BCD, BIN/OUT, BIN/7-SEG, etc.)
MUX	Multiplexer/data selector
DMUX or DX	Demultiplexer
Σ	Adder
P–Q	Subtracter
CPG	Look-ahead carry generator
π	Multiplier
COMP	Magnitude comparator
ALU	Arithmetic logic unit

Representación Esquemática de Circuitos Digitales

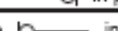
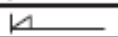
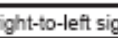
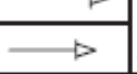
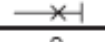
- En la parte superior del rectángulo se incluye un símbolo para indicar la operación que realiza.

⎍	Retriggerable monostable
1 ⎍	Nonretriggerable monostable (one shot)
G ⎍⎍	Astable element. Showing waveform is optional.
!G ⎍⎍	Synchronously starting astable
G! ⎍⎍	Astable element that stops with a completed pulse
SRGm	Shift register. m = number of bits
CTRm	Counter. m = number of bits; cycle length = 2^m
CTR DIVm	Counter with cycle length = m
RCTRm	Asynchronous (ripple-carry) counter; cycle length = 2^m
ROM	Read-only memory
RAM	Random-access read/write memory
FIFO	First-in, first-out memory
I = 0	Element powers up cleared to 0 state
I = 1	Element powers up set to 1 state
Φ	Highly complex function; *gray-box* symbol with limited detail shown under special rules



Representación Esquemática de Circuitos Digitales

- Las entradas y salidas pueden ser cualificadas por símbolos externos al rectángulo del dispositivo

SYMBOL	DESCRIPTION
	Logic negation at input. External 0 produces internal 1.
	Logic negation at output. Internal 1 produces external 0.
	Active-low input. Equivalent to  in positive logic.
	Active-low output. Equivalent to  in positive logic.
	Active-low input in the case of right-to-left signal flow
	Active-low output in the case of right-to-left signal flow
	Signal flow from right to left. If not otherwise indicated, signal flow is from left to right.
	Bidirectional signal flow
	Dynamic inputs active on indicated transition Positive Logic: 1→0; not used; 0→1 Negative Logic: 0→1; not used; 1→0 Polarity Indication: not used; H→L; L→H
	Nonlogic connection. A label inside the symbol usually defines the nature of this pin.
	Input for analog signals (on a digital symbol) (see Figure 14)
	Input for digital signals (on an analog symbol) (see Figure 14)
	Internal connection. 1 state on left produces 1 state on right.
	Negated internal connection. 1 state on left produces 0 state on right.
	Dynamic internal connection. Transition from 0 to 1 on left produces transitory 1 state on right.
	Internal input (virtual input). It always stands at its internal 1 state unless affected by an overriding dependency relationship.
	Internal output (virtual output). Its effect on an internal input to which it is connected is indicated by dependency notation.

Representación Esquemática de Circuitos Digitales

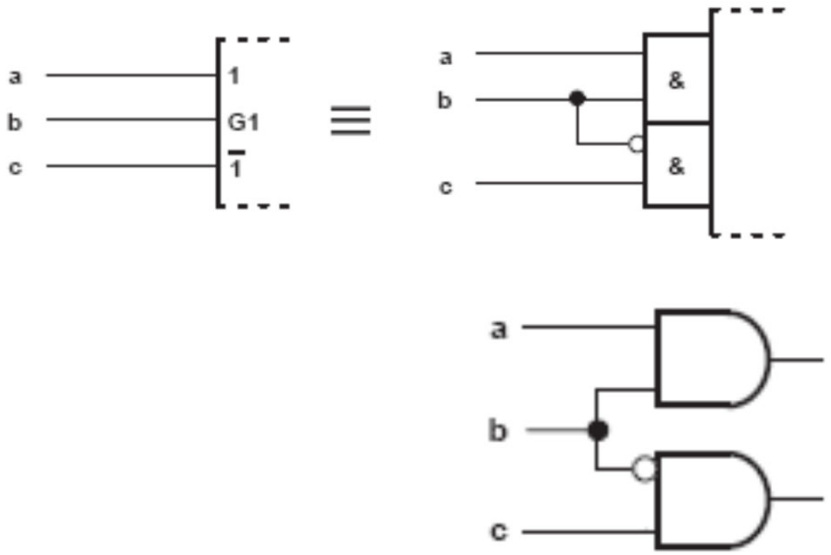
- Las entradas y salidas pueden ser cualificadas mediante símbolos internos al rectángulo del dispositivo.

SYMBOL	DESCRIPTION
⌐	Postponed output (of a pulse-triggered flip-flop). The output changes when input initiating change (e.g., a C input) returns to its initial external state or level (see Section 5).
⊓	Bi-threshold input (input with hysteresis)
Ω	N-P-N open-collector or similar output that can supply a relatively low-impedance L level when not turned off. Requires external pullup. Capable of positive-logic wired-AND connection.
Ω	Passive pullup output is similar to N-P-N open-collector output but is supplemented with a built-in passive pullup.
∇	N-P-N open-emitter or similar output that can supply a relatively low-impedance H level when not turned off. Requires external pulldown. Capable of positive-logic wired-OR connection.
∇	Passive pulldown output is similar to N-P-N open-emitter output but is supplemented with a built-in passive pulldown.
▽	3-state output
▷	Output with more than usual output capability (symbol is oriented in the direction of signal flow)
EN	Enable input. When at its internal 1 state, all outputs are enabled. When at its internal 0 state, open-collector and open-emitter outputs are off, 3-state outputs are at normally defined internal logic states and at external high-impedance state, and all other outputs (e.g., totem-poles) are at the internal 0 state.
J, K, R, S	Usual meanings associated with flip-flops (e.g., R = reset to 0, S = reset to 1)
T	Toggle input causes internal state of output to change to its complement.
D	Data input to a storage element equivalent to: S, R
m, m	Shift right (left) inputs, m = 1, 2, 3, etc. If m = 1, it usually is not shown.
+m, –m	Counting up (down) inputs, m = 1, 2, 3, etc. If m = 1, it usually is not shown.
0, m	Binary grouping. m is highest power of 2.
CT=15	The contents-setting input, when active, causes the content of a register to take on the indicated value.
CT=9	The content output is active if the content of the register is as indicated.
	Input line grouping . . . indicates two or more terminals used to implement a single logic input. X, $\overline{X}$, E
"1"	Fixed-state output always stands at its internal 1 state.

Representación Esquemática de Circuitos Digitales

- Para indicar relaciones de control entre señales se utiliza notación de dependencia. *Letra* + *Número* en una entrada, crea la dependencia *Número* de tipo *Letra*. *Números* (uno o varios separados por comas) sobre otra señal utilizan las dependencias indicadas.

DEPENDENCY TYPE OR OTHER SUBJECT
G, AND
V, OR
N, Negate (Exclusive-OR)
Z, Interconnection
X, Transmission
C, Control
S, Set and R, Reset
EN, Enable
M, Mode
A, Address



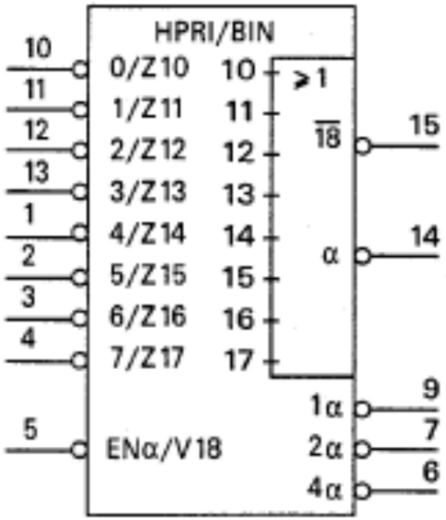
Representación Esquemática de Circuitos Digitales

- Para indicar relaciones de control entre señales se utiliza notación de dependencia.

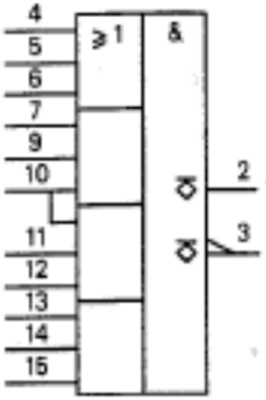
TYPE OF DEPENDENCY	LETTER SYMBOL†	AFFECTING INPUT AT ITS 1 STATE	AFFECTING INPUT AT ITS 0 STATE
Address	A	Permits action (address selected)	Prevents action (address not selected)
Control	C	Permits action	Prevents action
Enable	EN	Permits action	Prevents action of inputs ◇outputs off ▽outputs at external high impedance, no change in internal logic state. Other outputs at internal 0 state.
AND	G	Permits action	Imposes 0 state
Mode	M	Permits action (mode selected)	Prevents action (mode not selected)
Negate (Exclusive-OR)	N	Complements state	No effect
Reset	R	Affected output reacts as it would to S = 0, R = 1	No effect
Set	S	Affected output reacts as it would to S = 1, R = 0	No effect
OR	V	Imposes 1 state	Permits action
Transmission	X	Bidirectional connection exists	Bidirectional connection does not exist
Interconnection	Z	Imposes 1 state	Imposes 0 state

† These letter symbols appear at the *affecting* input (or output) and are followed by a number. Each input (or output) *affected* by that input is labeled with that same number. When the labels EN, R, and S appear at inputs without the following numbers, the descriptions above do not apply. The action of these inputs is described under Section 3.3.

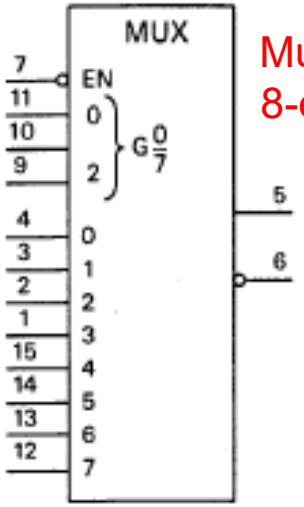
Representación Esquemática de Circuitos Digitales



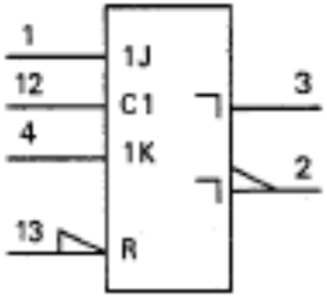
Codificador con prioridad 8 a 3



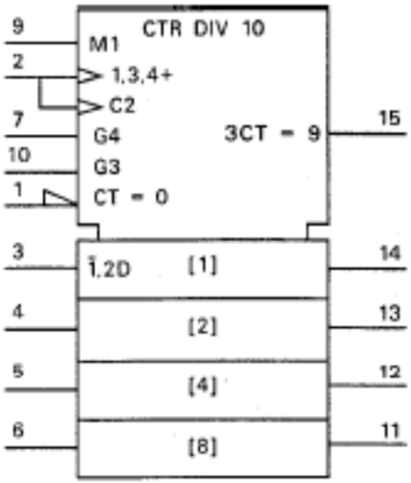
Puerta OR-AND



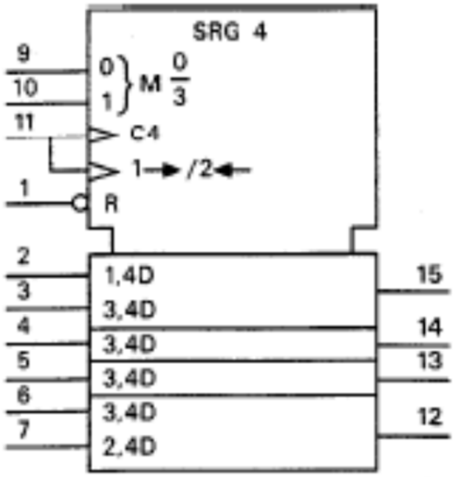
Multiplexor 8-entradas



Flip-flop J-K



Contador de décadas



Registro de desplazamiento

Introducción a VHDL

- Definición de las estructuras básicas. Entidades: genéricos y puertos. Tipos básicos. El tipo std_logic.
- Arquitecturas: operadores básicos.
- El proceso: variables, sentencias de control y de lazo.

Lenguajes HDL

- Los lenguajes HDL (Hardware Description Language) han surgido como una representación alternativa a la representación esquemática tradicional. Son descripciones de tipo texto, con sintaxis similar a la de un lenguaje de programación alto nivel, y cuya finalidad es la descripción del funcionamiento de circuitos digitales integrados.

- El VHDL (Very High-Speed Integrated Circuits Description Language), basado en el lenguaje de programación ADA es uno de los lenguajes (junto a Verilog, similar al lenguaje C) de uso más extendido.

Lenguajes HDL

- Las ventajas de una descripción mediante un HDL son:

  - Portable: la descripción de un circuito realizada para un entorno de diseño puede ser compilada por otro entorno.

  - Permite describir el circuito a varios niveles:

  Nivel de comportamiento y funcional. Cercano a las especificaciones, soporta sentencias de alto nivel, operaciones lógicas y aritméticas que podrían ser sintetizadas automáticamente en un circuito lógico (dependiendo de la capacidad del compilador).

  Nivel estructural. Cercano a la implementación final. Equivale a una descripción esquemática en la que el circuito se describe en base a la conexión de componentes.

  - Pueden aplicarse a estas descripciones herramientas CAD de simulación y de síntesis.

VHDL

- El lenguaje VHDL es muy amplio y tiene una alta complejidad. Permite la definición de funciones, de procedimientos, definición de estructuras complejas de datos, punteros, manejo de ficheros, etc. Esta introducción al VHDL se centra en las ideas y construcciones básicas del lenguaje, adecuadas para la representación VHDL de los circuitos digitales típicos y su síntesis.
- Construcciones principales de VHDL:

  Entity: vista externa del circuito.

  Architecture: asociada a una *entity*, contiene la descripción del comportamiento del circuito. Una *entity* puede tener asociadas varias *architectures*.

  Package: se utiliza para agrupar definiciones, funciones, etc, que pueden ser necesarios para definir una *entity*. Por ejemplo todo lo relacionado con un tipo de datos como el std_logic.

  Configuration: asociada a una *entity* se utiliza para caracterizar los módulos y parámetros de un circuito digital desde fuera de su *entity*. Por ejemplo para una *entity* dada con qué *entitys* y con qué *architectures* se realizan sus componentes internos.

Entity

- El formato básico de esta construcción es:

```
entity nombre is
generic (definiciones);
port (definiciones);
end nombre;
```

```
entity mux2 is
port ( I0, I1, S: in bit;    -- Entradas
       Z: out bit);          -- Salidas
end mux2;
```

- Dentro de la *entity* se definen valores genéricos y puertos de entrada y de salida. Se pueden definir otros elementos comunes a cualquier *architecture* asociada a la *entity*, pero esta introducción se limita a estos elementos.
- Los genéricos definen valores que se pueden luego utilizar en todos los elementos relacionados con la *entity*, por ejemplo el número N de bits de las entradas. El formato de los elementos genéricos es:

```
generic (nombre1: tipo (:= valor1);
         nombre2: tipo (:= valor2);
         ……….
         nombrex: tipo (:= valorX) );
```

Entity

- El formato de los puertos es:

  port (*nombrep1*: dirección tipo (:= valorp1);
  nombrep2: dirección tipo (:= valorp2);
  ……
  nombrepx: dirección tipo (:= valorpx));

  Varios puertos con el mismo tipo y dirección pueden indicarse juntos, separados por comas => I0, I1, S: **in bit**;

- La dirección de los *port* toma los valores:

  in para entradas, el puerto solo pueden aparecer en el lado derecho de una asignación.

  out para salidas, el puerto solo pueden aparecer en el lado izquierdo de una asignación.

  inout (entrada-salida).
- El tipo de un *generic* o un *port* indica el rango de valores que puede tomar el elemento.
- Es conveniente asignar un valor por defecto a un genérico, aunque es opcional tanto en los genéricos como en los puertos.

Entity

- Los tipos estándar en VHDL son:

  boolean. Valores TRUE y FALSE.

  bit. Valores '0', '1' (entre comilla simple).

  bit_vector. Conjunto o array de bits del tipo "011", "1000100" (entre comillas). El número de bits se indica mediante un formato del tipo **bit_vector**(3 **downto** 1): tres bits que se referencian para un nombre a como a(3) (más significativo), a(2) y a(1) (menos significativo), o **bit_vector**(1 **to** 3): tres bits que se referencian como a(1) (más significativo), a(2) y a(3) (menos significativo).

  integer. Números enteros entre -2147483647 y +2147483647. Se puede crear un rango dentro del tipo entero usando una sentencia del tipo **integer range** 0 **to** 15, que indica que el dato sólo puede tomar estos valores.

  real. Números reales en formato real 2.35, -4.2E-3. También se pueden crear rangos.

  character. Valores del tipo 'a', 'X', ';', etc, correspondiente al código ASCII de 7 bits.

  string. Conjunto de caracteres entre comillas como "Hola".

Entity

- Se pueden definir tipos físicos con unidades. Como tipo predefinido se tiene:

  time. Variable física que describe el tiempo. Se indica con valores numéricos seguidos de una unidad, por ejemplo 10 ns (10 nanosegundos), 2 min (dos minutos).

  Las unidades son fs (femto seg, 10E-15), ps (pico seg, 10E-12), ns (nano seg, 10E-9), us (micro seg, 10E-6), ms (mili seg, 10E-3), seg, min y hr.

- En las construcciones VHDL, en especial en *package* y *architecture* se pueden definir otros tipos (type) o subtipos (**subtype**), algunos complejos como estructuras (**record**) o **arrays**. Sentencias del tipo:

```
type valores is ('X', '0', '1');          -- Para poder definir don't cares
type conjunto is array (8 downto 1) of integer; -- 8 enteros
type semaforo is (rojo, amarillo, verde);  -- tipo enumerado
```

Tipo std_logic

- Los tipos estándar *bit*, *integer*, etc, no son capaces de describir todas las situaciones lógicas que se producen en un circuito, por ejemplo los *don´t cares*, ni situaciones electrónicas como la desconexión de un nudo. Las descripciones VHDL utilizan normalmente el tipo *std_logic* que sustituye al tipo bit y puede tener asignado uno de los siguientes nueve valores:

- ‘0’: 0 Lógico.
- ‘1’: 1 Lógico.
- ‘-’: don’t care.
- ‘Z’: alta impedancia. Nudo desconectado.
- ‘U’: no inicializado. El nudo está a 0 o a 1 pero no se sabe a qué valor. Se utiliza en circuitos secuenciales cuando comienza el circuito a operar.
- ‘L’: 0 débil. Valor de tensión bajo en circuitos electrónicos.
- ‘H’: 1 débil. Valor de tensión alto en circuitos electrónicos.
- ‘X’: Valor desconocido. Zona de incertidumbre no válida en circuitos electrónicos.
- ‘W’: desconocido débil.

Tipo std_logic

- Al igual que el tipo bit se substituye por el tipo std_logic, el tipo bit_vector es substituido por el tipo std_logic_vector.

- La definición del tipo std_logic se realiza en distintos paquetes contenidos en la librería ieee. Para usar este tipo en una entidad hay que utilizar al menos el paquete std_logic_1164 mediante estas dos sentencias:

```
library ieee;
use ieee.std_logic_1164.all;
```

- El paquete std_logic_1164 contiene la definición de los operadores lógicos para el tipo std_logic y std_logic_vector y funciones de conversión a los tipos bit y bit_vector, y a otros tipos.

Tipo std_logic

- Existen otros paquetes relacionados con el tipo std_logic que permiten utilizarlo con operadores aritméticos: sumas (+), restas (-), y multiplicaciones (*). Cuando se usa el paquete std_logic_signed se puede operar con los tipos std_logic y std_logic_vector como si fueran datos numéricos con signo en complemento-2; si se usa el paquete std_logic_unsigned se pueden usar estos tipos como datos numéricos sin signo.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

- El paquete std_logic_arith (o el paquete numeric_std) permite definir dos nuevos tipos: *signed* y *unsigned* similares al tipo std_logic_vector que operan directamente como números con signo (c-a-2) o sin signo respectivamente.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```

- Estos paquetes incluyen funciones de conversion: *conv_unsigned()*, *conv_integer()*, *conv_std_logic_vector()*, *conv_signed()*, etc, que permiten cambiar el tipo de los datos cuando es necesario.

Ejemplos de entidades

```
-- Selector de 2 entradas
library ieee;
use ieee.std_logic_1164.all;
entity mux2 is
port ( I0, I1, S: in std_logic;
       Z: out std_logic);
end mux2;
```

```
-- Calculo de la distancia de Hamming
library ieee;
use ieee.std_logic_1164.all;
entity DHgen is
generic(N: integer := 8);
port ( A, B: in std_logic_vector(N downto 1);
       DH: out integer range 0 to N);
end DHgen;
```

```
-- Toma de decision
library ieee;
use ieee.std_logic_1164.all;
entity decision is
port (T,M,P: in std_logic;
       D: out std_logic);
end decision;
```

```
-- Sumador sin signo
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity sumauns is
generic(N: integer:=4);
port (A, B: in std_logic_vector(N downto 1);
      Sum: out std_logic_vector(N+1 downto 1));
end sumauns;
```

Architecture

- Define la operación de una entity a la que está asociada. Su estructura básica es:

```
architecture nombre_arch of nombre_entidad is
-- declaraciones de tipos, señales, componentes, etc
begin
-- Sentencias concurrentes
z1 <= (I0 and (not S)) or (I1 and S);
……
process (I0, I1, S)
-- declaraciones de tipos, variables, etc
begin
    if ( S = '0' ) then    -- Sentencias secuenciales
      z2 <= I0;
    else z2 <= I1;
    end if;
……
end process;
……
U0: and2 port map (a, b, x);  -- Llamadas a otros módulos
……
end nombre_arch;
```

Architecture

- Una definición básica de una *architecture* puede hacerse a base de señales internas y de sentencias concurrentes (se realizan en paralelo, no importa el orden en el que se escriban) formadas por asignaciones y operaciones entre puertos y señales.

- Las señales internas de la *architecture* se definen en la zona de declaraciones mediante la siguiente sentencia, donde el tipo corresponde a un tipo estándar o a un tipo definido anteriormente. Se pueden definir varias señales en la misma línea, separadas por comas, si son del mismo tipo. El valor inicial es opcional.

  signal *nom_signal* : tipo (:= valor);
  -- Valor de inicialización al empezar a simular (en tiempo 0)

- Las asignaciones se realizan en sentencias del tipo:

  signal1 **<=** *signal2* op *signal3*;
  -- <= es el operador de asignación en señales y puertos

  donde *op* es un operador, y los operandos y el resultado son señales o puertos. En una sentencia se pueden encadenar operadores y operandos.

Architecture

- Los operadores se aplican sobre unos tipos determinados, un tipo definido por el usuario debería ser acompañado de la definición de operadores para operar con él. Los operadores estándar son:

  Lógicos: **not**, **and**, **nand**, **or**, **nor**, **xor**, **xnor**. Operan con los tipos bit, bit_vector y boolean. También están definidos para el tipo std_logic (y std_logic_vector).

  Relacionales (o de comparación). **=, /=** (distinto que), **>**, **<**, **>=**, **<=**. Utilizable por todos los tipos. El resultado es un dato de tipo boolean.

  Aritméticos: (**+**, **-**), (***, **/**), (** (exponenciación), **abs** (valor absoluto), **rem** (resto), **mod** (módulo)). Operan sobre tipos numéricos como el integer y el real, y tipos físicos como el time.

  Concatenación: **&**. Une datos para formar un array: C <= A & B; une dos señales, A y B, de tipo bit para formar una señal C de tipo bit_vector (2 downto 1) donde C(2) es A, y C(1) es B.

Architecture

- El orden de precedencia de los operadores de mayor a menor es:

**, ABS, NOT
*, /, MOD, REM
+, - (signo)
+, -, & (operaciones)
=, /=, <, <=, >, >=
AND, OR, NAND, NOR, XOR

```
architecture mux2_5 of mux2 is
begin
Z <= (I0 and (not S) ) or (I1 and S);
end mux2_5;
```

Se deben usar paréntesis para establecer la prioridad en asignaciones complejas. Por ejemplo: A NAND B NAND C no es una NAND de tres entradas (sería (A AND B) NAND C).

- VHDL es un lenguaje fuertemente tipado. La compilación de las sentencias de asignación produce error cuando los tipos de los operandos no son correctos.

- A las asignaciones se las puede añadir un retraso temporal, útil en simulación, mediante la construcción **after** *tiempo*.

C <= A **and** B **after** 10 ns;

Ejemplo de sentencias concurrentes

Una corporación financiera debe resolver un problema transcendente para su futuro. Para ello su presidente pide opinión a sus tres mejores economistas A, B y C, y conociendo como razonan decide que se tomará una decisión positiva si A y B están a favor, o no lo están ni A ni C, o si lo está B pero no C. Los economistas utilizan el siguiente proceso de decisión:

- A está a favor si hace buen tiempo y, es antes del mediodía siendo el día del mes par o es después del mediodía.
- B está en contra si el día del mes es impar o hace buen tiempo y, es antes del mediodía o hace mal tiempo.
- C está en contra si es antes del mediodía, hace mal tiempo y el día del mes es par.

Encontrar las ecuaciones lógicas que definen el sistema y realizar una descripción VHDL del problema.

```
library ieee;
use ieee.std_logic_1164.all;
entity decision is          -- Toma de decision
port (T,M,P: in std_logic;
      D: out std_logic);
end decision;
```

```
architecture uno of decision is
signal A, B, C: std_logic;
begin
D <= (A and B) or ((not A) and (not C)) or (B and (not C));
A <= T and ((M and P) or (not M));
B <= not (((not P) or T) and (M or (not T)));
C <= not (M and (not T) and P);
end uno;
```

Ejemplo de sentencias concurrentes

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity sumauns is
generic(N: integer:=4);
port (A, B: in std_logic_vector(N downto 1);
      Sum: out std_logic_vector(N+1 downto 1));
end sumauns;

architecture uno of sumauns is
begin
Sum <= ('0' & A) + ('0' & B);
end uno;
```

Sumador sin signo

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity sumasig is
generic(N: integer:=4);
port (A, B: in std_logic_vector(N downto 1);
      Sum: out std_logic_vector(N+1 downto 1));
end sumasig;

architecture uno of sumasig is
begin
Sum <= (A(N) & A) + (B(N) & B);
end uno;
```

Sumador con signo

Architecture

- Existen especificaciones que son más fáciles describir mediante sentencias que se realizan una después de otra de forma ordenada, que mediante sentencias en paralelo. Las sentencias ordenadas se describen dentro de la *architecture* en sentencias de tipo *process*. Pueden existir varios *process* dentro de una *architecture* operando en paralelo entre ellos:

```
process (lista de sensibilidad)
-- declaraciones de tipos, variables, etc
begin
-- Sentencias secuenciales
end process;
```

- Lista de sensibilidad: grupo de señales (separadas por comas). Los cambios de valor (o activación de eventos) en cualquiera de estas señales producen la activación del proceso y la ejecución de sus sentencias. Los cambios en las señales que no estén en la lista no activan el proceso, aunque se usen dentro de él.

Architecture

- En la zona de declaraciones del *process* se definen tipos, variables, etc, sólo reconocibles dentro del *process*. No se definen señales sino variables:

  variable *nombre* : tipo (:= valor);
  -- Valor de inicialización al empezar a simular (en tiempo 0)
  -- Para otros tiempos la variable arranca con el valor con el que
  -- acabo el proceso anterior.

- Las asignaciones sobre variables se realizan con el operador := en vez del operador <= usado sobre las señales:

  variable1 **:=** *signal2* op *port3* op *variable4*;

- Dentro del *process* se puede operar con los terminales (port), las señales definidas en la arquitectura (signal) y las variables definidas en el *process*. Las señales y puertos tienen sentido físico y se actualizan solo al final del *process*, aunque se hagan asignaciones sobre ellas en mitad del mismo. Las variables no son afectadas por tipos de tipo físico (*time*) y se actualizan en el momento que la sentencia es ejecutada dentro del *process*. La forma normal de operar sería realizar los cálculos intermedios sobre variables y cargar los resultados en señales antes de finalizar el *process*.

Architecture

- Sentencias en los *process:*

  - Sentencias de asignación en base a operadores usando señales o variables.

  - Sentencias del tipo IF-ELSE:

```
if condición then
   sentencias;
end if;
```

```
if condición then
   sentencias;
else
   sentencias;
end if;
```

```
if condición1 then
   sentencias;
elsif condición2 then
   sentencias;
……
else sentencias;
end if;
```

Si durante la ejecución de un *process* no se asigna valor a una *variable* o a una *signal* mantiene su valor anterior, y se genera un circuito secuencial (latch). Solución sencilla: inicializar variables y/o señales al principio del *process*.

```
architecture mux2_1 of mux2 is
begin
process (I0, I1, S)
begin
if ( S = '0' ) then Z <= I0;
else Z <= I1;
end if;
end process;
end mux2_1;
```

Architecture

- Sentencias de tipo CASE

```
case expresión is
   when valor1 => sentencias1;
   when valor2 => sentencias2;
   ……….
  (when others => sentenciasx;)
end case;
```

- Sentencias de tipo LAZO

```
for i in vinicial to vfin loop
    -- en función de i
    sentencias;
end loop;
```

(o **downto**)

```
while condición loop
    sentencias;
end loop;
```

```
architecture mux2_3 of mux2 is
begin
process (I0, I1, S)
variable t:std_logic_vector(3 downto 1);
begin
t := S & I0 & I1;
case t is
    when "000" => Z <= '0';
    when "001" => Z <= '0';
    when "100" => Z <= '0';
    when "110" => Z <= '0';
    when others => Z <= '1';
end case;
end process;
end mux2_3;
```

```
architecture uno of DHgen is
begin
process (A,B)
variable inter:integer range 0 to N;
begin
inter := 0;
for i in 1 to N loop
    if ( A(i) /= B(i) ) then
        inter := inter + 1;
    end if;
end loop;
DH <= inter;
end process;
end uno;
```

Architecture

Una sociedad está formada por 5 socios A, B, C, D y E que tienen respectivamente el 25%, 25%, 25%, 15% y 10% de las acciones. Los estatutos de la sociedad indican que una toma de decisión es positiva si el tanto por ciento a favor es mayor del 65%, o si estando entre el 35% y el 65% (ambos incluidos) hay mayoría de votos a favor entre los tres socios más antiguos C, D y E (sin contar su porcentaje respectivo). En caso contrario, la decisión es negativa. Realizar una descripción VHDL de alto nivel del problema a partir de este enunciado (por ejemplo utilizar variables de tipo entero para calcular el porcentaje de voto favorable o el número de votos a favor).

```
library ieee;
use ieee.std_logic_1164.all;
entity votacion is
port (A,B,C,D,E: in std_logic;
      V: out std_logic);
end votacion;
```

Architecture

```
library ieee;
use ieee.std_logic_1164.all;
entity votacion is
port (A,B,C,D,E: in std_logic;
      V: out std_logic);
end votacion;
```

```
architecure uno of votacion is
begin
process(A,B,C,D,E)
variable porc: integer range 0 to 100;
variable n_voto: integer range 0 to 3;
begin
porc := 0; n_voto := 0;
if (A = '1') then porc := porc +25; end if;
if (B = '1') then porc := porc +25; end if;
if (C = '1') then porc := porc +25; n_voto := n_voto + 1; end if;
if (D = '1') then porc := porc +15; n_voto := n_voto + 1; end if;
if (E = '1') then porc := porc +10; n_voto := n_voto + 1; end if;

if ( porc > 65 ) then V <= '1';
elsif ( porc < 35 ) then V <= '0';
elsif ( n_voto > 1 ) then V <= '1';
else V <= '0';
end if;

end process;
end uno;
```

Architecture

Se quiere diseñar una pequeña ALU (unidad aritmético lógica) para una aplicación que realice cuatro operaciones lógicas para tres operandos de datos A, B y C de 1 bit. Las operaciones que deben realizar se muestran en la siguiente tabla en función de dos señales de control S1 y S0. Además, se sabe que los valores lógicos del par de entradas (BC) no coinciden nunca con los valores del par (S1S0). Realizar una descripción VHDL de este circuito.

S1 S0	Función
00	$\overline{B}\,C$
01	$\overline{A \oplus C}$
10	$\overline{A} + \overline{B}$
11	$A\,C + \overline{A}\,B\,\overline{C}$

```
library ieee;
use ieee.std_logic_1164.all;
entity ALU is
port (A,B,C,S1,S0: in std_logic;
      F: out std_logic);
end ALU;
```

Architecture

```
library ieee;
use ieee.std_logic_1164.all;
entity ALU is
port (A,B,C,S1,S0: in std_logic;
      F: out std_logic);
end ALU;
```

S1 S0	Función
00	$\overline{B}\,C$
01	$\overline{A \oplus C}$
10	$\overline{A} + \overline{B}$
11	$A\,C + \overline{A}\,B\,\overline{C}$

```
architecure uno of ALU is
begin
process(A,B,C,S1,S0)
variable S,BC: std_logic_vector(2 downto 1);
begin
S := S1 & S0; BC := B & C;

if (S = BC) then F <= '-';
else
   case S is
      when "00" => F <= (not B) and C;
      when "01" => F <= not (A xor C);
      when "10" => F <= (not A) or (not B);
      when others => F <= (A and C) or ((not A) and B and (not C));
   end case;
end if;
end process;
end uno;
```