CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

ELECTRÓNICA DIGITAL

Unidad 1

Descripción y Simulación de

Circuitos Digitales Utilizando VHDL

Agosto, 2014.

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

ELECTRÓNICA DIGITAL

Unidad 1: Descripción y Simulación de Circuitos Digitales Utilizando VHDL

Profesor Titular: Dr. Mario Alfredo Reyes Barranca
Profesores Auxiliares: Dr. Oliverio Arellano Cárdenas
M. en C. Luis Martín Flores Nava

Revisión 5.0, Agosto 2014.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

ELECTRÓNICA DIGITAL

Unidad 1: Descripción y Simulación de Circuitos Digitales Utilizando VHDL

Profesor Titular: Dr. Mario Alfredo Reyes Barranca
Profesores Auxiliares: Dr. Oliverio Arellano Cárdenas
M. en C. Luis Martín Flores Nava

Objetivo

Proporcionar al alumno una herramienta de descripción de circuitos digitales para el proceso de diseño y simulación, así como implementar estos circuitos en dispositivos programables.

Temario

Capítulo 1 **Estado actual de la lógica programable**
1.1 Conceptos fundamentales
1.2 Dispositivos lógicos programables simples (SPLD's)
1.3 Dispositivos lógicos programables complejos (CPLD's)
1.4 Arreglo de compuertas programables en campo (FPGA's)

Capítulo 2 **Sintaxis del lenguaje**
2.1 Introducción a la descripción en VHDL de circuitos digitales
2.2 Estilos de programación en VHDL
2.3 Operadores y expresiones
2.4 Objetos de datos
2.5 Tipos de datos
2.6 Declaración de entidad y arquitectura

Capítulo 3 **Circuitos Lógicos combinatorios**
3.1 Declaraciones concurrentes
3.1.1 La construcción ***when-else*** y ***with-select-when***
3.2 Declaraciones secuenciales
3.2.1 La construcción ***if-then-else*** y ***case***

Capítulo 4 **Circuitos Lógicos secuenciales**
4.1 Diseño lógico secuencial
4.2 Elementos de memoria
4.3 Registros
4.4 Contadores
4.5 Máquinas de estado

Bibliografía

1. Circuit Design with VHDL
Volnei A. Pedroni
Massachussets Institute of Technology, 2004.

2. HDL Chip Design
Douglas J. Smith
Doone Publications, Madison, AL, USA 1996.

3. Analysis and Design of Digital Systems with VHDL
Allen M. Dewey
PWS Publishing Company, Boston, MA 1997.

4. VHDL: Lenguaje para síntesis y modelado de circuitos
Fernando Pardo y José A. Boluda
Alfaomega, 2000.

CENTRO DE INVESTIGACION Y ESTUDIOS AVANZADOS DEL I.P.N.

Departamento de Ingeniería Eléctrica

Curso:

Electrónica Digital

Cinvestav

Descripción y Simulación de Circuitos Digitales Utilizando VHDL

Profesor Titular: Dr. Mario Alfredo Reyes Barranca

Profesores Auxiliares: Dr. Oliverio Arellano Cárdenas
M. en C. Luis Martín Flores Nava

Capítulo 1

Estado Actual de la Lógica Programable

Cinvestav

PLD's

DISPOSITIVOS LOGICOS PROGRAMABLES

Una alternativa en el Diseño de Sistemas Digitales

Introducción

Cinvestav

- La realidad del diseño lógico actual:
  - Complejidad creciente
  - Tiempos menores de introducción al mercado
  - Disminución de costos
- Las exigencias que plantea son:
  - Confiabilidad
  - Accesibilidad para pruebas
- La meta principal es:
  - Contar con una solución de uso universal

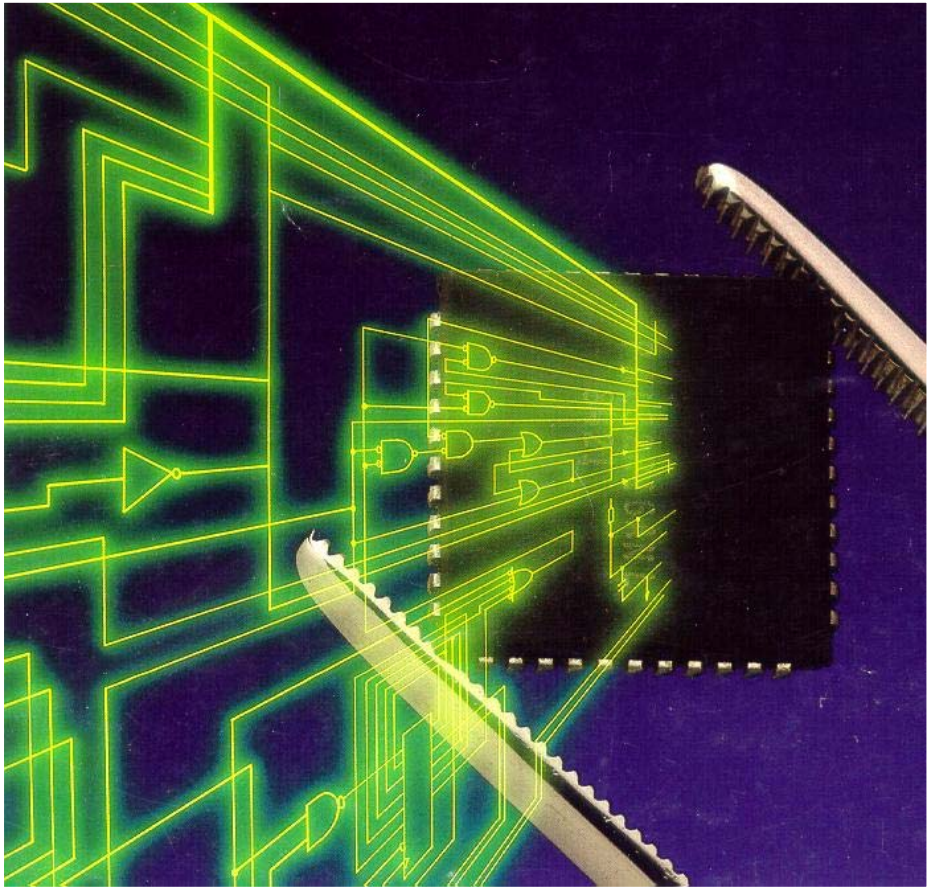
Beneficios de una solución universal

Cinvestav

- Fácil adaptabilidad a cambios de diseño:
  - Aumento de la vida comercial útil del producto
- Mayor desempeño:
  - Rápido, pequeño, confiable y fácil de armar
- Mejora en cuanto al aprovechamiento de los recursos de ingeniería:
  - Menor costo de desarrollo

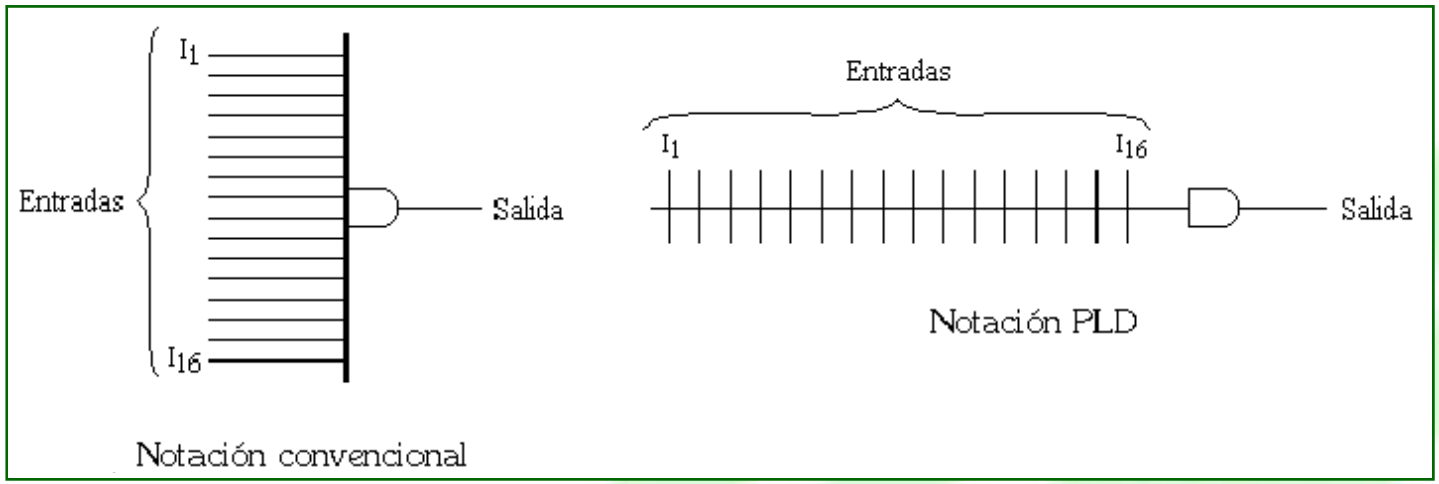
¿ Qué es un PLD ?

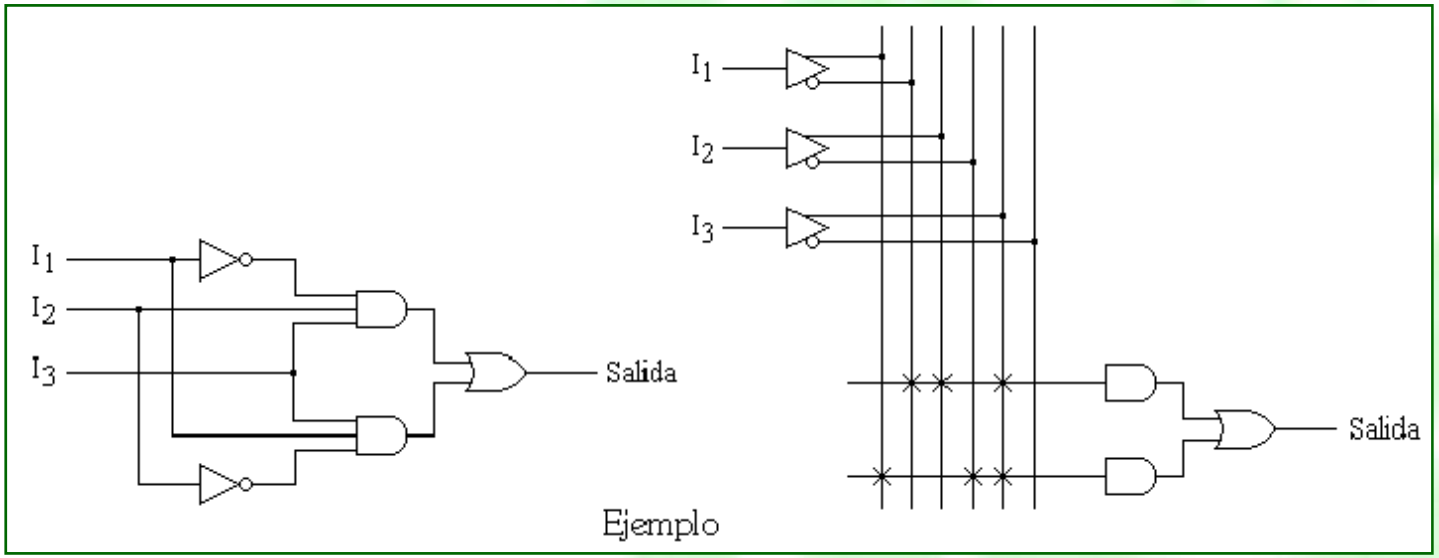
Cinvestav



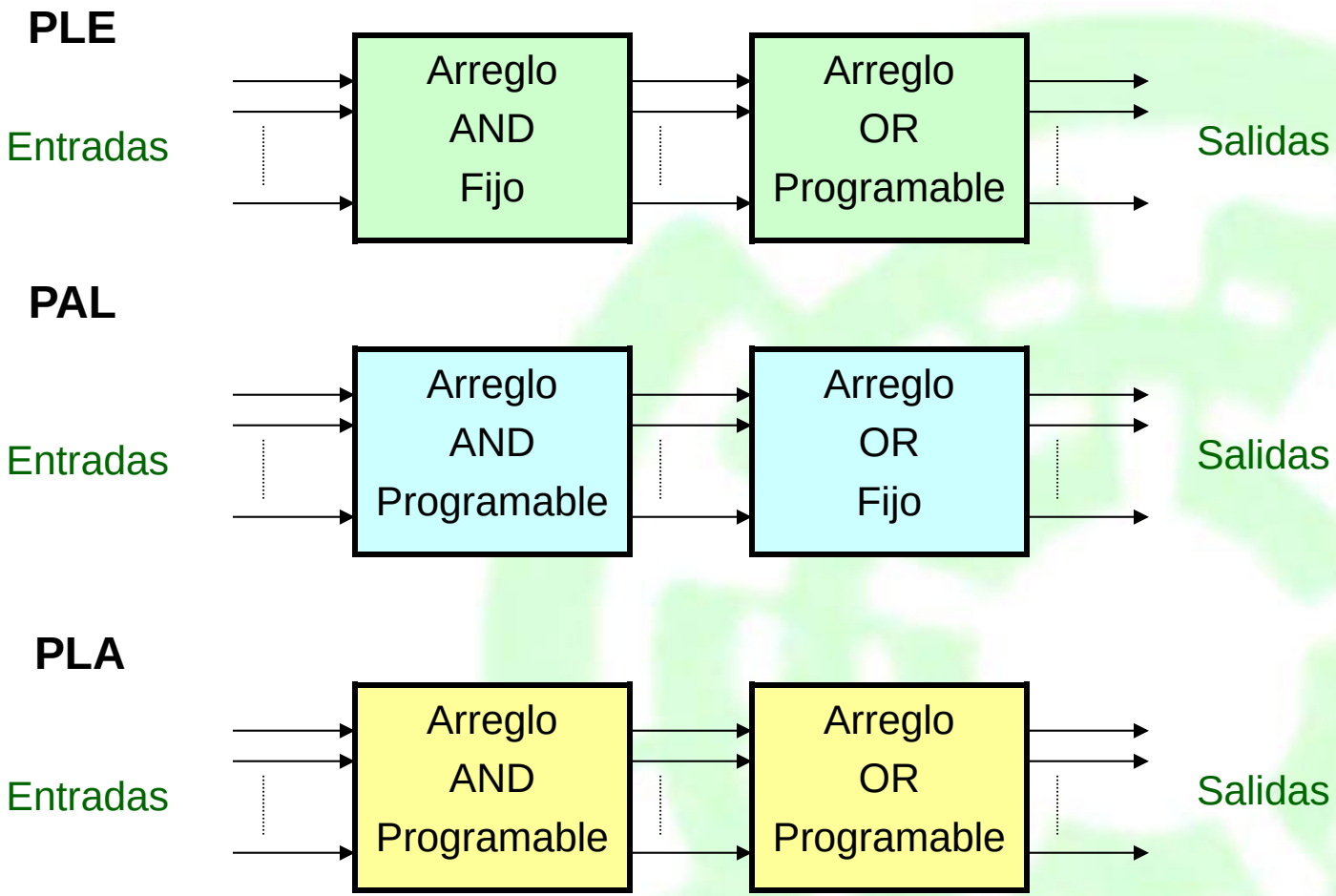
Es un circuito integrado que contiene una gran cantidad de elementos lógicos y a través de la programación se interconectan para que realicen una función específica.

Notación convencional y notación PLD



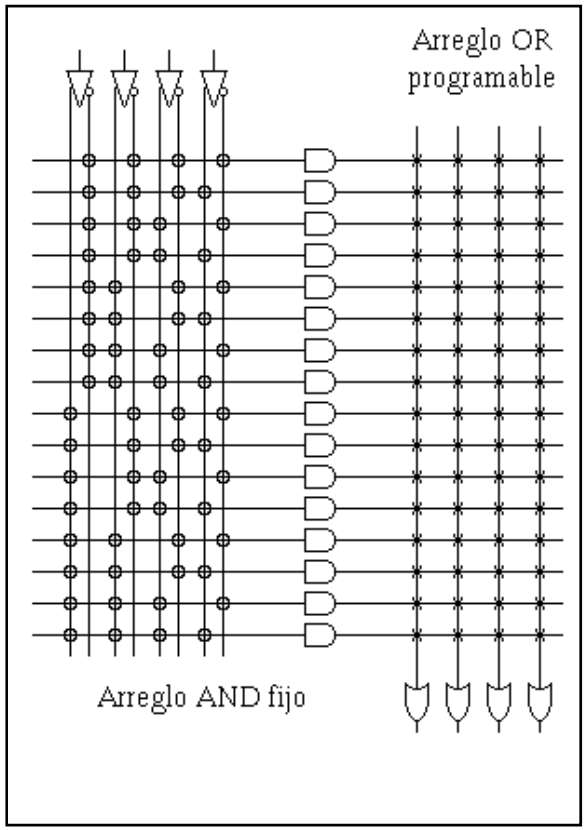


Configuraciones básicas

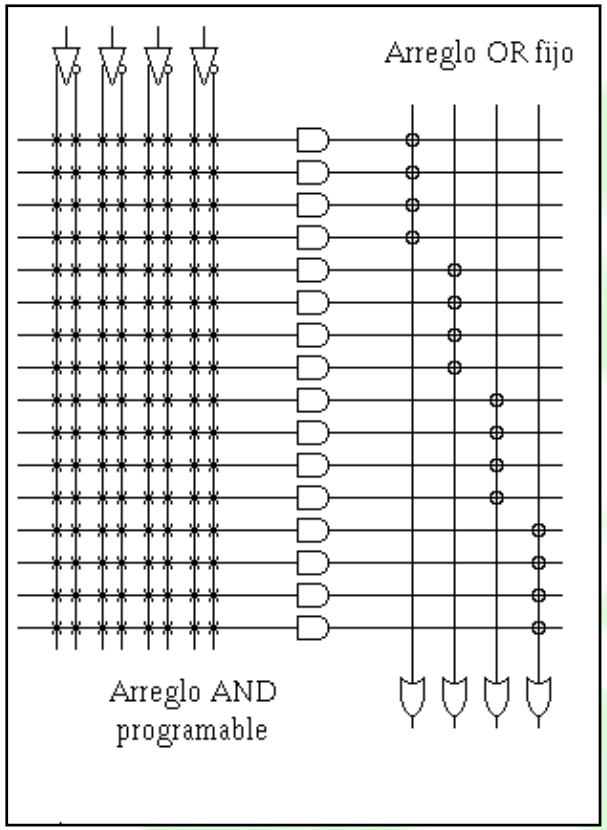


Configuraciones básicas

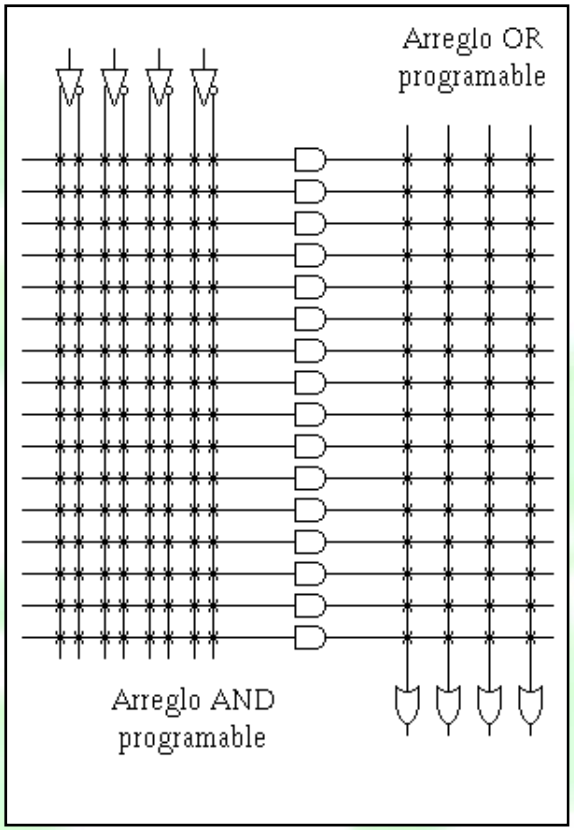
Cinvestav



PLE/PROM



PAL



PLA

Clasificación de los PLD's

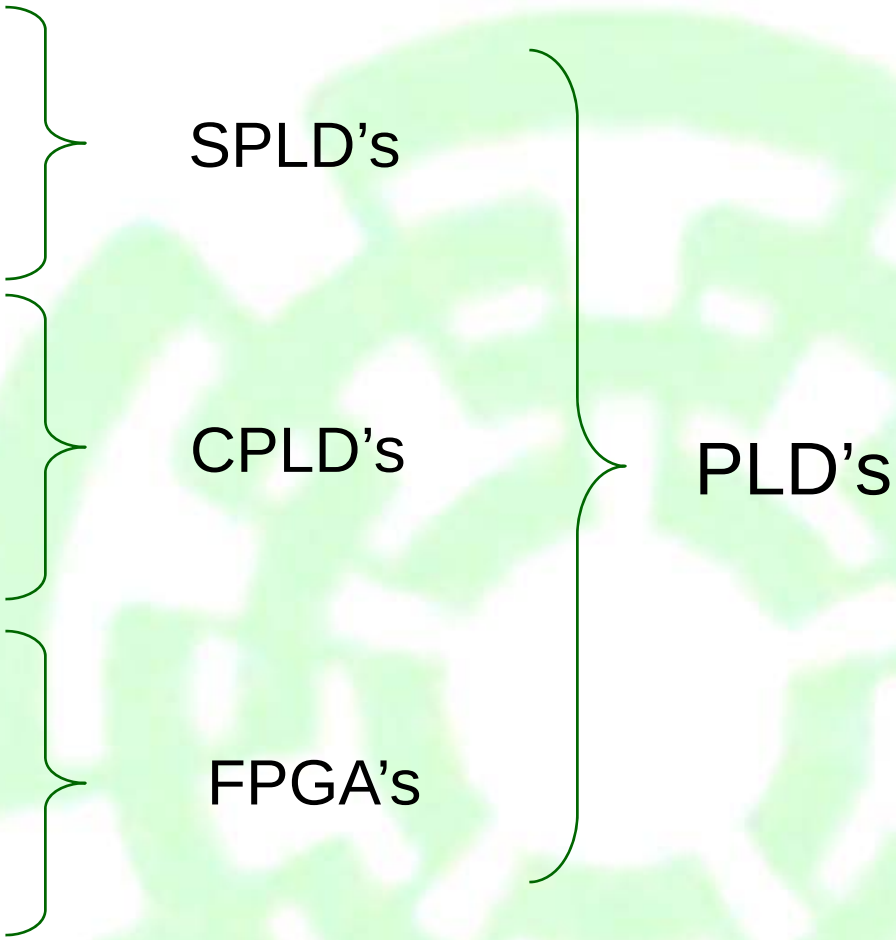
Cinvestav

Productos comerciales

Fabricante	SPLD
Altera	Clásicos, FLASHLogic
Atmel	PAL
Cypress	PAL
Lattice	GAL
Philips	PLA, PAL
Vantis	PAL

Fabricante	CPLD
Altera	MAX 5000,7000 y 9000
Atmel	ATF, ATV
Cypress	FLASH370, ULTRA37000
Lattice	IspLSI1000 a 8000
Philips	XPLA
Vantis	MACH 1 a 5
Xilinx	XC9500, CoolRunner

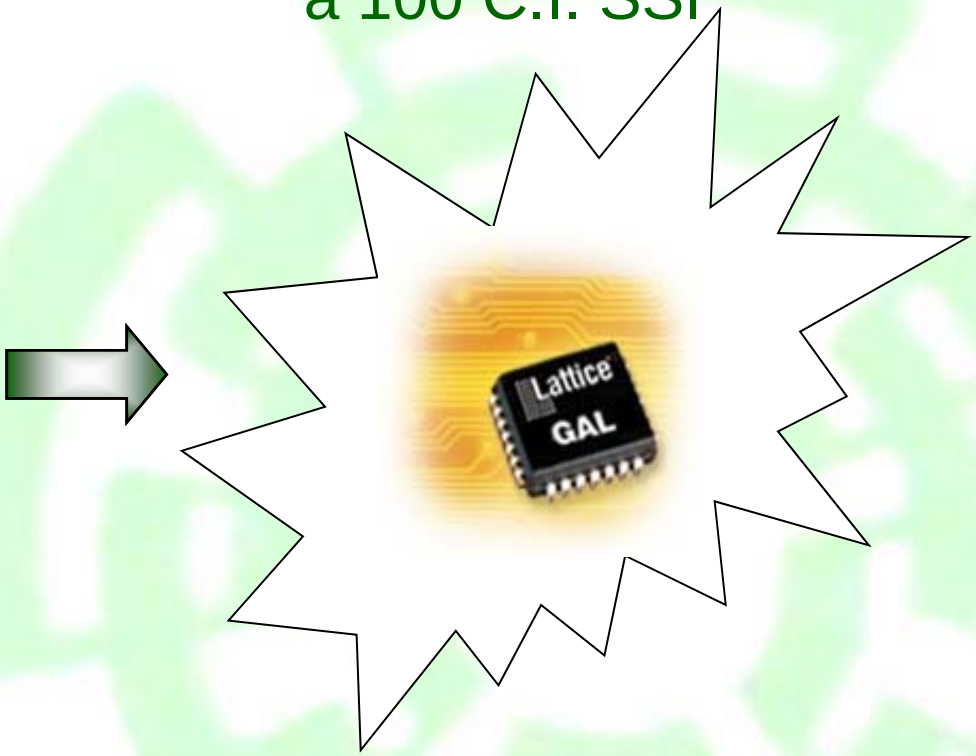
Fabricante	FPGA
Actel	ACT 1 a 3 , MX, SX
Altera	Flex, Cyclone, Stratix
Atmel	AT6000, AT40K
Lucent	ORCA 1 a 3
QuickLogic	pASIC1 a 3
Vantis	VF1
Xilinx	XC4000, Spartan, Virtex



Integración en un SPLD

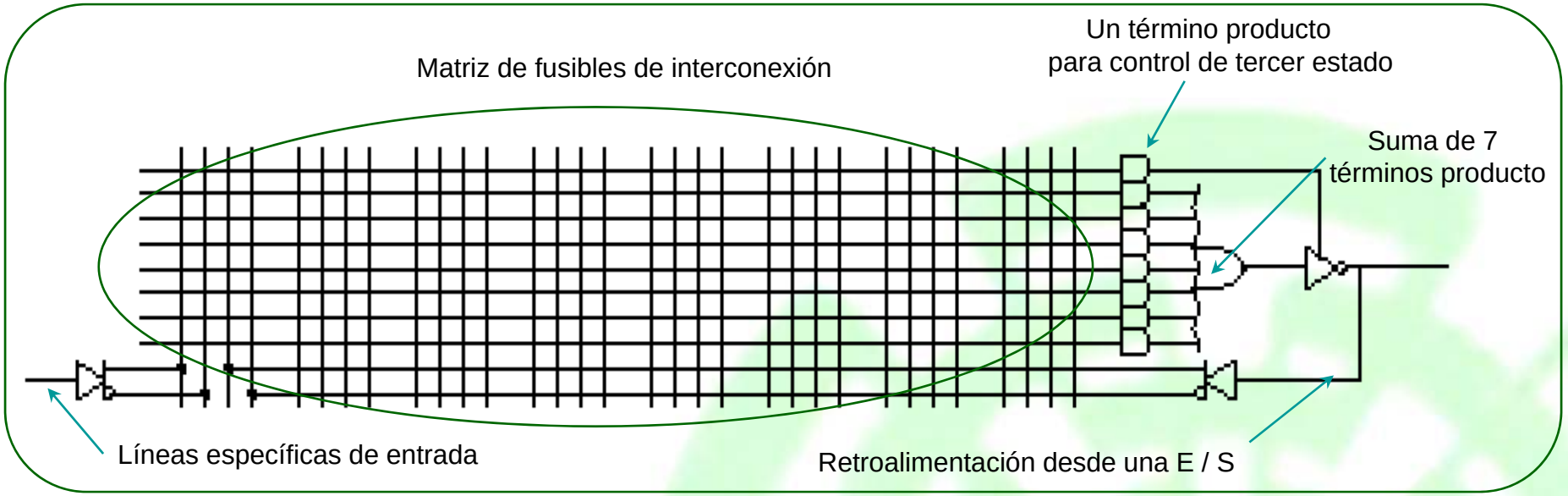
C.I. Series 74xx y 40xx

Sustituye
aproximadamente
a 100 C.I. SSI



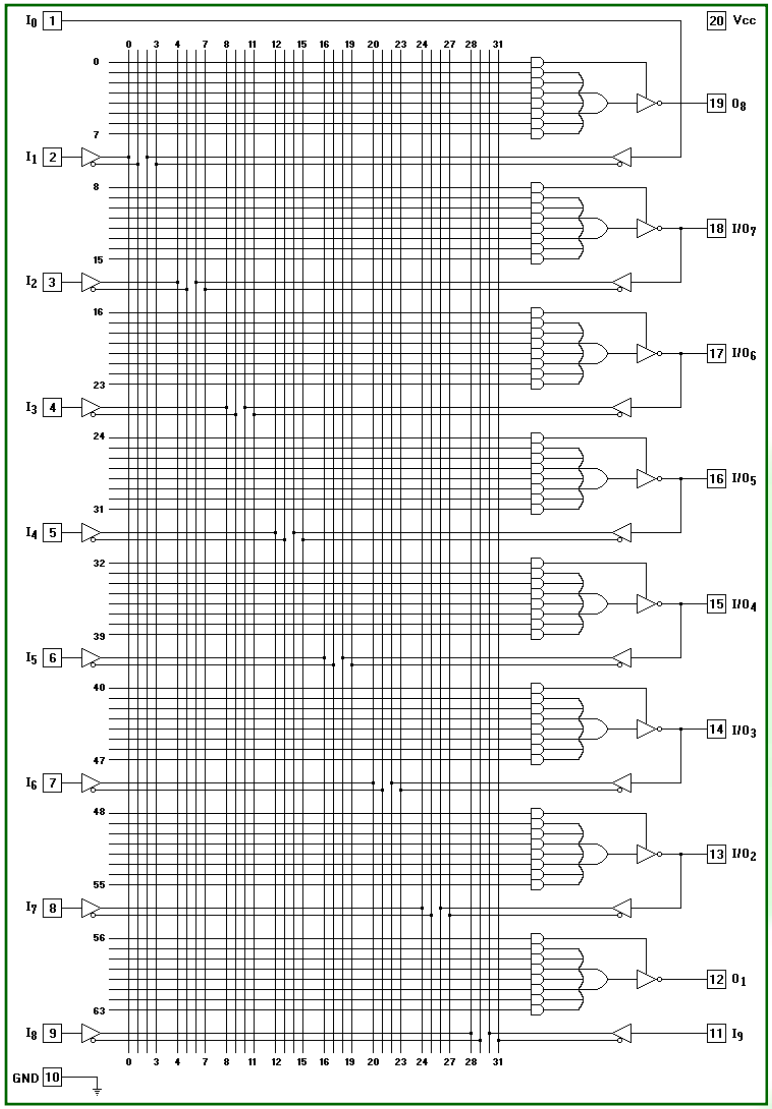
PAL's ó GAL's

Arquitectura PAL



- Diagrama esquemático de un PAL
  - Se cuenta a lo largo de **TODO EL CHIP** con las literales de todas las variables de entrada.
  - Mediante lógica alambrada es posible generar términos producto (**AND**) de la cantidad de literales que se desee.
  - Para generar la función solo es posible sumar (**OR**) hasta 7 u 8 términos producto.

PAL16L8



Características:

64 AND de 32 entradas

8 OR de 7 entradas

8 Inversores de tercer estado

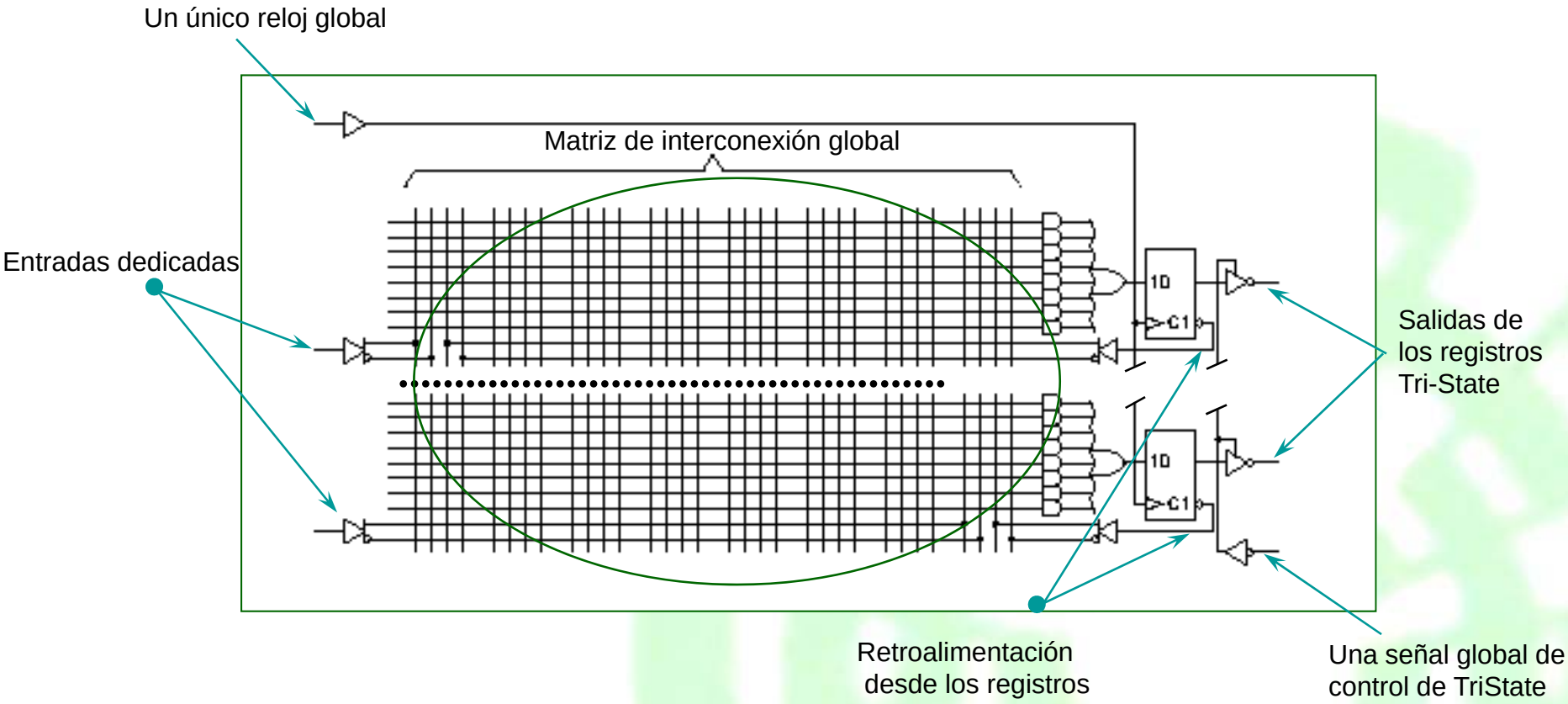
16 Buffers doble salida

Aproximadamente :

200 C.I. SSI (TTL o CMOS)

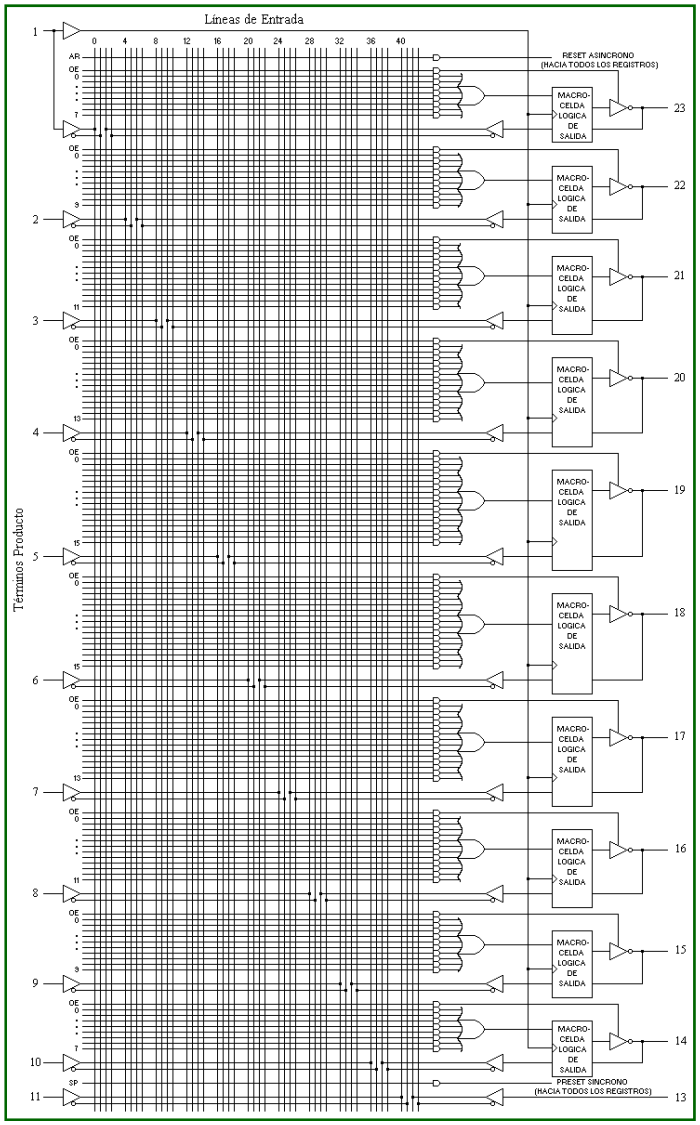
serie 74xx o 40xx

PAL16R8

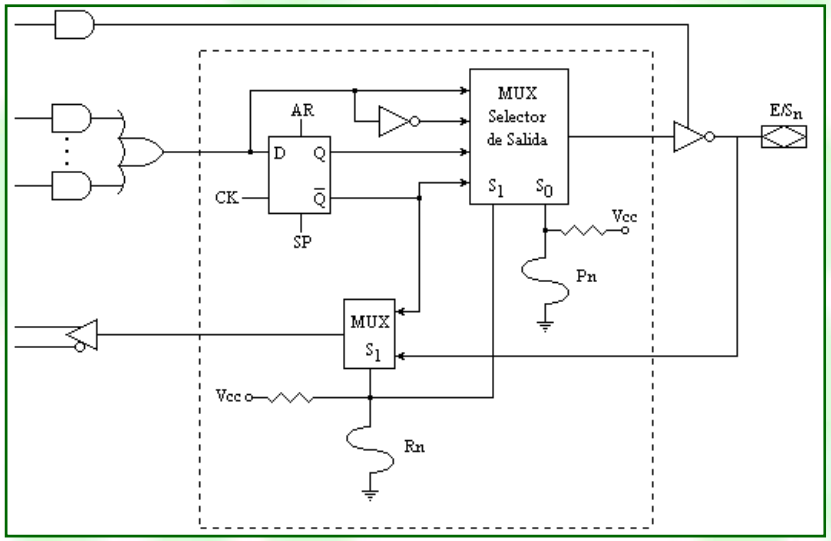


- Incorporación de elementos de memoria
  - Ideal para la síntesis de máquinas secuenciales

Arquitectura GAL



- Suma de 8 a 16 términos producto
- Macroceldas lógicas de salida



La macrocelda consta de:

- Un Flip-Flop
- Dos multiplexores

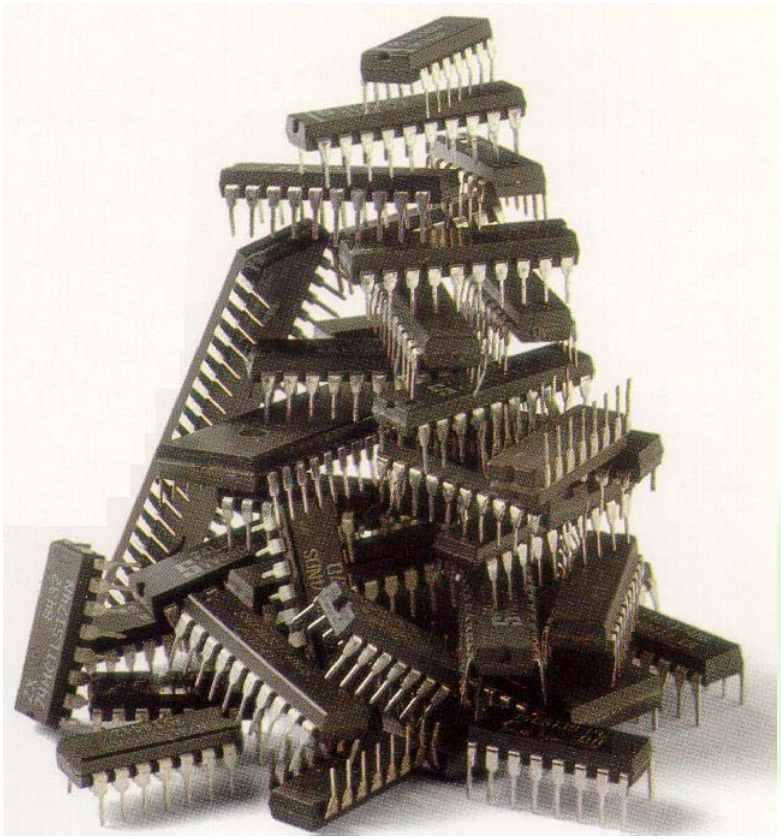
Limitaciones de los SPLD

Cinvestav

- Reducida cantidad de macroceldas.
- Cuando se utiliza el flip-flop de la macrocelda para realizar lógica enterrada se desperdicia una terminal de entrada/salida.
- La distribución de todas la señales por todo el chip consume mucha superficie del silicio y genera retardos capacitivos de importancia.
- En los primeros PAL, el uso de fusibles afectaba seriamente la confiabilidad del dispositivo.

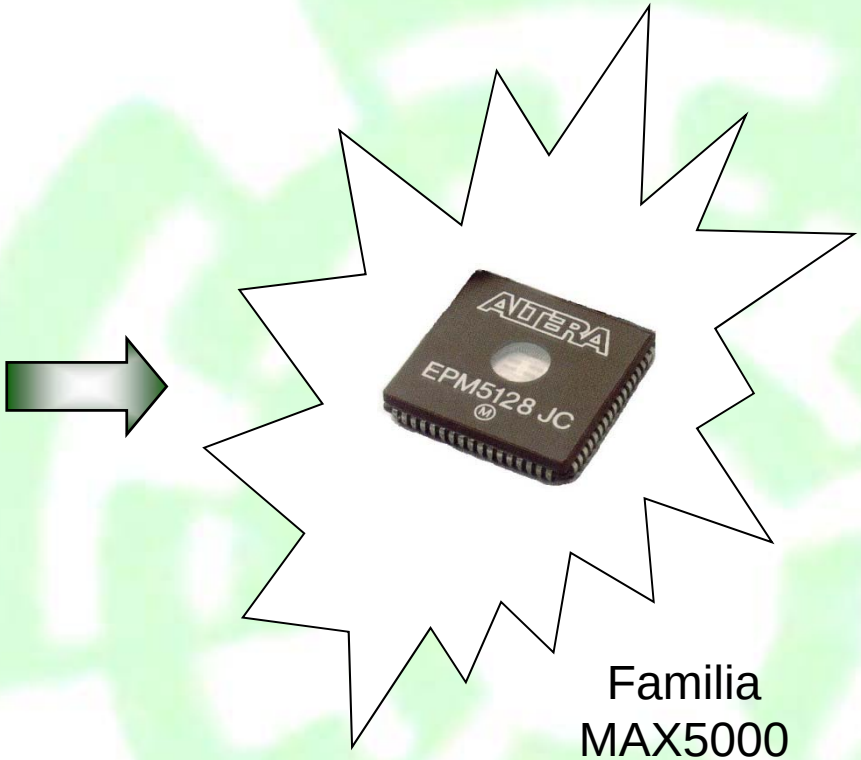
Integración en un CPLD

Cinvestav



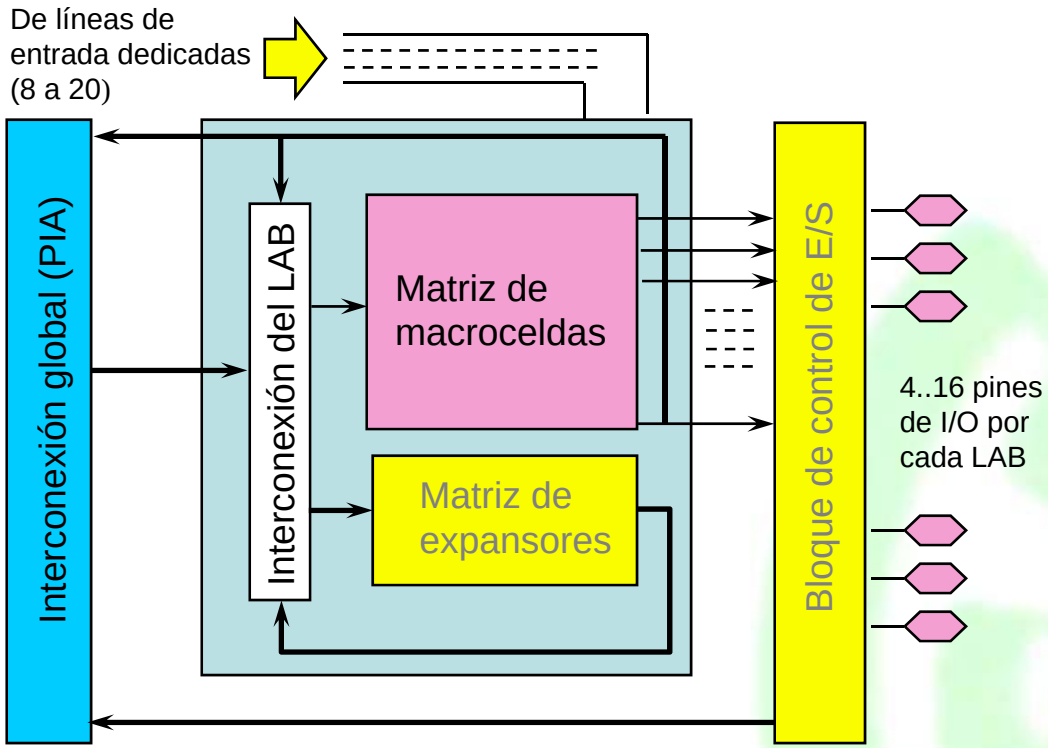
PAL's y GAL's

Sustituye
aproximadamente
a 50 SPLD's



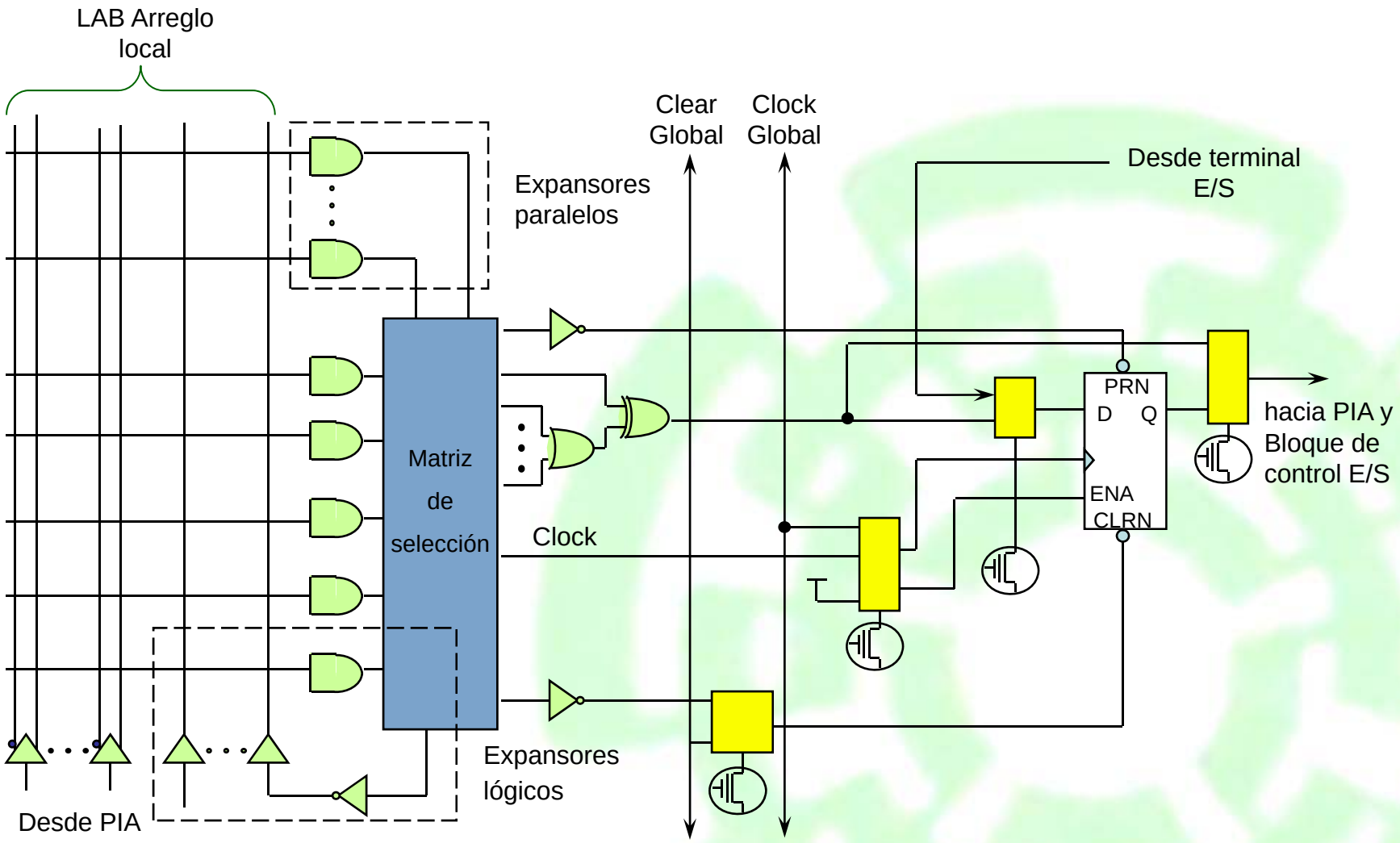
Familia
MAX5000

CPLD's



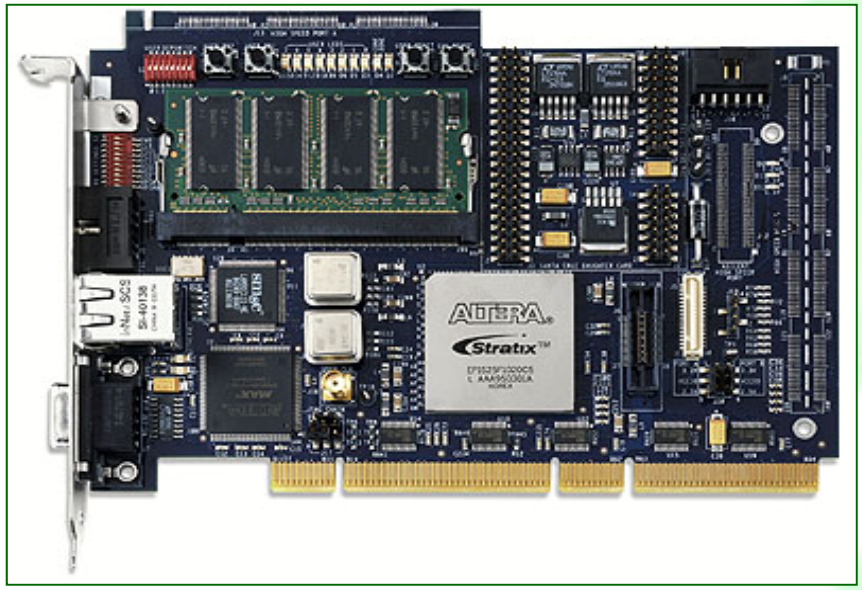
- Agrupamiento de las macroceldas **(LABs)**
- Generación de áreas de conexionado global **(PIA)**
- Generación de áreas de conexión dentro del **LAB**
- Expansores para generar términos producto auxiliares
- Con un bloque de E/S por cada macrocelda con **dual feedback**
- De 32 a 192 macroceldas en chips de 28 a 100 terminales

Macrocelda y Expansores



FPGA's

Cinvestav



➲ **F**ield **P**rogrammable **G**ate **A**rray
(Arreglo de Compuertas Programables en Campo)

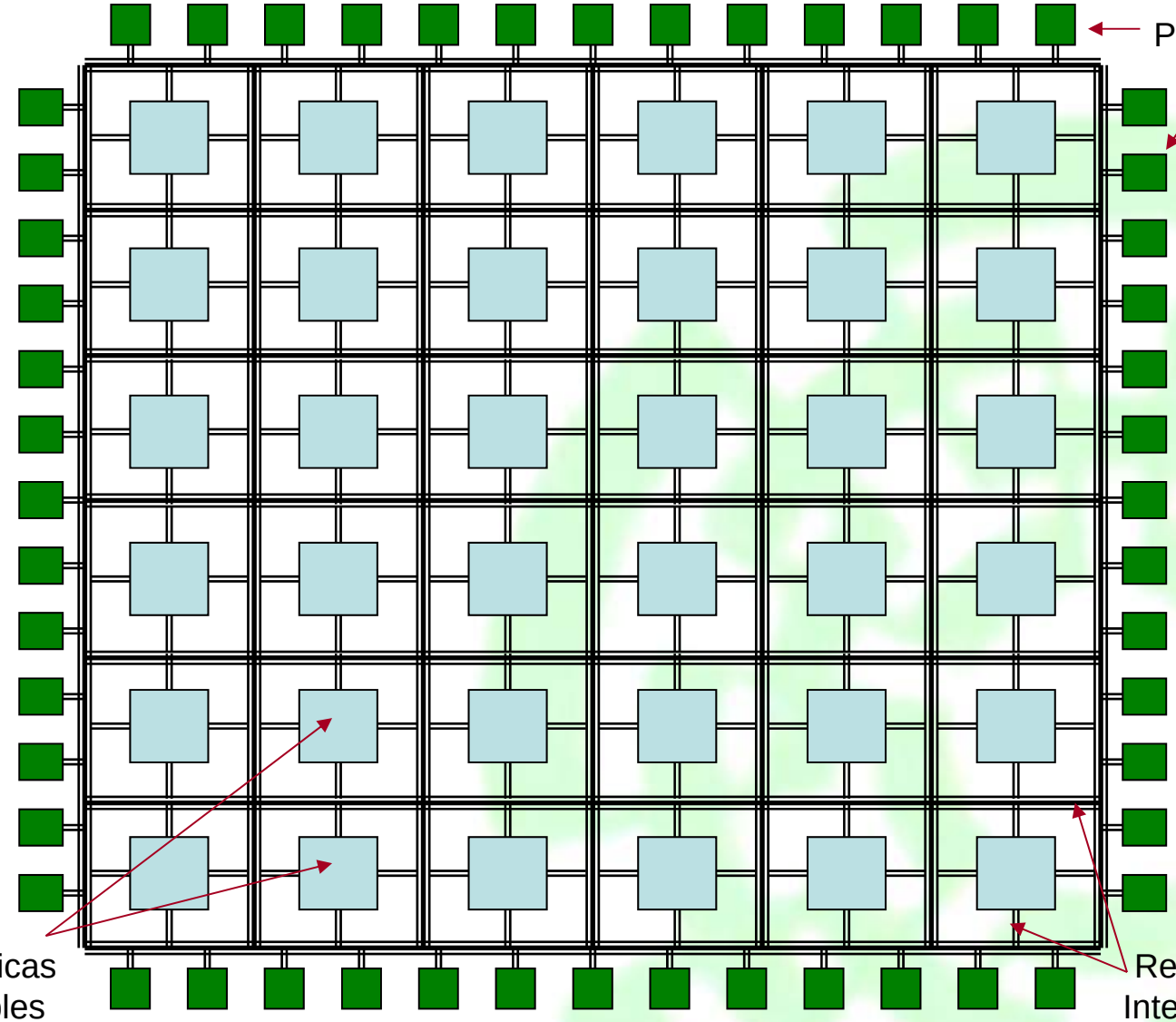
➲ Es un circuito integrado que contiene Celdas Lógicas Configurables

➲ Las Celdas Lógicas se interconectan por medio de una Matriz de Interconexión Programable

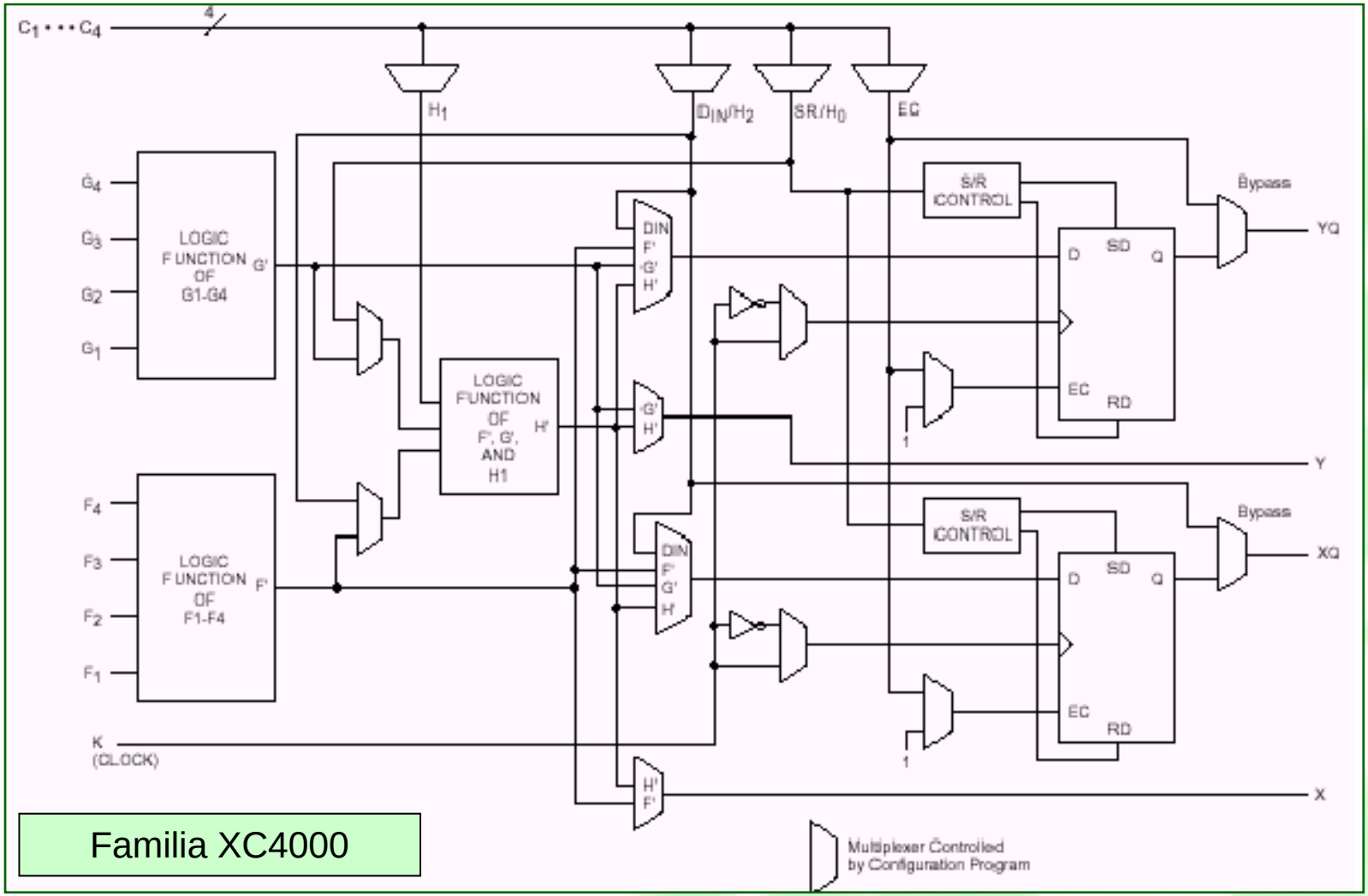
➲ Cuenta en su periferia con Puertos de Entrada/Salida.

Arquitectura del FPGA

Cinvestav



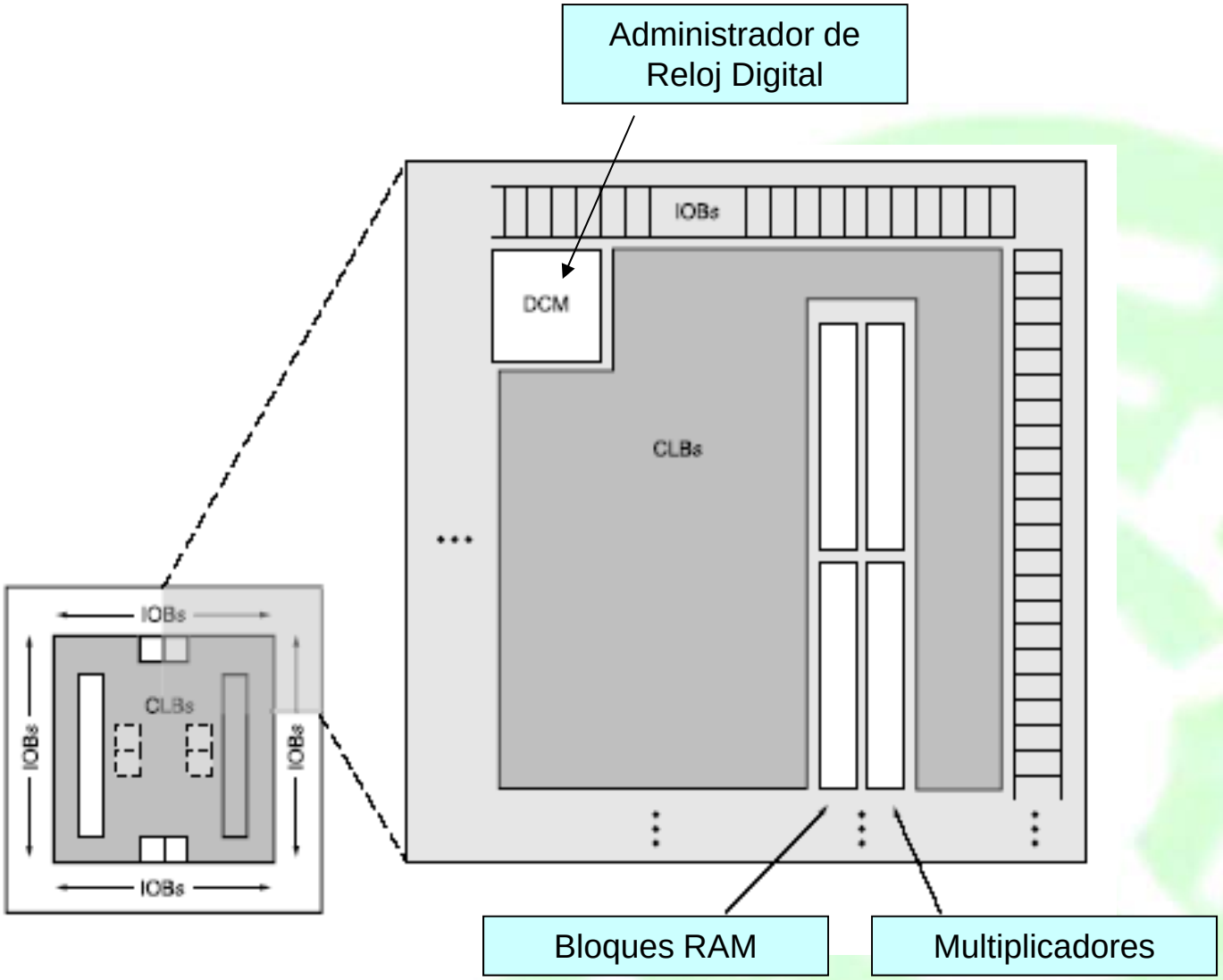
Celda Lógica del FPGA



Familia XC4000

FPGA de la Familia Spartan 3E

Cinvestav



Densidades de FPGA's

Cinvestav

Spartan II XC2S15	15,000*
Spartan IIE XC2S150E	150,000*
Virtex E XCV50E	72,000*
Virtex E XCV3200E	4,047,000*
Virtex II XC2V40	40,000*
Virtex II XC2V8000	8,000,000*

* Compuertas de sistema

Densidades de IP Cores

Cinvestav

Encriptador AES	40,000*
Microcontrolador 80530	130,000*
Microcontrolador 8051	150,000*
Decodificador Viterbi	190,000*
Controlador de Ethernet	195,000*
Decodificador JPEG color	780,000*

* Compuertas de sistema

Costos de FPGA's

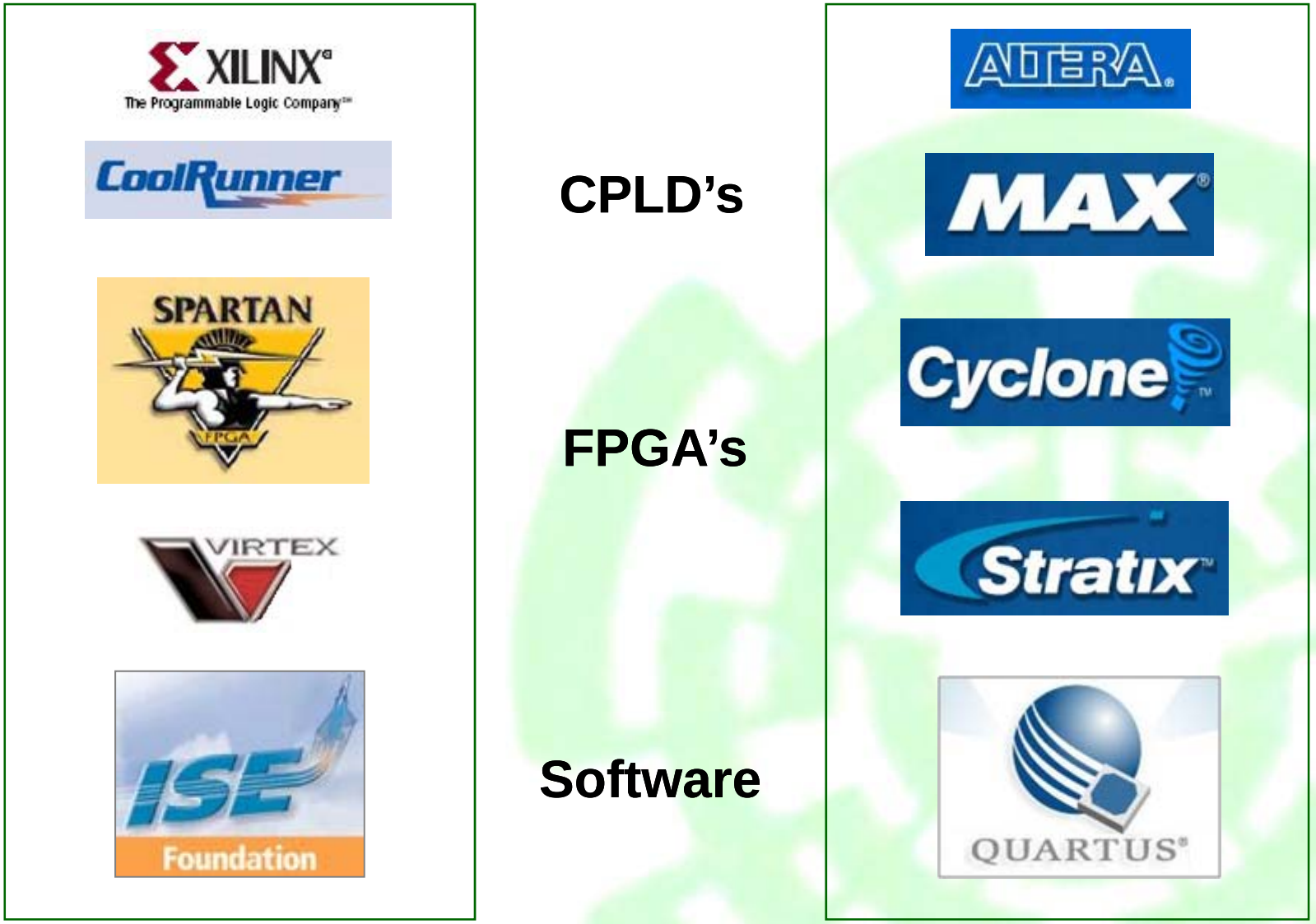
Cinvestav

Varían dependiendo del encapsulado y velocidad

- Spartan 20,000 compuertas ~ 1 DL
- Spartan 100,000 compuertas ~ 20 DLS
- Virtex 300,000 compuertas ~ 150 DLS
- Virtex II 8,000,000 compuertas ~ 8,000 DLS

Xilinx vs. Altera

Cinvestav



Diseño usando lógica programable

Cinvestav

Conclusiones :

- El uso de lógica programable no descarta el uso de lógica discreta, sino que la restringe a casos muy simples.
- Es una herramienta rápida, de alta confiabilidad, y de bajísimo costo por compuerta.
- La fácil modificación de un diseño permite asegurar el mantenimiento y actualización de un producto.
- Conocer profundamente las técnicas de diseño lógico es la mejor manera de aprovechar la lógica programable.
- Se pasa del diseño por compuertas al diseño por sistemas.

Flujo de Diseño para Lógica Programable

Cinvestav

Introducción/Descripción del Diseño

- Captura Esquemática
- Descripción basada en Lenguaje

Realización/Implementación del Diseño

- Translación/Síntesis del Diseño
- Verificación de Reglas de Diseño
- Partición y Mapeo de Lógica
- Asignación o Colocación (Place) de la Lógica en los Bloques configurables
- Enrutamiento (Route)
- Creación de Archivo de Programación

Programación-Dispositivo

Verificación del Diseño

Simulación Funcional

- Verificación de la funcionalidad de la Lógica
- Temporización estimada (opcional)

Simulación Temporizada

- Se requiere de información de temporización posterior a los procesos de Colocación (Place) y Enrutamiento (Route)

Análisis de Temporización Estático

- Se requiere de información de temporización posterior a los procesos de Colocación (Place) y Enrutamiento (Route)
- Determina retardos de trayectorias de manera rápida y reporta violación de restricciones.

Depuración del diseño integrado al Sistema (In-System) siendo desarrollado

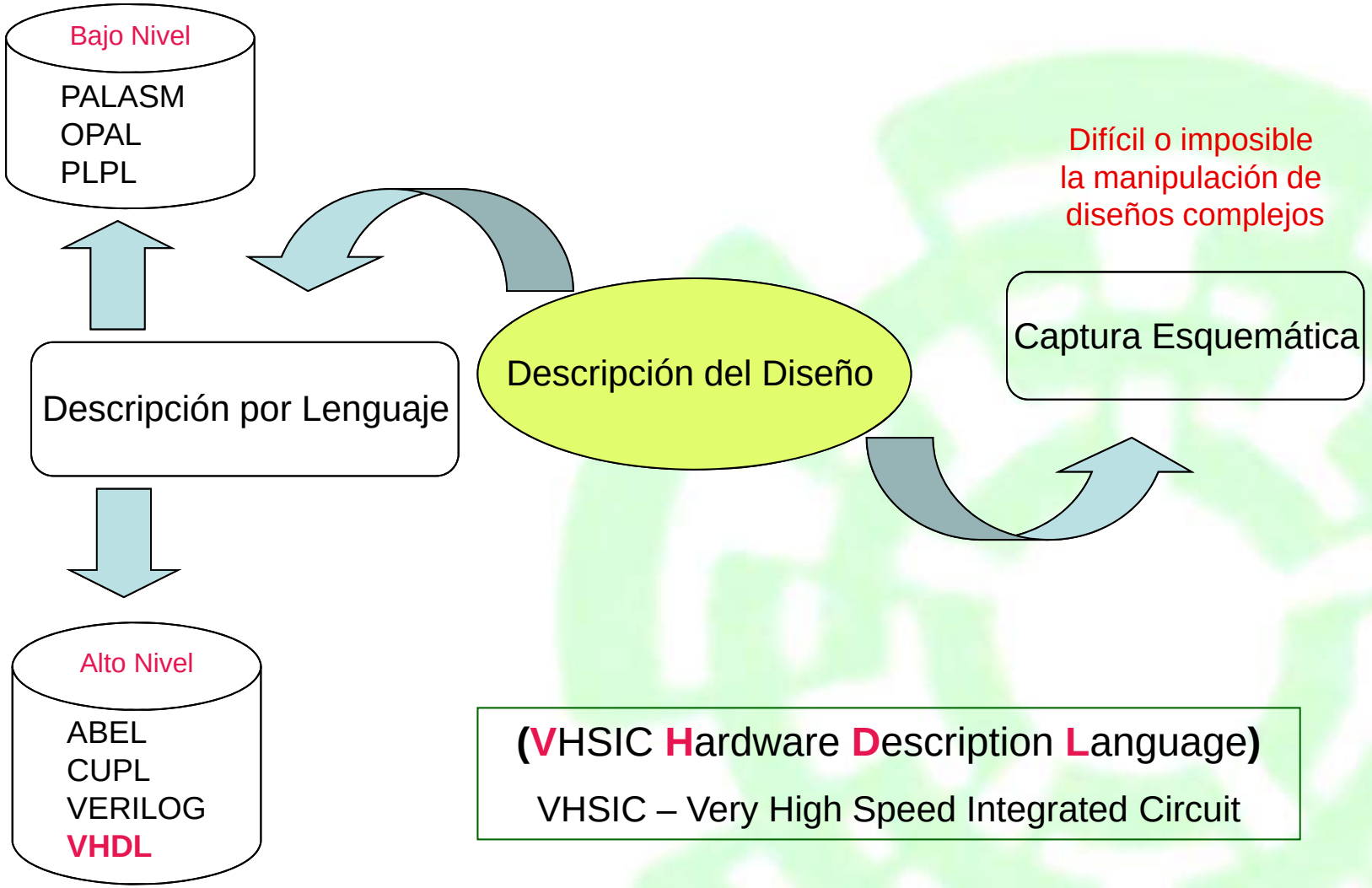
- Para dispositivos reprogramables
- Uso de otros Sistemas de Software & Hardware

Sistema Básico de Desarrollo

- **Computadora Personal / Estación de Trabajo**
- **Software CAE/CAD – p.ej. WebPack (Gratuito) de Xilinx**

- **Programador – Opcional**

Métodos de Descripción del Diseño



Cinvestav

Capítulo 2

Sintaxis del Lenguaje

Cinvestav

VHDL

(**V**HSIC **H**ardware **D**escription **L**anguage)

VHSIC – Very High Speed Integrated Circuit

Antecedentes de VHDL

Cinvestav

Desarrollo de Lenguajes para Descripción de Hardware

1970's

IDL/IBM, HDL/TI, ZEUS/GE
Desarrollo en Área Industrial

AHPL, DDL, CDL, ISPS
Desarrollo en Área Académica

1980's

AHDL, ABEL, CUPL

VHDL y Verilog

Departamento de la Defensa de los E.U.A.

Programa: Very High Speed Integrated Circuits (VHSIC)

1983 ➔ VHDL Desarrollo:

IBM, Texas Instruments e Intermetrics

Estándar IEEE VHDL

Estándar IEEE -1076-1987

Revisiones

Estándar IEEE -1076-1993
Estándar IEEE -1076-2000
Estándar IEEE -1076-2002
Estándar IEEE -1076-2008

Ventajas de VHDL

Cinvestav

Ventajas de VHDL
Notación Estandarizada
Disponibilidad al Público
Independencia del Sistema de Desarrollo (con algunas excepciones)
Independencia de la Metodología de Diseño (PLDs, ASICs, FPGAs)
Independencia de la Tecnología y Proceso de Fabricación (CMOS, Bipolar, BiCMOS)
Reutilización de Código
Capacidad descriptiva del comportamiento del sistema en distintos niveles de abstracción, modelación o abstracción: **Algoritmo**, **RTL** (Register Transfer Logic), **Lógico y Compuerta**
Facilitar la Verificación/Prueba del Sistema siendo diseñado.
Adición de la extensión analógica (IEEE1076.1) que permite la especificación, simulación y síntesis de sistemas digitales, analógicos y mixtos

Sintaxis de VHDL

Cinvestav

Elementos sintácticos de VHDL	
Comentarios	Se consideran comentarios después de dos guiones medios seguidos “--”
Símbolos especiales	Existen caracteres especiales sencillos como (&, #) o dobles como (:=, <=)
Identificadores	Es lo que se usa para dar nombre a los diferentes objetos del lenguaje
Números	Se considera que se encuentra en base 10, se admite la notación científica convencional es posible definir números en otras bases utilizando el símbolo # : 2#11000100#
Caracteres	Es cualquier letra o caracter entre comillas simples: ‘3’, ‘t’
Cadenas	Son un conjunto de caracteres englobados por comillas dobles: “hola”
Cadenas de bits	Los tipos bit y bit_vector son en realidad tipo caracter y arreglo de caracteres respectivamente, se coloca un prefijo para indicar la base : O”126”, X”FE”
Palabras reservadas	Son las instrucciones, órdenes y elementos que permiten definir sentencias.

Reglas para Identificadores

Cinvestav

Identificadores
Nombres o etiquetas que se usan para referirse a: Variables, Constantes, Señales, Procesos, Entidades, etc.
Están formados por números, letras (mayúsculas o minúsculas) y guión bajo “_” con las reglas especificadas en la tabla siguiente.
Longitud (Número de Caracteres): Sin restricciones
Palabras reservadas por VHDL no pueden ser identificadores
En VHDL, un identificador en mayúsculas es igual a su contraparte en minúsculas

Reglas para especificar un identificador	Incorrecto	Correcto
Primer caracter debe ser siempre una letra mayúscula o minúscula	4Suma	Suma4
Primer caracter no puede ser un guión bajo (_)	_S4bits	S_4bits
Dos guiones bajos seguidos no son permitidos	Resta__4	Resta_4_
Un identificador no puede utilizar símbolos especiales	Clear#8	Clear_8

Palabras reservadas en VHDL

Cinvestav

Lista de palabras reservadas en VHDL				
abs	downto	library	postponed	subtype
access	else	linkage	procedure	then
after	elsif	literal	process	to
alias	end	loop	pure	transport
all	entity	map	range	type
and	exit	mod	record	unaffected
architecture	file	nand	register	units
array	for	new	reject	until
assert	function	next	rem	use
attribute	generate	nor	report	variable
begin	generic	not	return	wait
block	group	null	rol	when
body	guarded	of	ror	while
buffer	if	on	select	with
bus	impure	open	severity	xnor
case	in	or	shared	xor
component	inertial	others	signal	
configuration	inout	out	sla	
constant	is	package	sra	
disconnect	label	port	srl	

Precedencia de Operadores

Cinvestav

Mayor

Menor

Precedencia de Operadores					
**	ABS	NOT			
*	/	MOD	REM		
+ (signo)	- (signo)				
+	-	&			
=	/=	<	<=	>	>=
AND	OR	XOR	NAND	NOR	XNOR

La precedencia de operadores se encuentran ordenados de mayor (arriba) a menor (abajo), los operadores que se encuentran en la misma fila tienen la misma precedencia.

PRECEDENCIA DE OPERADORES

Los operadores anteriores se definen de la siguiente manera:

1. Operadores lógicos binarios: **and or nand nor xor xnor**.
2. Operadores relacionales: = /= < <= > >=.
3. Operadores de desplazamiento: **sll srl sla sra rol ror**.
4. Operadores de adición: + - & (concatenación).
5. Operadores de signo: + -
6. Operadores de multiplicación: * / **mod rem**.
7. Operadores misceláneos: **not abs** **

Cuando no se usan los paréntesis, los operadores de la clase 7 tienen la mayor precedencia y se aplican primero, seguidos en ésta por los de la clase 6, luego la 5 y así sucesivamente. Los de la clase 1 tienen la menor precedencia y se aplican al último. Los operadores de la misma precedencia se aplican de izquierda a derecha en la expresión.

El orden de la precedencia puede cambiarse mediante los paréntesis.

PRECEDENCIA DE OPERADORES

Ejercicio:

Realizar las siguientes operaciones al vector ***A*** = "**10010101**":

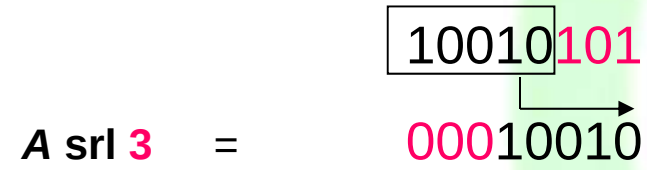
***A* sll 2** (desplazamiento lógico hacia la izquierda llenando con ceros "0"):

PRECEDENCIA DE OPERADORES

Ejercicio:

Realizar las siguientes operaciones al vector ***A*** = "**10010101**":

***A* srl 3** (desplazamiento lógico hacia la derecha llenando con ceros "0"):

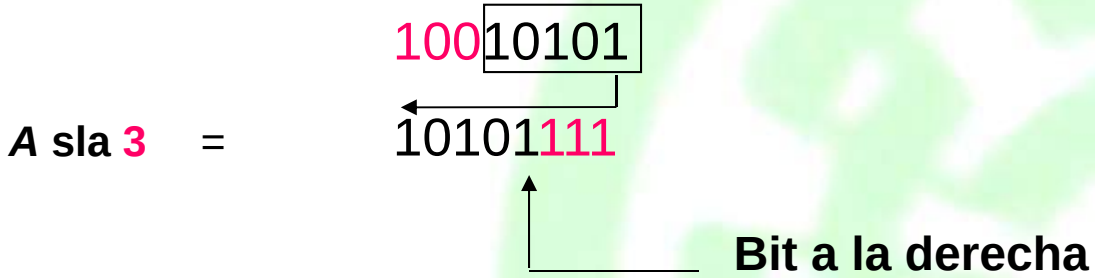


PRECEDENCIA DE OPERADORES

Ejercicio:

Realizar las siguientes operaciones al vector ***A*** = "**10010101**":

***A* sla 3** (desplazamiento aritmético hacia la izquierda llenando con el bit a la derecha):

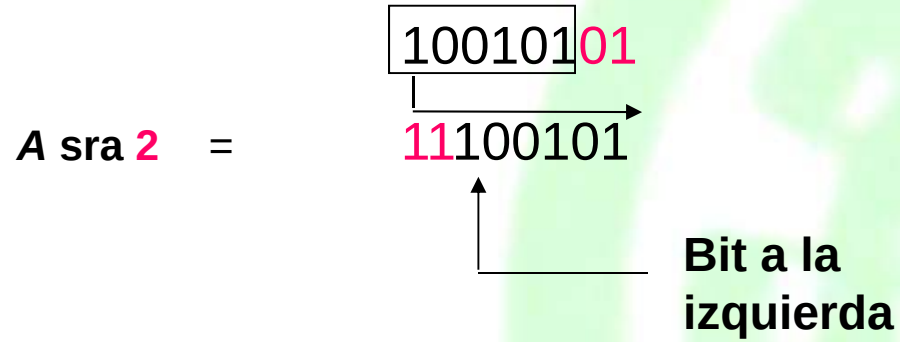


PRECEDENCIA DE OPERADORES

Ejercicio:

Realizar las siguientes operaciones al vector ***A*** = "**10010101**":

***A* sra 2** (desplazamiento aritmético hacia la derecha llenando con el bit a la izquierda):

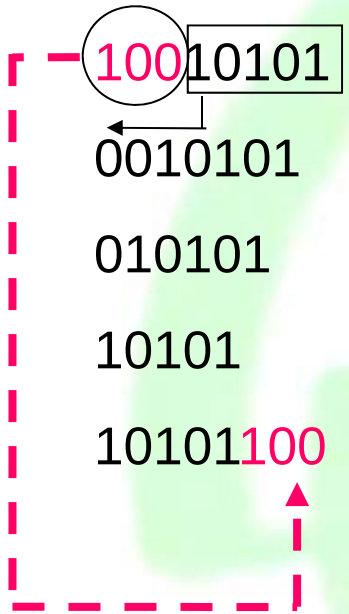


PRECEDENCIA DE OPERADORES

Ejercicio:

Realizar las siguientes operaciones al vector ***A*** = "**10010101**":

***A* rol 3** (rotación a la izquierda):



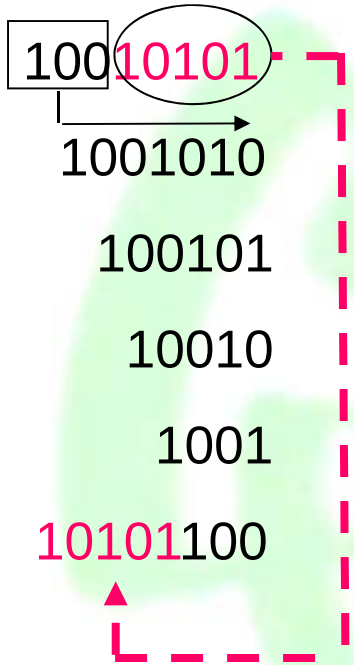
***A* rol 3** =

PRECEDENCIA DE OPERADORES

Ejercicio:

Realizar las siguientes operaciones al vector ***A*** = "**10010101**":

***A* ror 5** (rotación a la derecha):



***A* ror 5** =

PRECEDENCIA DE OPERADORES

Ejercicio: En la siguiente expresión, ***A***, ***B***, ***C*** y ***D*** son del tipo *bit_vector*.

(***A*** & **not** ***B*** or ***C*** **ror** 2 **and** ***D***) = "110010"

Entonces, los operadores se aplicarán en el siguiente orden:

not, &, **ror**, **or**, **and**, =

Si ***A***= "110", ***B***= "111", ***C***= "011000" y ***D***= "111011", las operaciones se realizan como se muestra a continuación:

not ***B*** = "000" (complemento bit por bit)

A & **not** ***B*** = "110000" (concatenación)

C **ror** 2 = "000110" (rotación a la derecha dos lugares)

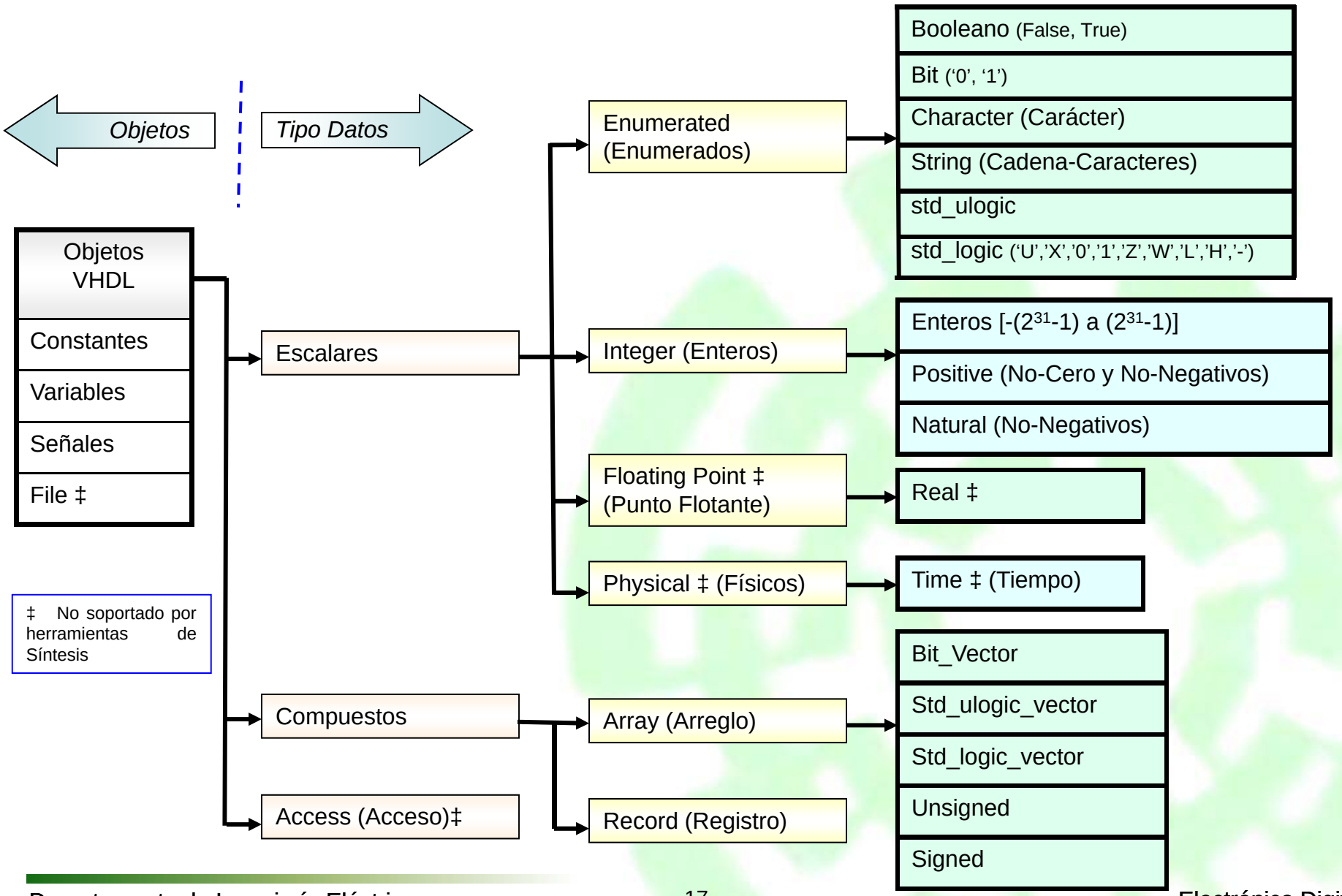
(***A*** & **not** ***B***) **or** (***C*** **ror** 2) = "110110" (operación **or** bit por bit)

(***A*** & **not** ***B*** **or** ***C*** **ror** 2) **and** ***D*** = "110010" (operación **and** bit por bit)

[(***A*** & **not** ***B*** **or** ***C*** **ror** 2 **and** ***D***) = "110010] = **TRUE** (el paréntesis fuerza a la prueba de igualdad al final, resultando en verdadero (**TRUE**))

Op. lógicos bin.: **and or nand nor xor xnor**.
Operadores relacionales: = /= < <= > >=.
Op. de desplazamiento: **sll srl sla sra rol ror**.
Operadores de adición: + - & (concatenación).
Operadores de signo: + -
Operadores de multiplicación: * / **mod rem**.
Operadores misceláneos: **not abs** **

Objetos y Tipos de Datos



Números Reales:

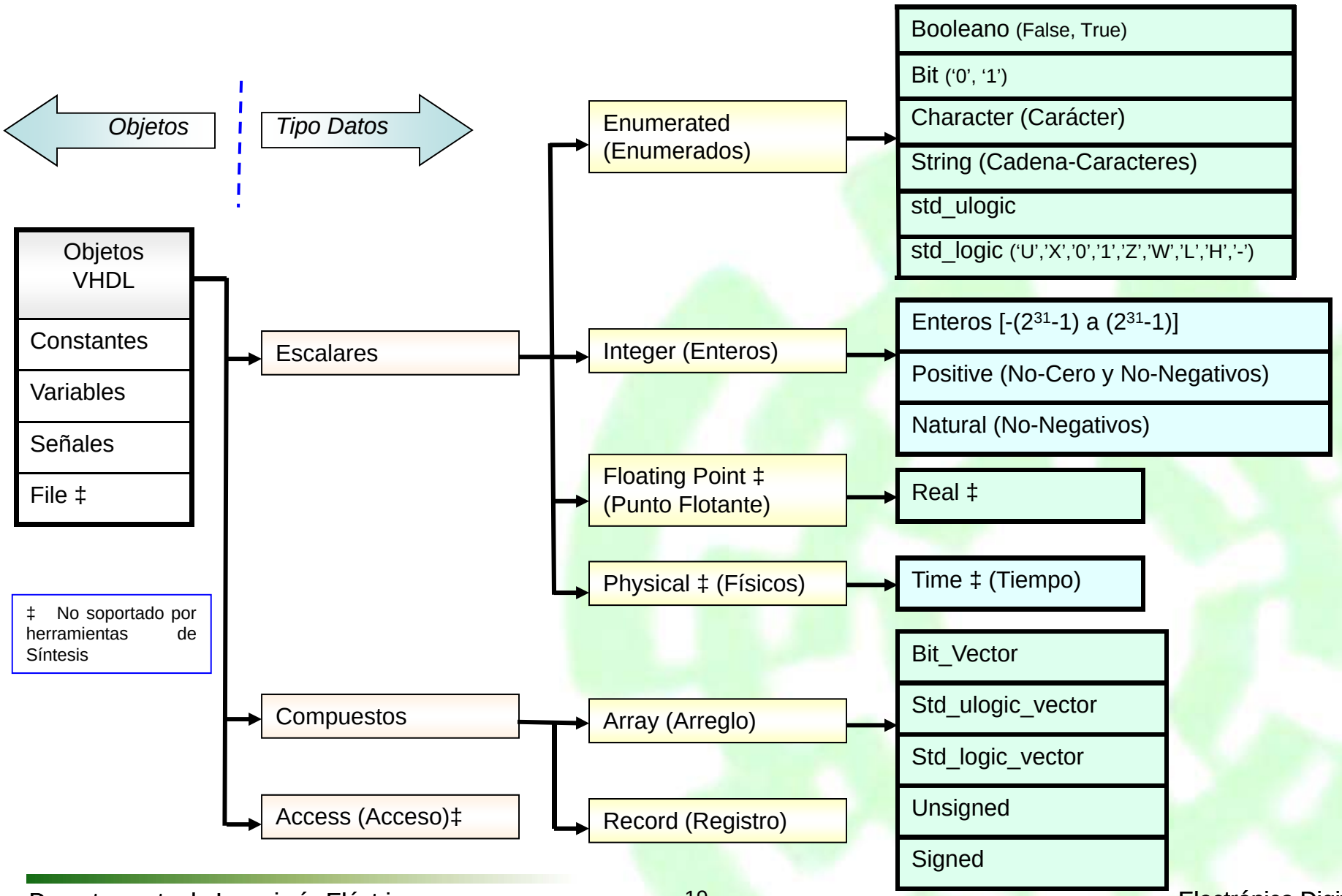
Racionales: -3/4; 5/8; 31/9;….

Enteros: -7; -1; 0; 5; 20;….

Irracionales: $\sqrt{2}$; $(1+\sqrt{5})/2$;….

Trascendentes: e ; π; ln(2);….

Objetos y Tipos de Datos



http://www.csee.umbc.edu/help/VHDL/stdpkg.html

Objetos de Datos

Cinvestav

Un objeto de datos en VHDL es un elemento que toma un valor de algún tipo de dato determinado, según sea el tipo de dato, el objeto poseerá un conjunto de propiedades. En VHDL los objetos de datos son generalmente una de las tres clases siguientes:

Constantes

Una constante es un elemento que puede tomar un único valor de un tipo de dato; las constantes pueden ser declaradas dentro de entidades, arquitecturas, procesos y paquetes.

CONSTANT identificador : tipo := valor;

Ejemplo

CONSTANT byte: integer := 8;

Variables

Las variables pueden ser modificadas cuando sea necesario, pueden ser declaradas solamente dentro de los procesos y subprogramas.

VARIABLE identificador : tipo [:= valor];

Ejemplo

VARIABLE aux1, aux2: bit;

Señales

Las señales sí pueden almacenar o pasar valores lógicos, por lo tanto, representan elementos de memoria o conexiones y si pueden ser sintetizadas. Son declaradas en las arquitecturas antes del **BEGIN**.

SIGNAL identificador : tipo [:= valor];

Ejemplo

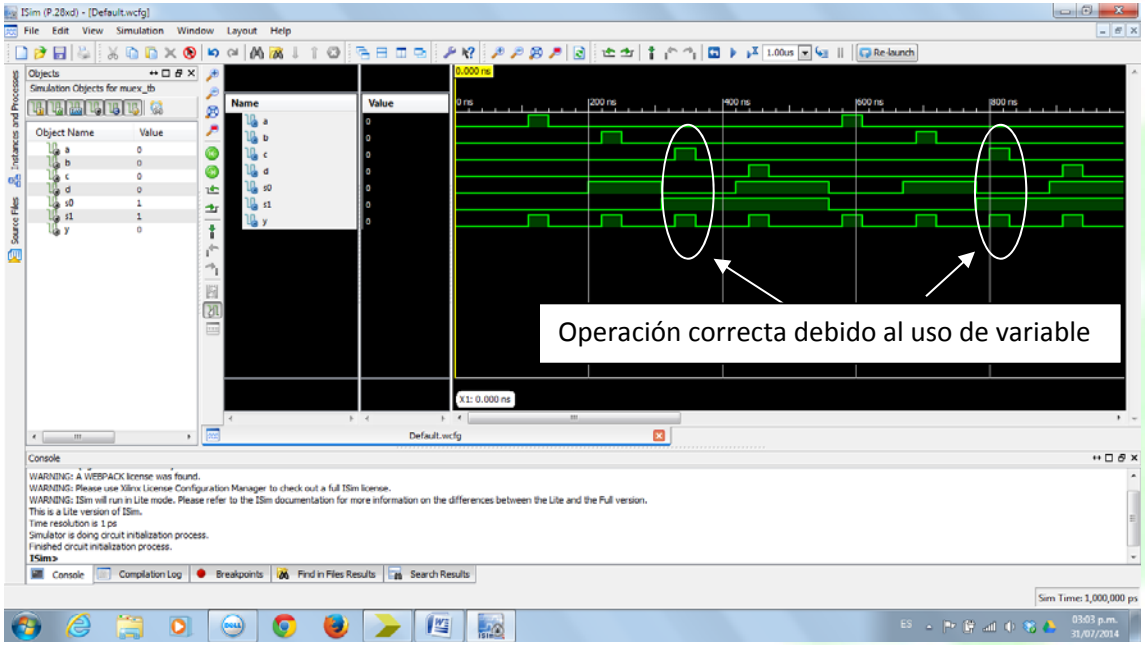
SIGNAL A, B : bit := '0';

SIGNAL dato: bit_vector (7 **downto** 0);

Objetos de Datos

Cinvestav

Es importante conocer las implicaciones que tiene el uso de señales y variables en la respuesta del circuito que se quiere sintetizar.

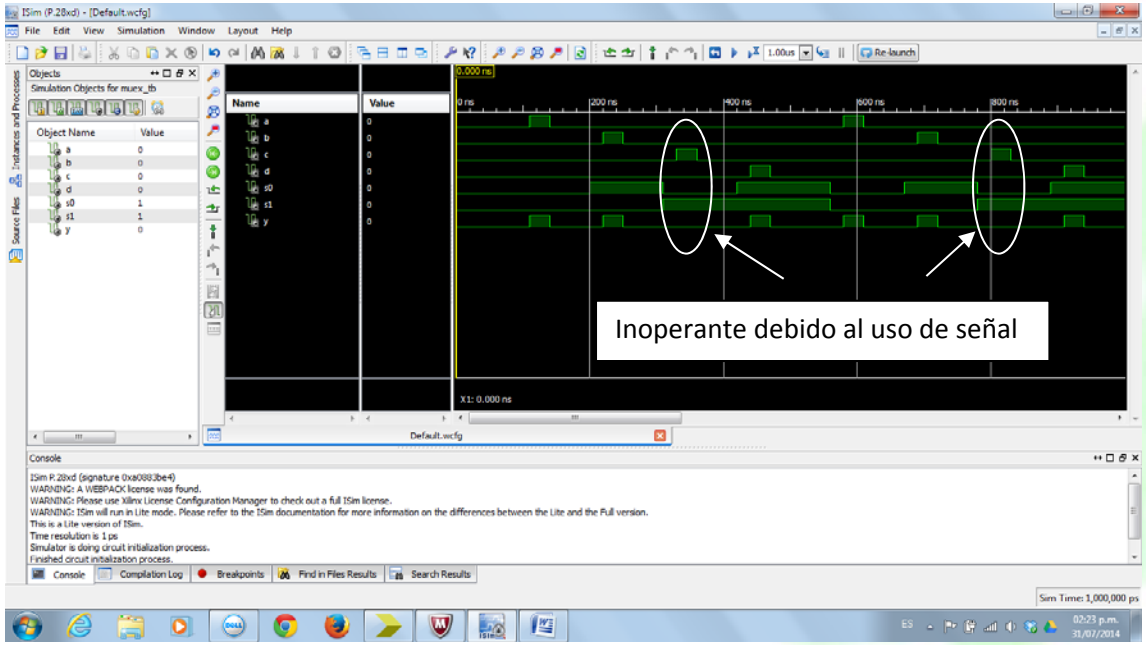



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux_OK is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           d : in  STD_LOGIC;
          s0 : in  STD_LOGIC;
          s1 : in  STD_LOGIC;
           y : out  STD_LOGIC);
end Mux_OK;

architecture Behavioral of Mux_OK is
begin
      process (a, b, c, d, s0, s1)
          variable sel : integer range 0 to 3;
Begin
                        sel := 0;
                        if (s0 = '1') then sel := sel + 1;
                        end if;
                        if (s1 = '1') then sel := sel + 2;
                        end if;


            case sel is
                        when 0 => y <= a;
                        when 1 => y <= b;
                        when 2 => y <= c;
                        when 3 => y <= d;
             end case;
 end process;

end Behavioral;
```

Objetos de Datos

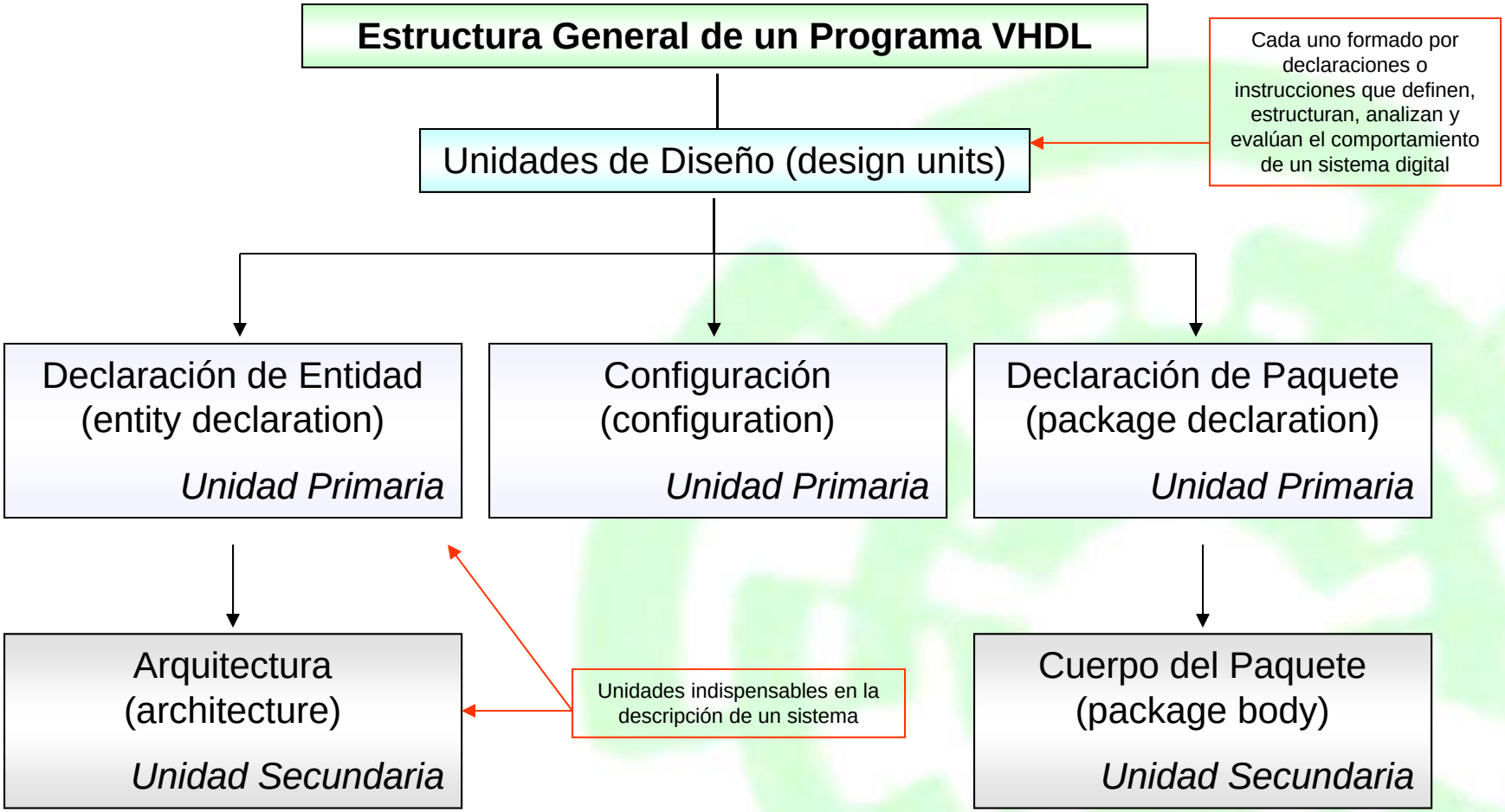
Cinvestav

Es importante conocer las implicaciones que tiene el uso de señales y variables en la respuesta del circuito que se quiere sintetizar.




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Mux_not_OK is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           d : in  STD_LOGIC;
           s0 : in  STD_LOGIC;
           s1 : in  STD_LOGIC;
           y : out  STD_LOGIC);
end Mux_not_OK;

architecture Behavioral of Mux_not_OK is
signal sel: integer range 0 to 3;
begin
        process (a, b, c, d, s0, s1)
Begin
        sel <= 0;
            if (s0 = '1') then sel <= sel + 1;
            end if;
            if (s1 = '1') then sel <= sel + 2;
            end if;

        case sel is
                    when 0 => y <= a;
                    when 1 => y <= b;
                    when 2 => y <= c;
                    when 3 => y <= d;
        end case;
end process;

end Behavioral;
```

Estructura de VHDL



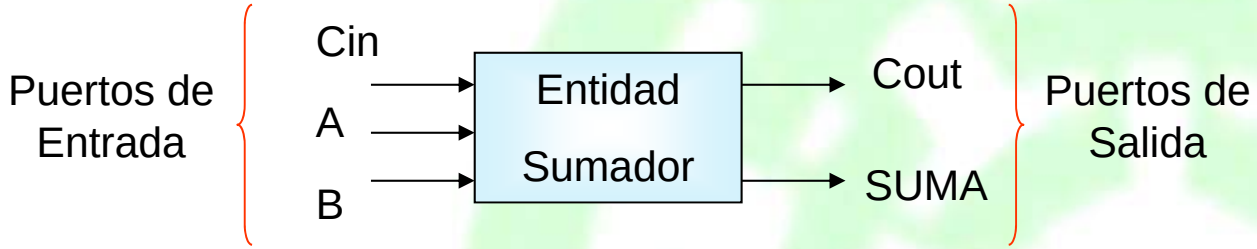
¿Qué es una Entidad?

Cinvestav

entidad (**entity)** ➔ Bloque elemental de diseño

Circuitos elementales digitales que forman de manera individual o en conjunto un sistema digital

Ejemplos: Compuertas, Flip-Flops, Sumadores/Restadores, Multiplexores, Contadores, Multiplicadores, ALUs, Neurona-Digital, etc.



Declaración de una entidad ➔ Consiste en la descripción de los puertos de entrada o salida de un circuito, el cual es identificado como una entidad (**entity)**

¡Importante!

No se describe cómo será realizado o implementado el circuito, es decir, su Arquitectura

¿Cómo se Describe a un Puerto?

Cinvestav

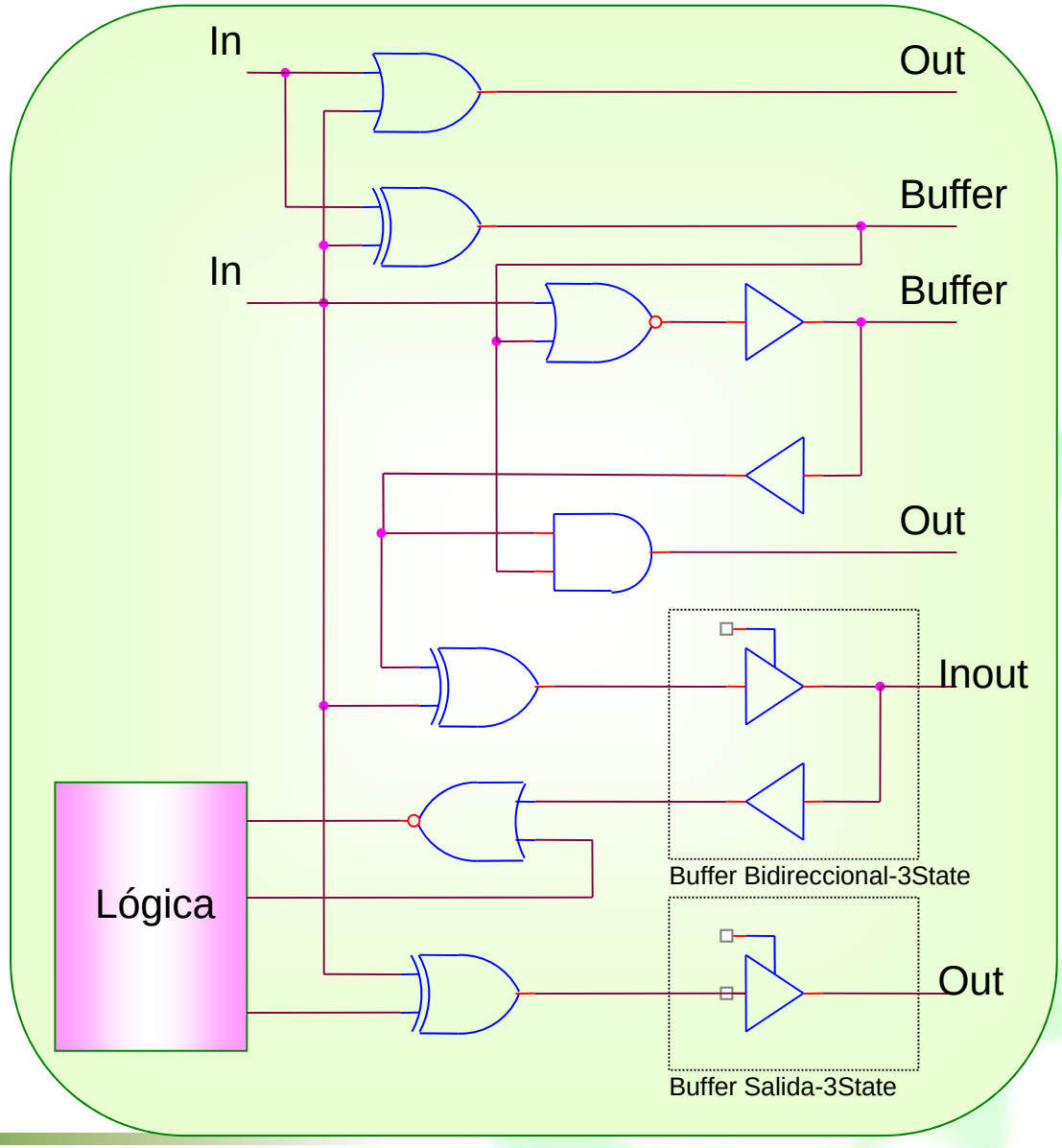
Nombre
Identificador

Modo
in = Entrada
out = Salida
inout •Puerto de Entrada (Lectura) y Salida (Escritura) •El valor leído (Entrada) es aquél que llega al puerto, y no el valor que se le asigna (Salida), en caso de existir.
buffer •Similar al Puerto de Salida (Escritura), pero además puede ser leído. •El valor leído (Entrada) es el mismo valor asignado (Salida) al puerto.

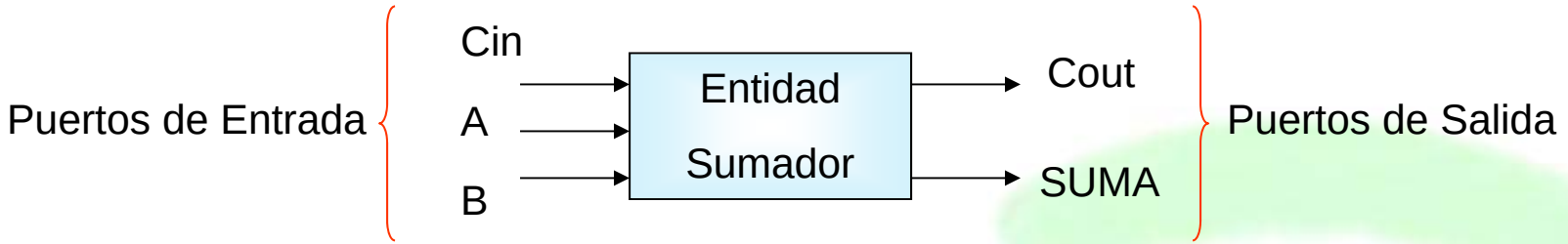
Tipo de Dato	
Conjuntos de Valores que se les ha asignado un nombre (p.ej. *bit, boolean, bit_vector*, etc), de tal forma que un objeto (p.ej. una Señal) de un determinado Tipo (p.ej. el tipo bit_vector) pueda tomar cualquier valor dentro del conjunto de valores que define al Tipo especificado.	
bit (pkg.*standard*)	Valores de '0' o '1' Lógico
boolean (pkg.*standard*)	Define valores de cierto o falso de acuerdo con una expresión
bit_vector (pkg.*standard*)	Conjunto de bits que representa a un grupo de señales de ent. o sal.
integer (pkg.*standard*)	Números enteros
std_logic (pkg.*std_logic_1164*)	Valores 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'
std_logic_vector (pkg.*std_logic_1164*)	Arreglos de std_logic
Más tipos **Se irán introduciendo conforme avance el curso**	

Paquete (pkg.) en el cual es definido el tipo.
Ver: "Uso de Bibliotecas y Paquetes"

Modos de un Puerto



Ejemplo: Sumador Completo



(entity) Inicia declaración de la entidad

(--) Indica Comentario

Identificador de la entidad

Nombres de los puertos

Línea N°.	Sumador-completo de dos datos con longitudes de 1-bit (Declaración de Entidad)
1	--Declaración de la entidad de un circuito sumador
2	**entity** sumador **is**
3	**port** (A, B, Cin: **in** bit;
4	SUMA, Cout: **out** bit);
5	**end** sumador;

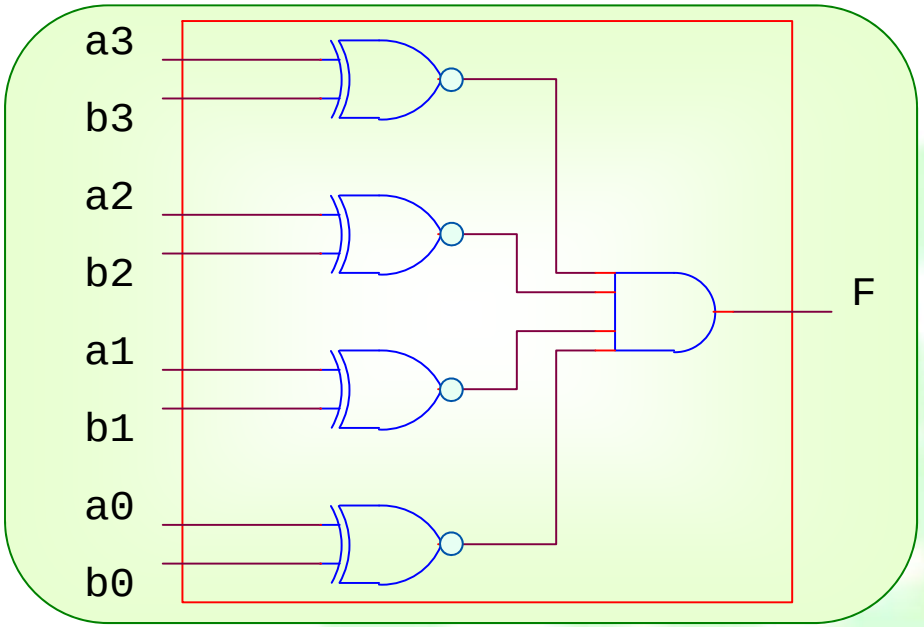
Modo de Operación

Tipo de Dato

(;) Finaliza declaración o subdeclaración

(end) Finaliza declaración de la entidad

Ejemplo: Comparador

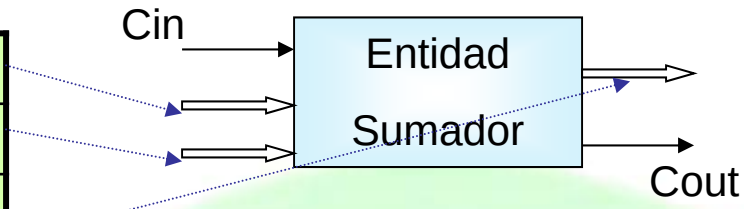


Línea No.	Comparador – Uso de dos datos con longitudes de 4-bit (Declaración de Entidad)
1	--Declaracion de la entidad
2	**entity** circuito **is**
3	**port** (a3, b3, a2, b2, a1, b1, a0, b0: **in** bit;
4	F: **out** bit);
5	**end** circuito;

Uso de Vectores

Cinvestav

vector_A	= [A3, A2, A1, A0]
vector_B	= [B3, B2, B1, B0]
vector_SUMA	= [S3, S2, S1, S0]



Declaración de Puertos Tipo-Vector

```
port (vector_A, vector_B: in bit_vector (3 downto 0);
      vector_SUMA: out bit_vector (3 downto 0));
```

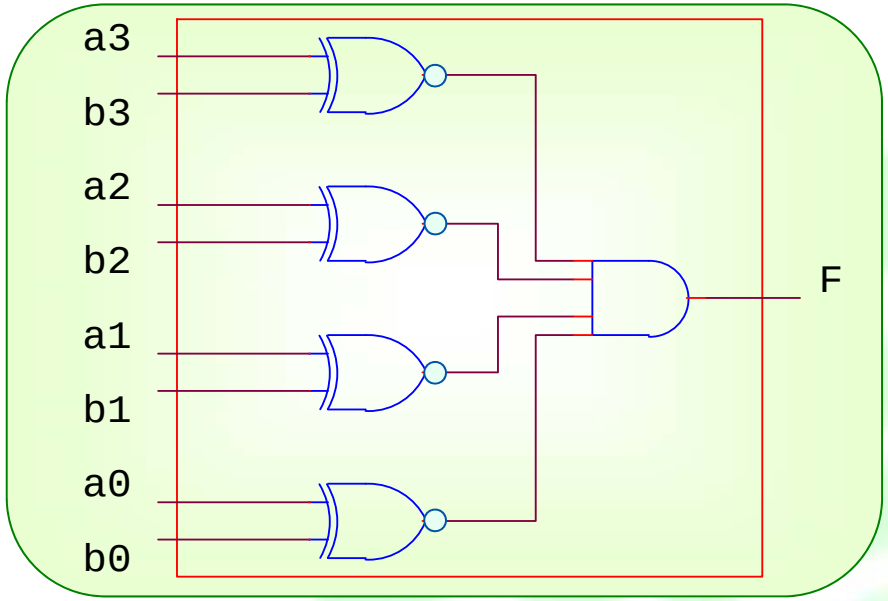
Para ordenar en forma ascendente utilizar **to** en lugar de **downto** (p.ej. 0 to 3)

Sumador-completo de dos datos con longitudes de 4-bit (Declaración de Entidad – Uso de Vectores)

```
entity sumador is
    port (A, B: in bit_vector (3 downto 0);
          Cin: in bit;
          Cout: out bit;
          SUMA: out bit_vector (3 downto 0));
end sumador;
```

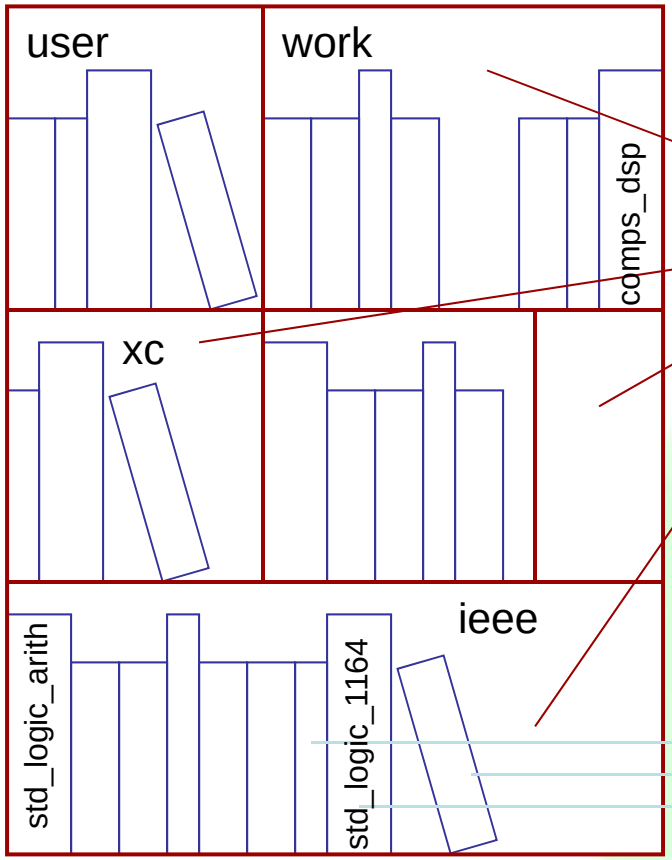
Ejemplo: Comparador (Uso de Vectores)

Cinvestav



Línea N°.	Comparador – Uso de dos datos con longitudes de 4-bit (Declaración de Entidad – Uso de Vectores)
1	--Declaracion de la entidad
2	**entity** circuito **is**
3	**port** (a, b: **in** bit_vector (3 downto 0);
4	F: **out** bit);
5	**end** circuito;

Uso de Bibliotecas y Paquetes



Biblioteca (library)

- Lugar donde se almacenan los Paquetes definidos por el fabricante de la herramienta de desarrollo o el usuario.
- Lugar donde se permite almacenar resultados de la compilación de diseños, con el fin de utilizarlos en otros.

Paquete (Package)

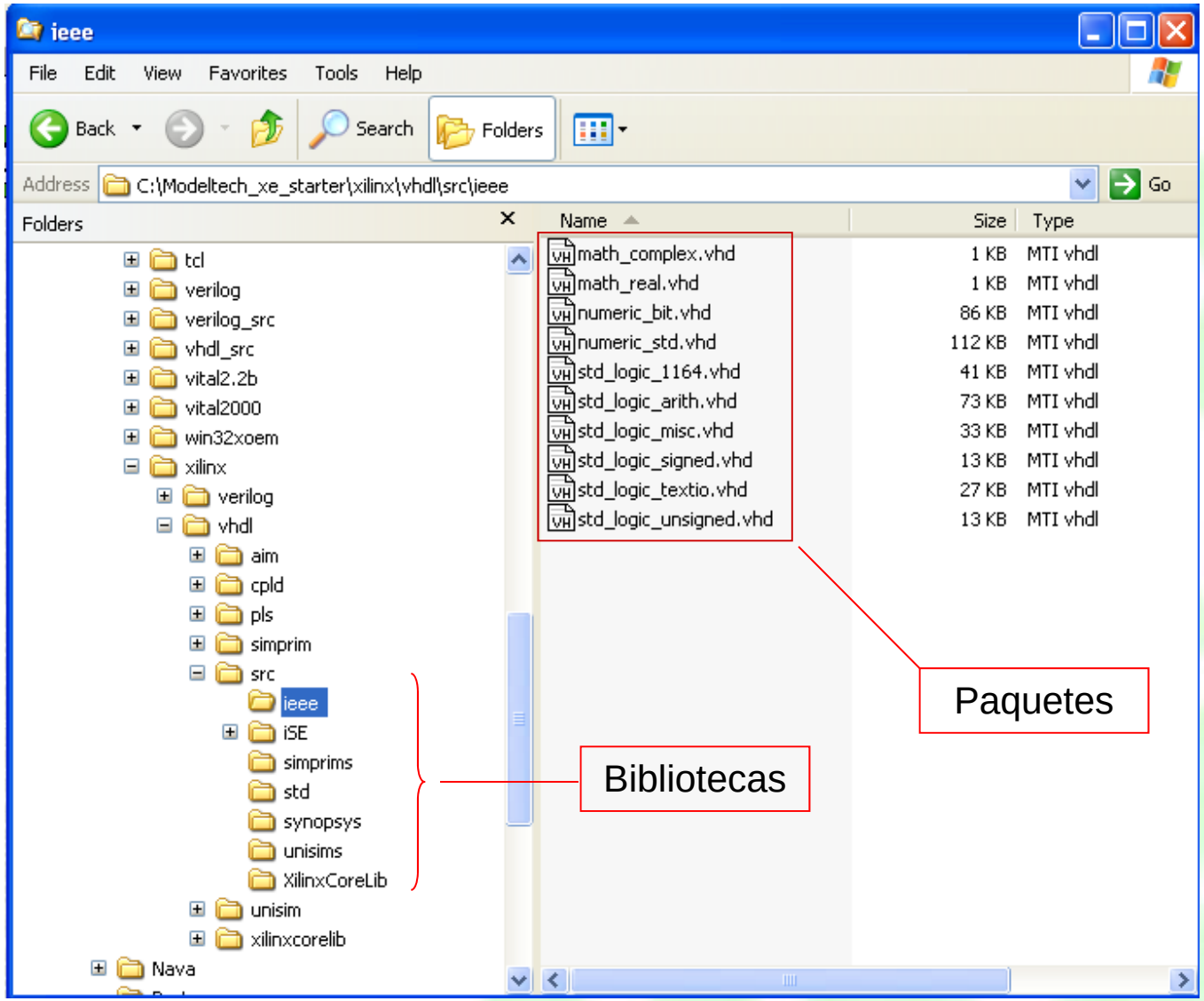
- Un paquete contiene:
  - Declaraciones de Tipos y Subtipos de Datos
  - Definiciones de Constantes
  - Definiciones de Funciones y Procedimientos
  - Declaraciones de Componentes (Sumadores, Restadores, Contadores, Multiplicadores, etc)

Un Paquete = Macro-Unidad de Diseño

Objetivo: Facilitar el diseño

Uso de Bibliotecas y Paquetes

Cinvestav



Uso de Bibliotecas y Paquetes

Cinvestav

Para llamar un paquete es necesario llamar a la biblioteca que lo contiene (donde ha sido compilado)

Sintaxis: **use** nombre_biblioteca**.**nombre_paquete**.all**;

Ejemplo: **use** ieee.std_logic_1164.**all**;

Uso del paquete **std_logic_1164** incluido en la biblioteca **ieee**

Permite el uso de todos los componentes almacenados en el paquete

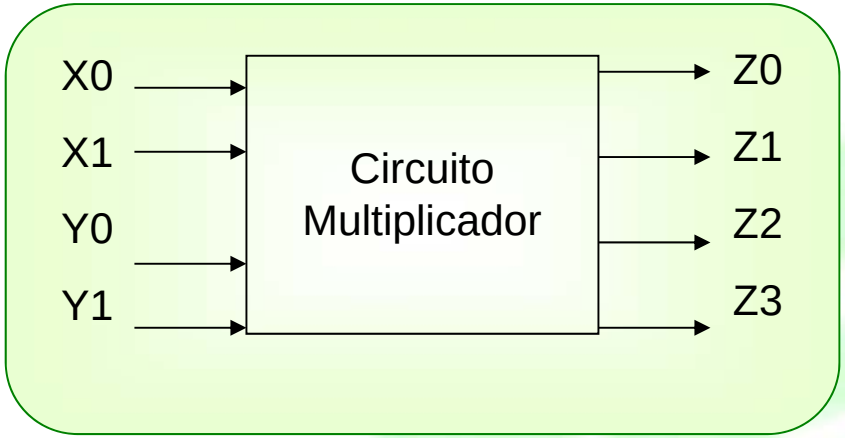
Bibliotecas y Paquetes

Cinvestav

Paquetes predefinidos comúnmente utilizados	
Standard	
standard	•Contiene tipos básicos: bit, bit_vector, integer •Paquete incluido por omisión.
IEEE	
std_logic_1164	•Define los tipos: std_logic, std_ulogic, std_logic_vector, std_ulogic_vector •Define funciones de conversión basadas sobre estos tipos.
numeric_bit	•Define tipos de vectores signados y no-signados basados en el tipo bit y todos los operadores aritméticos sobre estos tipos. •Define funciones extendidas y de conversión para dichos tipos.
numeric_std	Define tipos de vectores signados y no-signados basados en el tipo std_logic. Paquete equivalente al Paquete std_logic_arith
Synopsys	
std_logic_arith	•Define tipos de vectores signados y no-signados, y todos los operadores aritméticos sobre estos tipos. •Define funciones extendidas y de conversión para dichos tipos.
std_logic_unsigned	•Define operadores aritméticos sobre el tipo std_ulogic_vector y los considera como operadores no-signados.
std_logic_signed	•Define operadores aritméticos sobre el tipo std_logic_vector y los considera como operadores signados.
std_logic_misc	•Define tipos, subtipos, constantes y funciones complementarios para el paquete std_logic_1164.

Ejemplo: Multiplicador

Cinvestav



Multiplicador de dos datos con longitudes de 2-bit
(Declaración de Entidad – Uso de Biblioteca y Paquete)

```
library ieee;
use ieee.std_logic_1164.all;
entity multiplica is
        port (X0, X1, Y0, Y1: in std_logic;
                Z3, Z2, Z1, Z0: out std_logic);
end multiplica;
```

¿Qué es Arquitectura?

Cinvestav

arquitectura (**architecture)**

Unidad de Diseño Secundaria que describe el comportamiento interno de una entidad.

¿Cómo? - A través de la programación de varios procedimientos que permitan que la entidad (**entity**) cumpla con las condiciones de operación o comportamiento deseadas.

Niveles de ***Descripción*** utilizados

Estilo Programación o Modelización

- Nivel Algoritmo → *Funcional*
- Nivel de Transferencia entre Registros (*RTL*) → *Flujo de Datos*
- Nivel Lógico → *Flujo de Datos*
- Nivel Compuerta → *Estructural*

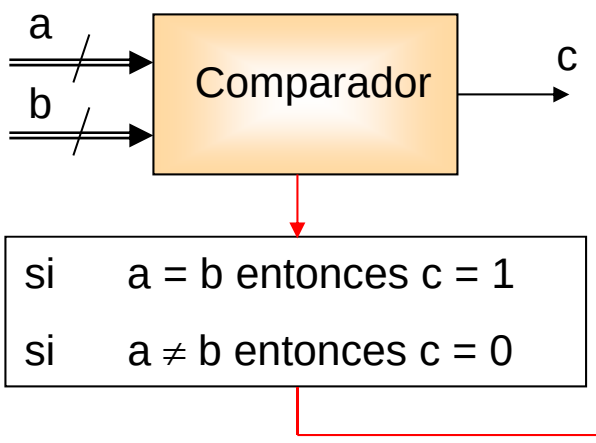
Nivel Transistor (Topología / Layout)

Estilo de Modelización - Funcional

Cinvestav

Funcional - En este caso, se describen las relaciones entre las entradas y salidas, sin importar la estructura o implementación física del sistema o circuito.

Uso de **if-then-else** (construcción secuencial)

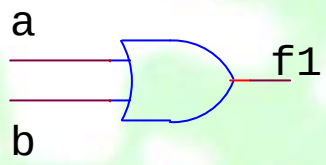


Línea Nº	Arquitectura - Comparador de Igualdad de dos Datos de Long. = 2Bits
1	--Ejemplo de una descripción abstracta (funcional)
2	**library** ieee;
3	**use** ieee.std_logic_1164.**all**;
4	**entity** comp **is**
5	**port** (a,b: **in** bit_vector (1 **downto** 0);
6	c: **out** bit);
7	**end** comp;
8	**architecture** funcional **of** comp **is**
9	**begin**
10	compara: **process** (a,b)
11	**begin**
12	**if** a = b **then**
13	c <= '1';
14	**else**
15	c <= '0';
16	**end if**;
17	**end process** compara;
18	**end** funcional;

Estilo de Modelización - Funcional

Cinvestav

Línea Nº	Arquitectura - Compuerta OR de dos entradas
1	--Ejemplo de una descripción abstracta (funcional)
2	**library** ieee;
3	**use** ieee.std_logic_1164.**all**;
4	**entity** com_or **is**
5	**port** (a,b: **in** std_logic;
6	f1: **out** std_logic);
7	**end** com_or;
8	**architecture** funcional **of** com_or **is**
9	**begin**
10	**process** (a,b) **begin**
11	**if** (a = '0' **and** b='0') **then**
12	f1 <= '0';
13	**else**
14	f1 <= '1';
15	**end if**;
16	**end process**;
17	**end** funcional;



a	b	f1
0	0	0
0	1	1
1	0	1
1	1	1

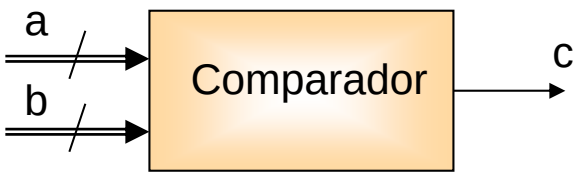
Estilo de Modelización – Flujo de Datos

Cinvestav

Flujo de Datos - En este caso, se describe la forma en la que los datos se pueden transferir entre los diferentes módulos operativos que constituyen la entidad (sistema o circuito)

La construcción

when-else

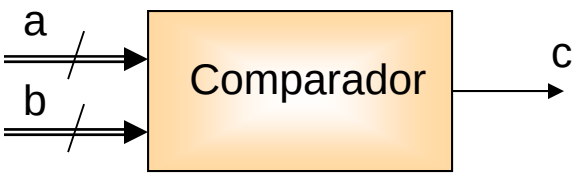


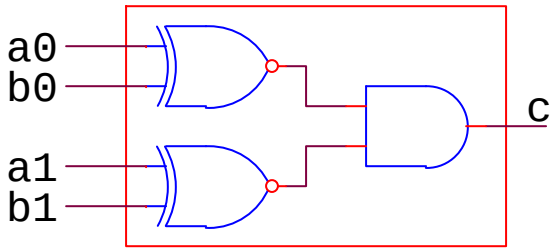
Línea Nº	Arquitectura - Comparador de Igualdad de dos Datos de Long. = 2Bits
1	--Ejemplo de una arquitectura usando when-else
2	**library** ieee;
3	**use** ieee.std_logic_1164.**all**;
4	**entity** comp **is**
5	**port** (a,b: **in** bit_vector (1 **downto** 0);
6	c: **out** bit);
7	**end** comp;
8	**architecture** fun_datos **of** comp **is**
9	**begin**
10	c <= ‘1’ **when** (a = b) **else** ‘0’;
11	**end** fun_datos;
12	

Estilo de Modelización – Flujo de Datos

Cinvestav

Uso de **ecuaciones booleanas**



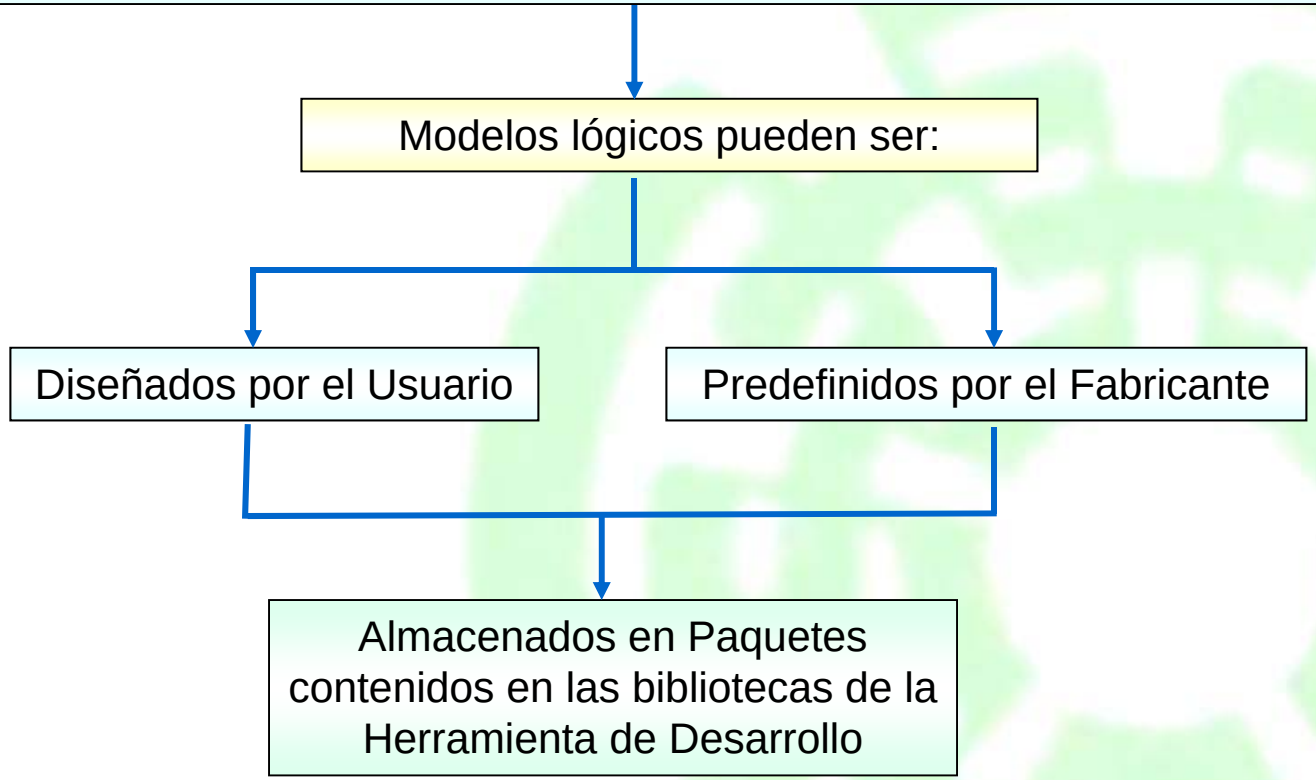


Línea Nº	Arquitectura - Comparador de Igualdad de dos Datos de Long. = 2Bits
1	--Ejemplo de una arquitectura usando ecs. booleanas
2	**library** ieee;
3	**use** ieee.std_logic_1164.**all**;
4	**entity** comp **is**
5	**port** (a,b: **in** bit_vector (1 **downto** 0);
6	c: **out** bit);
7	**end** comp;
8	**architecture** booleana **of** comp **is**
9	**begin**
10	c <= (a(1) **xnor** b(1)) **and** (a(0) **xnor** b(0));
11	**end** booleana;
12	

Estilo de Modelización – Estructural

Cinvestav

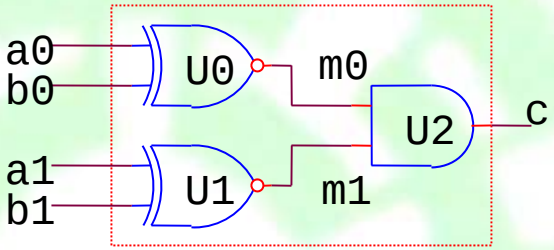
Estructural - En este caso, el comportamiento de un sistema o circuito es descrito mediante modelos lógicos establecidos de los componentes que conforman al sistema o circuito, como son: Compuertas, Sumadores, Contadores, etc.



Estilo de Modelización – Estructural

Cinvestav

Línea Nº	Arquitectura - Comparador de Igualdad de dos Datos de Long. = 2Bits
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**use** work.compuertas.**all**;
4	**entity** comp **is**
5	**port** (a,b: **in** bit_vector (0 **to** 1);
6	c: **out** bit);
7	**end** comp;
8	**architecture** estructural **of** comp **is**
9	**signal** m: bit_vector (0 **to** 1);
10	**begin**
11	U0: xnor2 **port map** (a(0), b(0), m(0));
12	U1: xnor2 **port map** (a(1), b(1), m(1));
13	U2: and2 **port map** (m(0), m(1), c);
14	**end** estructural;



Resumen

Cinvestav

En resumen, se puede decir que:

La descripción **funcional** se basa principalmente en el uso de procesos y de declaraciones *secuenciales*. Esta descripción es similar a la hecha en un lenguaje de programación de alto nivel, por su alto nivel de abstracción.

Mas que especificar la estructura o la forma en que se deben conectar los componentes de un diseño, nos limitamos a describir su comportamiento.

Una descripción **funcional** consiste de una serie de instrucciones, que ejecutadas *secuencialmente*, modelan el comportamiento del circuito.

La ventaja de este tipo de descripción, es que no se requiere enfocar a un nivel de compuerta para implementar un diseño.

En **VHDL** una descripción **funcional** necesariamente implica el uso de por lo menos un bloque **PROCESS** (**if-then-else**)

Resumen

Cinvestav

Definición de instrucción secuencial:

Las instrucciones secuenciales son aquellas que son ejecutadas serialmente, una después de otra. La mayoría de los lenguajes de programación, como C o Pascal, utilizan este tipo de instrucciones.

En **VHDL** las instrucciones *secuenciales* son implementadas únicamente dentro del bloque **PROCESS**.

***COMENTARIO**:*

Dentro de una arquitectura en **VHDL**, no existe un orden específico de ejecución de las asignaciones. El orden en el que las instrucciones son ejecutadas depende de los eventos ocurridos en las señales, similar al funcionamiento del circuito.

Resumen

Cinvestav

Descripción por flujo de datos:

La descripción por flujo de datos indica la forma en que los datos se pueden transferir de una señal a otra sin necesidad de declaraciones *secuenciales*.

Este tipo de descripciones permite definir el flujo que tomarán los datos entre módulos encargados de realizar operaciones: **when-else**.

Esta forma de descripción, puede realizarse también mediante ecuaciones booleanas, en donde se emplean los operadores correspondientes: **or**, **and**, **nand**, **nor**, **xor**, **xnor**.

Descripción estructural:

Este tipo de descripción basa su comportamiento en modelos lógicos establecidos (*compuertas*, *sumadores*, *contadores*, *etc*.).

El usuario puede diseñar estas estructuras y guardarlas para su uso posterior o tomarlas de los *paquetes* contenidos en las *librerías* de diseño del software que se esté utilizando.

Resumen

Cinvestav

Comparación entre los estilos de diseño.

El estilo de diseño utilizado en la programación del circuito depende del diseñador y de la complejidad del proyecto. Por ejemplo, un diseño puede describirse por medio de **ecuaciones booleanas**, pero si es muy extenso quizá sea más apropiado emplear **estructuras jerárquicas** para dividirlo.

Ahora bien, si se requiere diseñar un sistema cuyo funcionamiento dependa sólo de sus entradas y salidas, es conveniente utilizar la descripción **funcional**, la cual presenta la ventaja de requerir menos instrucciones y el diseñador no necesita un conocimiento previo de cada componente del circuito.

Resumen

Las instrucciones *concurrentes* (*flujo de datos y estructural*) se utilizan fuera de un bloque **PROCESS**, a diferencia de las instrucciones *secuenciales*, que únicamente se utilizan dentro del bloque **PROCESS**.

Entonces, las descripciones se pueden distinguir entre **secuenciales** (*funcional*) y **concurrentes** (*flujo de datos y estructural*).

Ejercicios

Cinvestav

Para la siguiente declaración:

```
library ieee;
use ieee.std_logic_1164.all;
entity seleccion is port (
m: in std_logic_vector(0 to 3);
f: out std_logic);
end seleccion;
```

Indicar:

nombre de la entidad:_____

los puertos de entrada:_____

los puertos de salida:_____

el tipo de dato:_____

Ejercicios

Cinvestav

Señale cuál de los siguientes identificadores son correctos o incorrectos:

1logico _____

con_trol _____

Pagina _____

registro _____

2Suma _____

Desp_laza _____

N_ivel _____

architecture _____

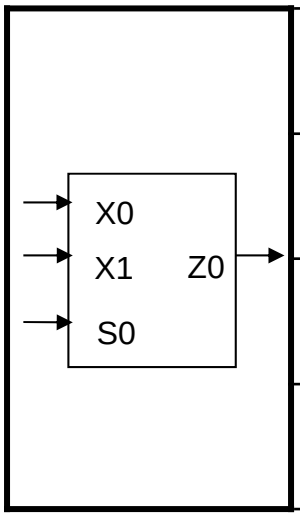
S_uma# _____

Res__ta _____

Ejercicios

Cinvestav

Describir la siguiente función por VHDL.

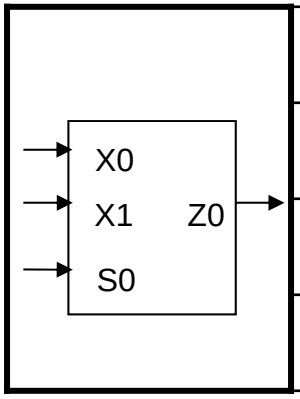


ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

MULTIPLEXOR

Ejercicios



ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

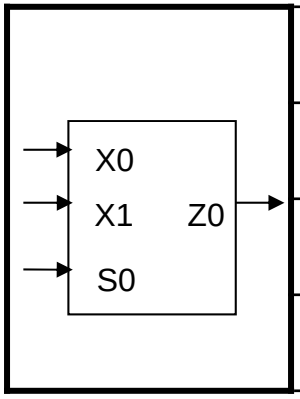
```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
   SIGNAL temp: bit_vector (2 downto 0);
BEGIN
   z0 <=   '0' WHEN temp =  "000" ELSE
           '0'  WHEN temp =  "001" ELSE
           '1' WHEN temp =  "010" ELSE
           '1' WHEN temp =  "011" ELSE
           '0' WHEN temp =  "100" ELSE
           '1' WHEN temp =  "101" ELSE
           '0' WHEN temp =  "110" ELSE
           '1' ;
temp <= s0 & x0 & x1; -- concatenación
                      -- de las entradas en un
                      -- solo bus
END data_flow;
```

Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Para la primera descripción que se mostrará, se empleará el código de la tabla extendida, resultando en el siguiente listado VHDL.

Ejercicios

Cinvestav



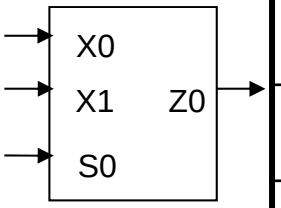
ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
   SIGNAL temp: bit_vector (2 downto 0);
BEGIN
   z0 <=   '0' WHEN temp =  "000" ELSE
           '0'  WHEN temp =  "001" ELSE
           '1' WHEN temp =  "010" ELSE
           '1' WHEN temp =  "011" ELSE
           '0' WHEN temp =  "100" ELSE
           '1' WHEN temp =  "101" ELSE
           '0' WHEN temp =  "110" ELSE
           '1' ;
temp <= s0 & x0 & x1; -- concatenación
                      -- de las entradas en un
                      -- solo bus
END data_flow;
```

Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Los valores asignados al tipo bit_vector deber ser especificados con comillas dobles (“_”) y los valores asignados al tipo bit simple, son asignados con comillas simples (‘_’).

Ejercicios



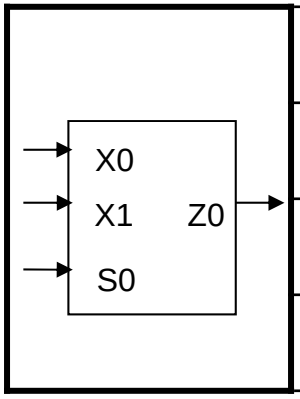
ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
   SIGNAL temp: bit_vector (2 downto 0);
BEGIN
   z0 <=   '0' WHEN temp =  "000" ELSE
           '0'  WHEN temp =  "001" ELSE
           '1' WHEN temp =  "010" ELSE
           '1' WHEN temp =  "011" ELSE
           '0' WHEN temp =  "100" ELSE
           '1' WHEN temp =  "101" ELSE
           '0' WHEN temp =  "110" ELSE
           '1' ;
temp <= s0 & x0 & x1; -- concatenación
                      -- de las entradas en un
                      -- solo bus
END data_flow;
```

Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Se empleó el objeto de datos SIGNAL para crear el bus "temp" y concatenar "s0", "x0" y "x1" en un solo objeto de datos y así facilitar la descripción.

Ejercicios



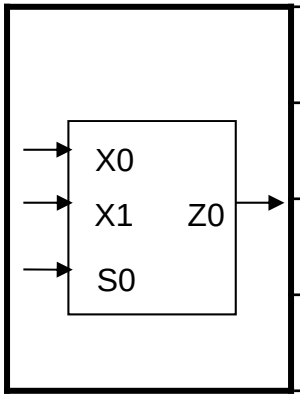
ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
   SIGNAL temp: bit_vector (2 downto 0);
BEGIN
   z0 <=   '0' WHEN temp =  "000" ELSE
           '0'  WHEN temp =  "001" ELSE
           '1' WHEN temp =  "010" ELSE
           '1' WHEN temp =  "011" ELSE
           '0' WHEN temp =  "100" ELSE
           '1' WHEN temp =  "101" ELSE
           '0' WHEN temp =  "110" ELSE
           '1' ;
temp <= s0 & x0 & x1; -- concatenación
                      -- de las entradas en un
                      -- solo bus
END data_flow;
```

Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

¿Qué tipo de descripción se realizó en este multiplexor?

Ejercicios



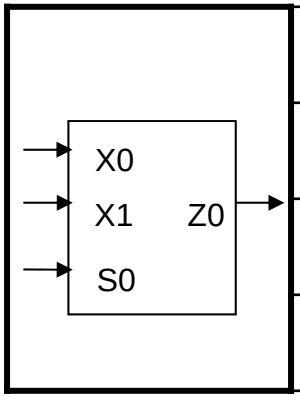
ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
   SIGNAL temp: bit_vector (2 downto 0);
BEGIN
   z0 <=   '0' WHEN temp =  "000" ELSE
           '0'  WHEN temp =  "001" ELSE
           '1' WHEN temp =  "010" ELSE
           '1' WHEN temp =  "011" ELSE
           '0' WHEN temp =  "100" ELSE
           '1' WHEN temp =  "101" ELSE
           '0' WHEN temp =  "110" ELSE
           '1' ;
temp <= s0 & x0 & x1; -- concatenación
                      -- de las entradas en un
                      -- solo bus
END data_flow;
```

Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

¿Qué tipo de descripción se realizó en este multiplexor?

Ejercicios



ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

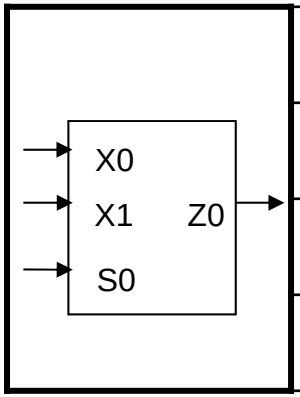
```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
BEGIN
   z0 <= x0   WHEN s0 = '0' ELSE  x1;
END data_flow;
```

Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

La siguiente descripción se deriva de la tabla simplificada, ya que se ve que Z0 depende solamente del estado de S0.

Por lo tanto, la descripción resulta más sencilla.

Ejercicios



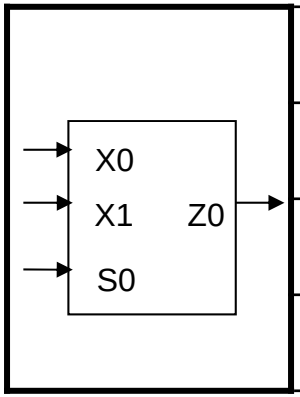
ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
BEGIN
   z0 <= x0   WHEN s0 = '0' ELSE  x1;
END data_flow;
```

Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

¿Qué tipo de descripción se realizó en este multiplexor?

Ejercicios



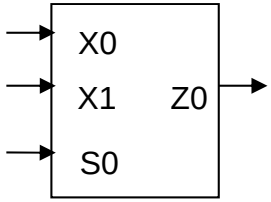
ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
BEGIN
   z0 <= x0   WHEN s0 = '0' ELSE  x1;
END data_flow;
```

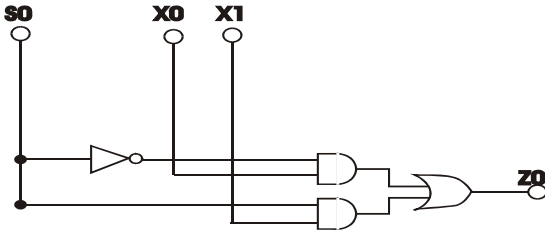
Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

¿Qué tipo de descripción se realizó en este multiplexor?

Ejercicios



ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

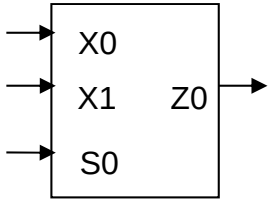



```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
   SIGNAL not_s0, and1, and2: bit;
BEGIN
   z0 <= and1 OR and2;
  and1 <= not_s0 AND x0;
  not_s0 <= NOT s0;
  and2 <= s0 AND x1;
END data_flow;
```

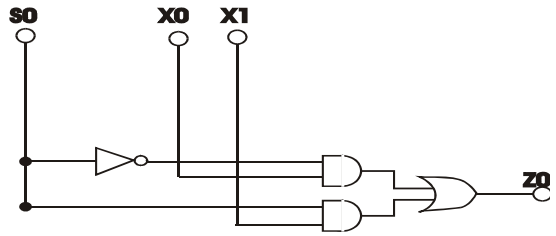
Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

La descripción mostrada a continuación, hace uso del siguiente diagrama explícito del multiplexor.

Ejercicios



ENTRADA	SALIDA
S0	Z0
0	X0
1	X1

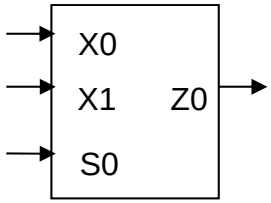



```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
   SIGNAL not_s0, and1, and2: bit;
BEGIN
   z0 <= and1 OR and2;
  and1 <= not_s0 AND x0;
  not_s0 <= NOT s0;
  and2 <= s0 AND x1;
END data_flow;
```

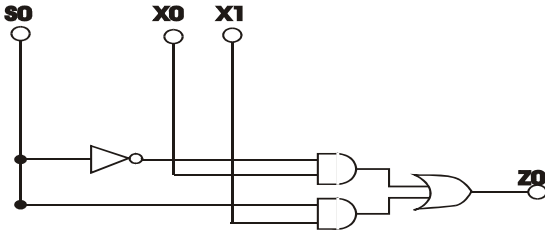
Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

¿Qué tipo de descripción se realizó en este multiplexor?

Ejercicios



ENTRADA	SALIDA
S0	Z0
0	X0
1	X1




```
ENTITY multiplexor IS
   PORT (s0, x0, x1: IN bit;
         z0: OUT bit);
END multiplexor;

ARCHITECTURE data_flow OF multiplexor IS
   SIGNAL not_s0, and1, and2: bit;
BEGIN
   z0 <= and1 OR and2;
  and1 <= not_s0 AND x0;
  not_s0 <= NOT s0;
  and2 <= s0 AND x1;
END data_flow;
```

Entradas			Salidas
S0	X0	X1	Z0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

¿Qué tipo de descripción se realizó en este multiplexor?

Cinvestav

Capítulo 3

Circuitos Lógicos Combinatorios

Programación de Estructuras Básicas

Cinvestav

Diseño (Programación) de una Estructura Básica Combinatoria

Declaración *Entidad*

Declaración *Arquitectura*

Sintaxis:

architecture *nombre_arquitectura* **of** *nombre_entidad* **is**

{*Declarativas de Bloque*} –Se analizarán posteriormente

begin

{***Enunciados Concurrentes***}

end [*nombre_arquitectura*]

Enunciado Concurrente.

Unidad de Cómputo/Cálculo que realiza lo siguiente:

- Lectura de Señales.
- Realiza cálculos basados en los valores de las Señales.
- Asigna los valores calculados a Señales específicas.

Enunciados Concurrentes

Cinvestav

Tipos de Enunciados Concurrentes	
Asignación de Señal	Permite asignar un valor calculado a una señal o puerto.
Proceso (**process**)	Permite definir un algoritmo secuencial que lee valores de Señales y calcula nuevos valores que son asignados a otras Señales.
Bloque (**block**)	Grupo de enunciados concurrentes.
Llamada a un Componente predefinido	
Llamada a un Subprograma (**procedure o function**)	Llama a un algoritmo que calcula y asigna valores a Señales

Asignación de Señales

Tipos:

- Asignaciones de Señales mediante **Ecuaciones Booleanas**
- Asignaciones Condicionales de Señales – La construcción **when-else**
- Asignaciones de Señales por Selección – La construcción **with-select-when**

Nota: Se puede utilizar el término Estructura de Control, en lugar del término Construcción

Operadores Lógicos

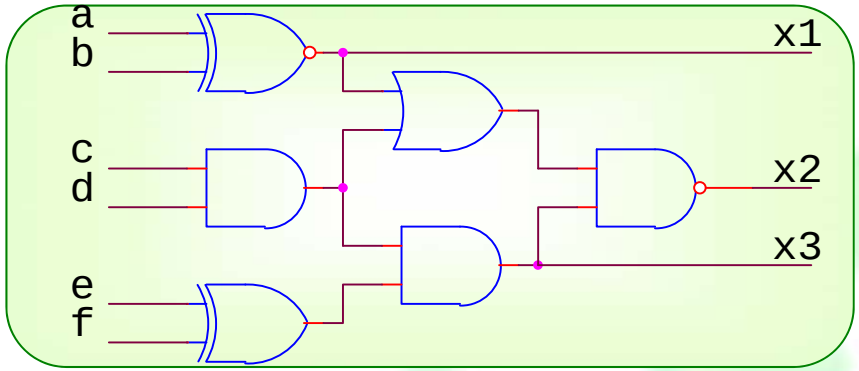
Cinvestav

Operadores Lógicos
and, or, xor, nand, nor, xnor, not
Tipos de Operandos permisibles: *bit, boolean, std_logic, también arreglos unidimensionales* (del tipo bit, boolean y std_logic)
Operandos deben tener la misma longitud, excepto para el operador **not**, el cual se aplica por lo general a un solo operando.
Si una expresión incluye varios de estos operadores (p.ej. AND, OR, XNOR) es necesario utilizar paréntesis para evaluarla correctamente.

Ecuación Booleana	Expresión VHDL
$q = a + (b \cdot c)$	q = a or (b and c)
$y = a + (\overline{b} \cdot \overline{c}) + d$	y = a or (not b and not c) or d

Asignación de Señales con Ecuaciones Booleanas

Cinvestav



L	Ejemplo Nº 1 – Asignaciones de Señales – Uso de Ecs. Booleanas
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** logica **is**
4	**port** (a,b,c, d, e, f: **in** std_logic;
5	x1, x2, x3: **out** std_logic);
6	**end** logica;
7	**architecture** booleana **of** logica **is**
8	**begin**
9	x1 <= a **xnor** b;
10	x2 <= ((c **and** d) **or** (a **xnor** b)) **nand** ((e **xor** f) **and** (c **and** d));
11	x3 <= (e **xor** f) **and** (c **and** d);
12	**end** booleana;

En este tipo de asignaciones, cada función de salida es descrita mediante su ecuación booleana correspondiente, lo cual implica el uso de operadores lógicos.

Asignación de Señales con Ecuaciones Booleanas

Cinvestav

A	B	C	P	Q	R
0	0	0	1	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	0	0

L	Ejemplo Nº 2 – Asignaciones de Señales – Uso de Ecs. Booleanas
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** logica **is**
4	**port** (A,B,C: **in** std_logic;
5	P,Q,R: **out** std_logic);
6	**end** logica;
7	**architecture** arq_log **of** logica **is**
8	**begin**
9	P <= (not A and not B and not C) or (not A and not B and C)
10	or (not A and B and C) or (A and B and C);
11	Q <= (not A and not B and C) or (A and not B and C)
12	or (A and B and not C);
13	R<= (not A and not B and not C) or (not A and B and not C)
14	or (not A and B and C);
15	**end** arq_log;

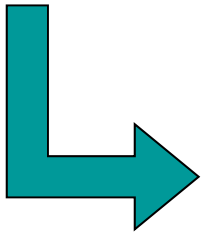
$$P = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + ABC$$
$$Q = \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C}$$
$$R = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC$$

Formato del enunciado WHEN-ELSE

Cinvestav

Enunciados WHEN-ELSE:

La construcción **when-else** es una asignación condicional, que debe incluir todas las opciones posibles de variación de una señal.



SINTAXIS:

[*etiqueta:* **]**

señal **<= [** *opción* **] [** *valor* **] when** *condición 1* **else**
[*valor* **] when** *condición 2* **else**
unaffected;

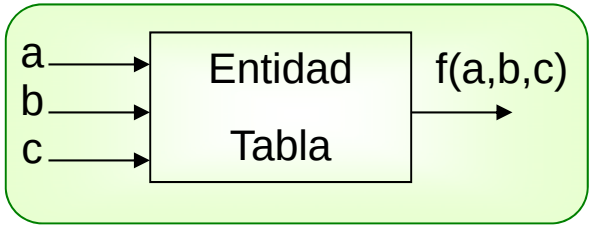
unaffected : Permite que no se realice ninguna acción.

Se pueden anidar varias condiciones en una misma asignación

Ejemplo:

```
s <= "11" when a = b else
     "10" when a > b else
     "01";
```

Asignación Condicional de Señales



a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

L	Ejemplo Nº 3 - Uso de la construcción **when-else**
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** tabla **is**
4	**port** (a,b,c: **in** std_logic;
5	f: **out** std_logic);
6	**end** tabla;
7	**architecture** arq_tabla **of** tabla **is**
8	**begin**
9	f <= ‘1’ **when** (a = ‘0’ **and** b=‘0’ **and** c=‘0’) **else**
10	‘1’ **when** (a = ‘0’ **and** b=‘1’ **and** c=‘1’) **else**
11	‘1’ **when** (a = ‘1’ **and** b=‘1’ **and** c=‘0’) **else**
12	‘1’ **when** (a = ‘1’ **and** b=‘1’ **and** c=‘1’) **else**
13	‘0’;
14	**end** arq_tabla;

La construcción **when-else** permite definir paso a paso el comportamiento de un sistema. Para esto, se declaran los valores que se deben asignar a una señal (o grupo) en función de las diferentes condiciones de entrada posibles. El orden en el que se declaren las condiciones de entrada, no es importante.

Asignación Condicional de Señales

Cinvestav

L	Ejemplo Nº 4 - Uso de la construcción **when-else**
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** funcion **is**
4	**port** (A,B,C,D: **in** std_logic;
5	F: **out** std_logic);
6	**end** funcion;
7	**architecture** arq_func **of** funcion **is**
8	**begin**
9	F <= ‘1’ **when** (A = ‘0’ **and** B=‘0’ **and** C=‘0’ **and** D=‘0’) **else**
10	‘1’ **when** (A = ‘0’ **and** B=‘1’ **and** C=‘0’ **and** D=‘1’) **else**
11	‘1’ **when** (A = ‘0’ **and** B=‘1’ **and** C=‘1’ **and** D=‘0’) **else**
12	‘1’ **when** (A = ‘1’ **and** B=‘1’ **and** C=‘1’ **and** D=‘1’) **else**
13	‘0’;
14	**end** arq_func;

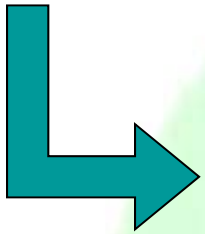
A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Formato del enunciado WITH-SELECT-WHEN

Cinvestav

Enunciados WITH-SELECT-WHEN:

La construcción **with-select-when** es una asignación por selección, la asignación se hace según el resultado de la expresión.



SINTAXIS

with *expresión* **select**
señal **<= [** *opciones* **] [** *valor* **] when** *caso 1*,
[*valor* **] when** *caso 2*;

caso :
•Valor que toma la expresión
•Intervalo de valores con **to** o **downto**
•Lista de valores separados por **|** [Alt-124]
•La palabra reservada **others**

EJEMPLO:

```
with estado select
semaforo <= "rojo"          when "01",
            "verde"         when "10",
            "amarillo"      when "11",
            "no funciona" when others;
```

Asignación de Señales por Selección

Cinvestav

a(1)	a(0)	C
0	0	1
0	1	0
1	0	1
1	1	1

L	Ejemplo Nº 5 – Uso de la construcción **with-select-when**
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** circuito **is**
4	**port** (a: **in** std_logic_vector (1 **downto** 0);
5	C: **out** std_logic);
6	**end** circuito;
7	**architecture** arq_cir **of** circuito **is**
8	**begin**
9	**with** a **select**
10	C <= '1' **when** "00",
11	'0' **when** "01",
12	'1' **when** "10",
13	'1' **when others**;
14	**end** arq_cir;

Únicamente, se utiliza la *coma* (**,**), el *punto y coma* (**;**) se utiliza cuando se finaliza la construcción ***with-select***

- La estructura **with-select-when** se utiliza para asignar un valor (de varios posibles) a una señal o grupo de señales con base a los diferentes valores de otra señal o grupo de señales previamente seleccionada(o).
- Por lo general, un grupo de señales forman un vector, como en el ejemplo descrito *a(1)* y *a(0)* forman el vector *a*.

Asignación de Señales por Selección

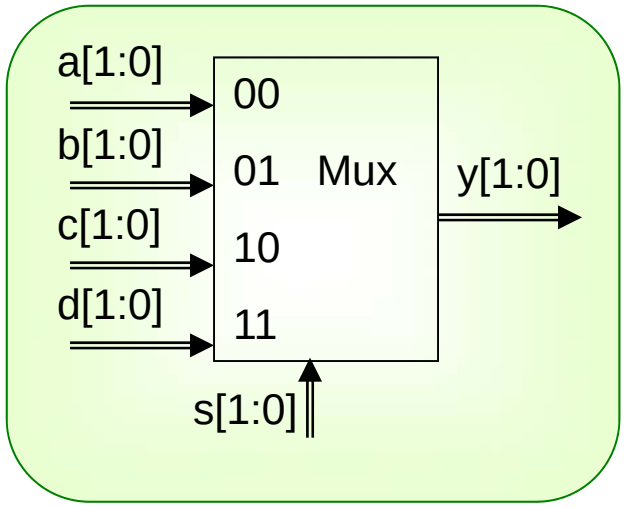
Cinvestav

L	Ejemplo Nº 6 – Uso de la construcción **with-select-when** *Circuito Combinatorio que detecte Números Primos de 4-Bits*
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** seleccion **is**
4	**port** (M: **in** std_logic_vector (3 downto 0);
5	F: **out** std_logic);
6	**end** seleccion;
7	**architecture** arq_selec **of** seleccion **is**
8	**begin**
9	**with** M **select**
10	F <= ‘1’ **when** “0001”,
11	‘1’ **when** “0010”,
12	‘1’ **when** “0011”,
13	‘1’ **when** “0101”,
14	‘1’ **when** “0111”,
15	‘1’ **when** “1011”,
16	‘1’ **when** “1101”,
17	‘0’ **when others**;
18	**end** arq_selec;

M3	M2	M1	M0	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Ejemplo: Multiplexor

Mux 4 a 1 (2-Bits)

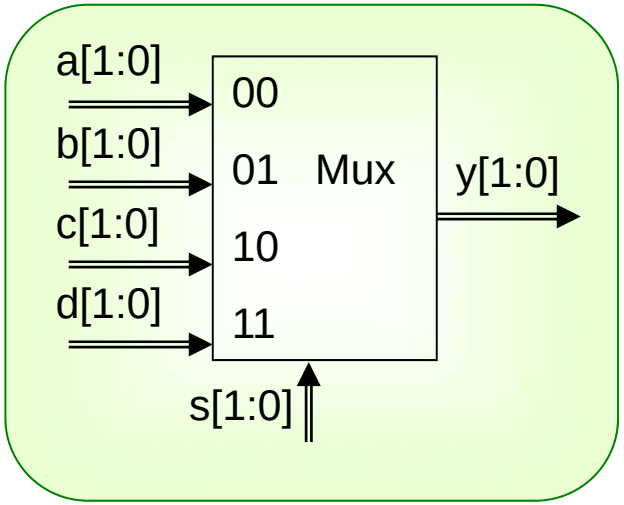


Ejemplo Nº 7 – Multiplexor 4 a 1 / Uso de Ecs. Booleanas

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
        port (a, b, c, d: in std_logic_vector (1 downto 0);
                s: in std_logic_vector (1 downto 0);
                y: out std_logic_vector (1 downto 0));
end mux;
architecture arqmux of mux is
begin
        y(1) <= (a(1) and not s(1) and not s(0)) or
                (b(1) and not s(1) and s(0)) or
                (c(1) and s(1) and not s(0)) or
                (d(1) and s(1) and s(0));
        y(0) <= (a(0) and not s(1) and not s(0)) or
                (b(0) and not s(1) and s(0)) or
                (c(0) and s(1) and not s(0)) or
                (d(0) and s(1) and s(0));
end arqmux;
```

Ejemplo: Multiplexor

Mux 4 a 1 (2-Bits)

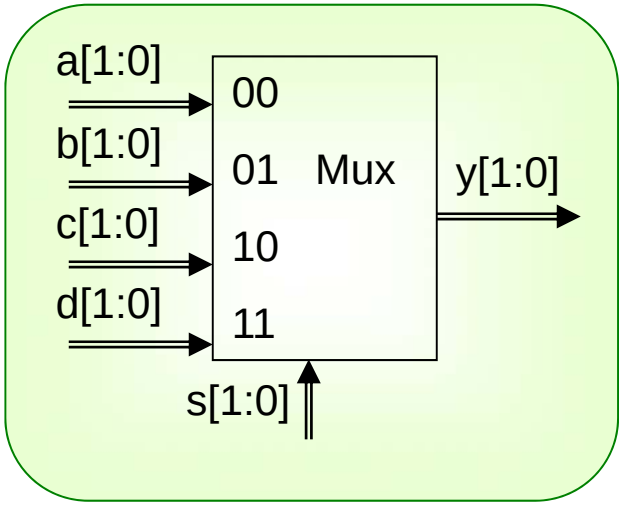


Ejemplo Nº 8 – Multiplexor 4 a 1 / Uso de **when-else**

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
        port (a, b, c, d: in std_logic_vector (1 downto 0);
                s: in std_logic_vector (1 downto 0);
                y: out std_logic_vector (1 downto 0));
end mux;
architecture arqmux of mux is
begin
        y <=  a when  s = “00” else
                b when  s = “01” else
                c when  s = “10” else
                d ;
end arqmux;
```

Ejemplo: Multiplexor

Mux 4 a 1 (2-Bits)



Ejemplo Nº 9 – Multiplexor 4 a 1 / Uso de **with-select-when**

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
        port (a, b, c, d: in std_logic_vector (1 downto 0);
                s: in std_logic_vector (1 downto 0);
                y: out std_logic_vector (1 downto 0));
end mux;
architecture arqmux of mux is
begin
with s select
        y <=  a when “00”,
                b when “01”,
                c when “10”,
                d when others;
end arqmux;
```

Procesos (process)

Cinvestav

Tipos de Enunciados Concurrentes.	
Asignación de Señal	Permite asignar un valor calculado a una señal o puerto.
Proceso (**process**)	Permite definir un algoritmo secuencial que lee valores de Señales y calcula nuevos valores que son asignados a otras Señales.
Bloque (**block**)	Grupo de enunciados concurrentes.
Llamada a un Componente predefinido	
Llamada a un Subprograma	Llama a un algoritmo que calcula y asigna valores a Señales

Proceso (process)

- Cada proceso es conformado por un conjunto de enunciados secuenciales.
- Enunciados Secuenciales ➔ Son interpretados por la herramienta de síntesis en forma secuencial, es decir, uno por uno, por lo que el orden en el cual son declarados tiene un efecto significativo en la lógica que se intenta describir o sintetizar.

Procesos (process)

Proceso (process)

Enunciados Secuenciales

- Enunciados de Asignación de Variables
- Enunciados de Asignación de Señales
- Enunciados ***if***
- Enunciados ***case***
- Enunciados ***loop***
- Enunciados ***next***
- Enunciados ***exit***
- Enunciados de Subprogramas
- Enunciados ***return***
- Enunciados ***wait***
- Enunciados ***null***

Nota importante:

Una ***señal*** que se vea involucrada dentro de un proceso no recibe inmediatamente el valor asignado, sólo hasta el final del mismo. Una ***variable*** que sea utilizada dentro de un proceso sí recibe el valor de forma inmediata.

Formato del enunciado IF-THEN-ELSE

Cinvestav

Enunciados if:

- La construcción **if-then-else**

```
if la_condición_es_cierta then
        {ejecuta grupo-1 de enunciados secuenciales};
else
        {ejecuta grupo-2 de enunciados secuenciales};
end if;
```

Enunciados if:

- La construcción **if-then-elsif-then-else**

```
if la_condición-1_se_cumple then
        {ejecuta grupo-1 de enunciados secuenciales};
elsif la_condición-2_se_cumple then
        {ejecuta grupo-2 de enunciados secuenciales};
else
        {ejecuta grupo-3 de enunciados secuenciales};
end if;
```

Operadores Relacionales

Cinvestav

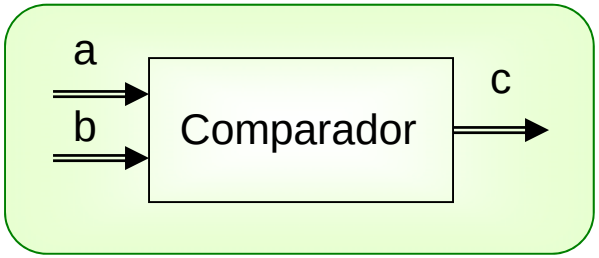
Operadores Relacionales	
Características. ➡ Uso: Para fines de comparación de datos. ➡ Operadores incluidos en los paquetes: std_numeric y std_logic_arith ➡ Los operadores de Igualdad y Desigualdad (= , /=) utilizan todos los tipos de datos. ➡ Los operadores (<, <=, >, >=) son definidos para los tipos escalar y arreglos unidimensionales de tipos de datos enumerados o enteros.	
Operador	**Significado**
=	Igual
/=	Diferente
<	Menor
<=	Menor o Igual
>	Mayor
>=	Mayor o Igual

Enunciado IF-THEN-ELSE

Cinvestav

La construcción: **if-then-else**

La construcción if-then-else sirve para seleccionar una operación con base al análisis (evaluación lógica ➔ Cierto o Falso) de una condición.



	Ejemplo Nº 10 - La construcción **if-then-else** *Comparador de dos palabras con long. de 2-bits*
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** comp **is**
4	**port** (a,b: **in** std_logic_vector (1 downto 0);
5	c: **out** std_logic);
6	**end** comp;
7	**architecture** funcional **of** comp **is**
8	**begin**
9	compara: **process** (a,b)
10	**begin**
11	**if** a = b **then**
12	c <= ‘1’;
13	**else**
14	c <= ‘0’;
15	**end if**;
16	**end process** compara;
17	**end** funcional;

Lista-Sensitiva

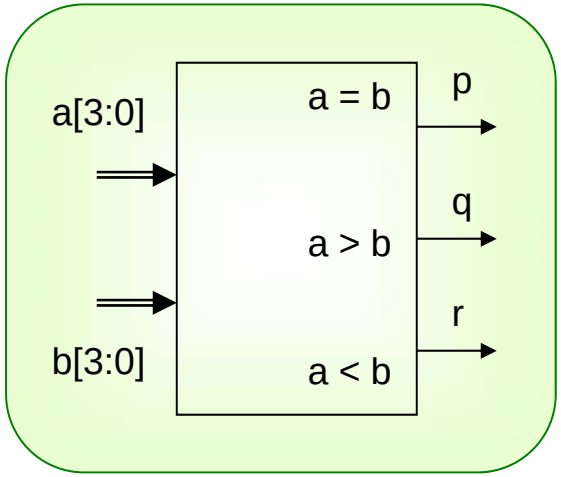
Señales (incluyendo puertos) leídas por el proceso.

Enunciado IF-THEN-ELSIF-THEN-ELSE

Cinvestav

La construcción:

if-then-elsif-then-else



¿Qué valores tienen las otras salidas en este instante?

La construcción **if-then-elsif-then-else** se utiliza cuando se requiere analizar más de una condición de entrada.

Ejemplo Nº 11 - La construcción **if-then-elsif-then-else**

Comparador de Magnitud 2-Words de 4-bits

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity comp4 is
4          port (a,b: in std_logic_vector (3 downto 0);
5                  p,q,r: out std_logic);
6  end comp4;
7  architecture arq_comp4 of comp4 is
8  begin
9  process (a,b)
10 begin
11         if (a = b) then
12                 p<= '1';
13         elsif (a > b) then
14                 q <= 1';
15         else
16                 r<= '1';
17         end if;
18 end process;
19 end arq_comp4;
```

¿Qué circuito es inferido?

Enunciado IF-THEN-ELSIF-THEN-ELSE

Cinvestav

A1	A0	B1	B0	S1	S0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1

Ejemplo Nº 12 - La construcción **if-then-elsif-then-else**

Comparador de Magnitud 2-Words de 2-Bits / Salida Codificada

```
library ieee;
use ieee.std_logic_1164.all;
entity comp is
        port (A,B: in std_logic_vector (1 downto 0);
                S: out std_logic_vector (1 downto 0));
end comp;
architecture arq_comp of comp is
begin
process (A,B)
begin
        if (A = B) then
                S<= “11”;
        elsif (A < B) then
                S <= “01”;
        else
                S <= “10”;
        end if;
end process;
end arq_comp;
```

Operación deseada:

Si: A = B entonces S = 11

Si: A < B entonces S = 01

Si: A > B entonces S = 10

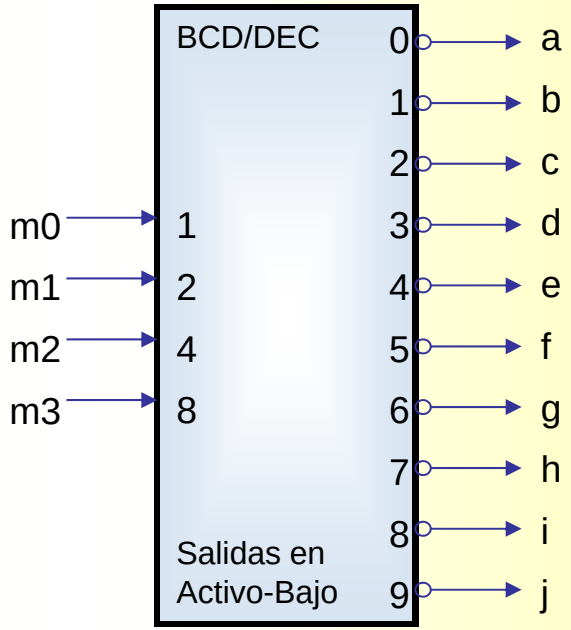
Enunciado IF-THEN-ELSIF-THEN-ELSE

Cinvestav

Decodificadores: BCD a Decimal

Ejemplo Nº 13 – Decodificador de BCD a Decimal

```
library ieee;
use ieee.std_logic_1164.all;
entity deco is
        port (m: in std_logic_vector (3 downto 0);
                a, b, c, d, e, f, g, h, i, j: out std_logic);
end deco;
architecture arqdeco of deco is
begin
process (m) begin
        a <= '1';
        b <= '1';
        c <= '1';
        d <= '1';
        e <= '1';
        f  <= '1';
        g <= '1';
        h <= '1';
        i  <= '1';
        j  <= '1';
```




```
if      m = "0000" then
          a <= '0';
elsif   m = "0001" then
          b <= '0';
elsif   m = "0010" then
          c <= '0';
elsif   m = "0011" then
          d <= '0';
elsif   m = "0100" then
          e <= '0';
elsif   m = "0101" then
          f <= '0';
elsif   m = "0110" then
          g <= '0';
elsif   m = "0111" then
          h <= '0';
elsif   m = "1000" then
          i <= '0';
elsif   m = "1001" then
          j <= '0';
end if;
end process;
end arqdeco;
```

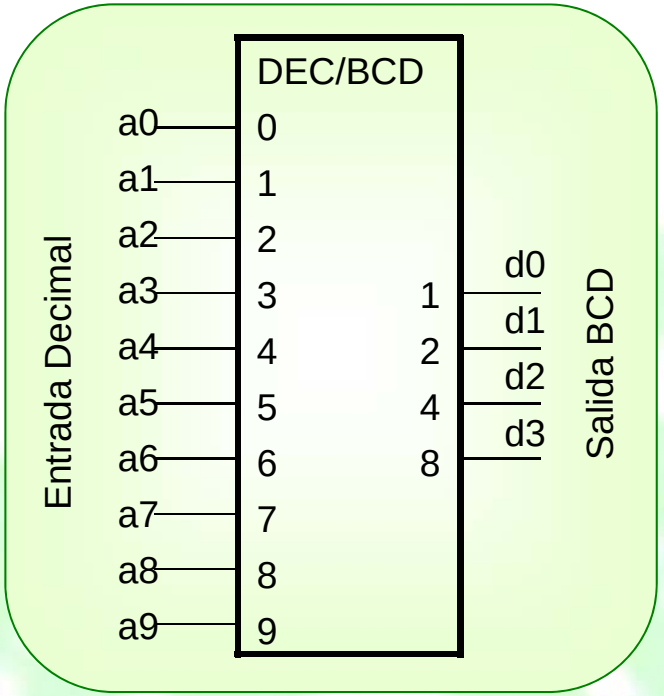
Enunciado IF-THEN-ELSIF-THEN-ELSE

Cinvestav

Codificadores: Decimal a BCD

Ejemplo Nº 14 – Codificador Decimal a BCD

```
library ieee;
use ieee.std_logic_1164.all;
entity codif is
        port (a: in std_logic_vector (9 downto 0);
                d: out std_logic_vector (3 downto 0));
end codif;
architecture arqcodif of codif is
begin
process (a)
begin
        if      a = “0000000001” then d <= “0000”;
        elsif a = “0000000010” then d <= “0001”;
        elsif a = “0000000100” then d <= “0010”;
        elsif a = “0000001000” then d <= “0011”;
        elsif a = “0000010000” then d <= “0100”;
        elsif a = “0000100000” then d <= “0101”;
        elsif a = “0001000000” then d <= “0110”;
        elsif a = “0010000000” then d <= “0111”;
        elsif a = “0100000000” then d <= “1000”;
        elsif a= “1000000000”  then d <= “1001”;
        else d <= “1111”;
        end if;
end process;
end arqcodif;
```

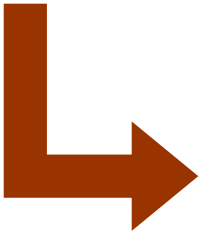


Formato del enunciado CASE

Cinvestav

Enunciados CASE:

La construcción **case** - **when** ejecuta una o varias instrucciones secuenciales que dependen del valor de una sola expresión.



SINTAXIS

case *expresion* **is**
when *caso1* **=>** *enunciados secuenciales;*
{when *caso2* **=>** *enunciados secuenciales;* **}**
[when others => *enunciados secuenciales;* **]**
end case;

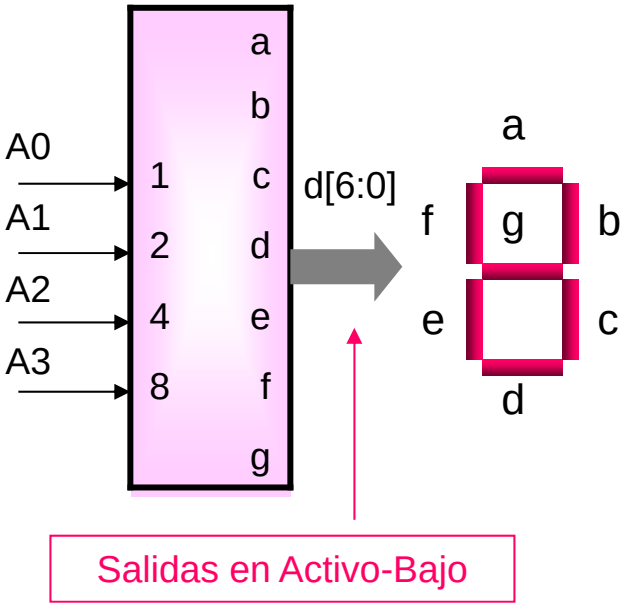
EJEMPLO:

```
case puntuacion is
        when 9 to 10        => acta <=“Sobresaliente”;
        when 8 downto 7     => acta <=“Notable”;
        when 5  |  6        => acta <=“Aprobado”;
        when 0              => acta <=“No presento”;
        when others         => acta <=“Suspendido”;
end case;
```

Enunciado CASE

Cinvestav

Decodificadores: BCD a 7-Segmentos



Código BCD (A)				Segmentos del Display (d)						
A3	A2	A1	A0	d6	d5	d4	d3	d2	d1	d0
				a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0

Enunciado CASE

Cinvestav

Ejemplo Nº 15 – Decodificador
BCD a 7-Segmnetos
(Uso de construcción **case-when**)

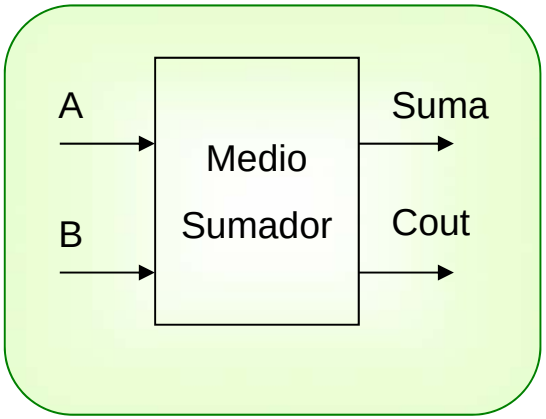
```
library ieee;
use ieee.std_logic_1164.all;
entity decobcd_7s is
        port (A: in std_logic_vector (3 downto 0);
                d: out std_logic_vector (6 downto 0));
end decobcd_7s;
architecture arqdeco of decobcd_7s is
begin
process (A) begin
        case A is
        when “0000” => d <= “0000001”;
        when “0001” => d <= “1001111”;
        when “0010” => d <= “0010010”;
        when “0011” => d <= “0000110”;
        when “0100” => d <= “1001100”;
```

Decodificadores: BCD a 7-Segmentos

```
        when “0101” => d <= “0100100”;
        when “0110” => d <= “0100000”;
        when “0111” => d <= “0001110”;
        when “1000” => d <= “0000000”;
        when “1001” => d <= “0000100”;
        when others => d <= “1111111”;
        end case;
end process;
end arqdeco;
```

Construcción **case-when**: En esta construcción se evalúa la expresión especificada (***case***) y el valor que se obtenga se compara con los asociados a las diferentes opciones descritas. Aquella opción (***when***) que coincida con dicho valor, le serán ejecutados sus enunciados secuenciales adyacentes.

Ejemplo: Medio Sumador



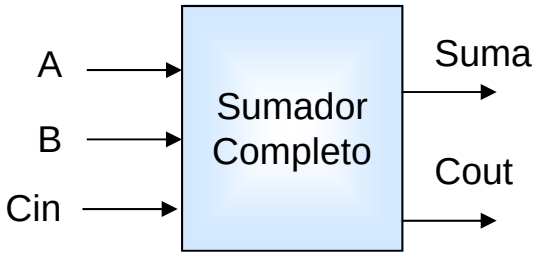
A	B	Suma	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Ejemplo Nº 16 – Medio Sumador

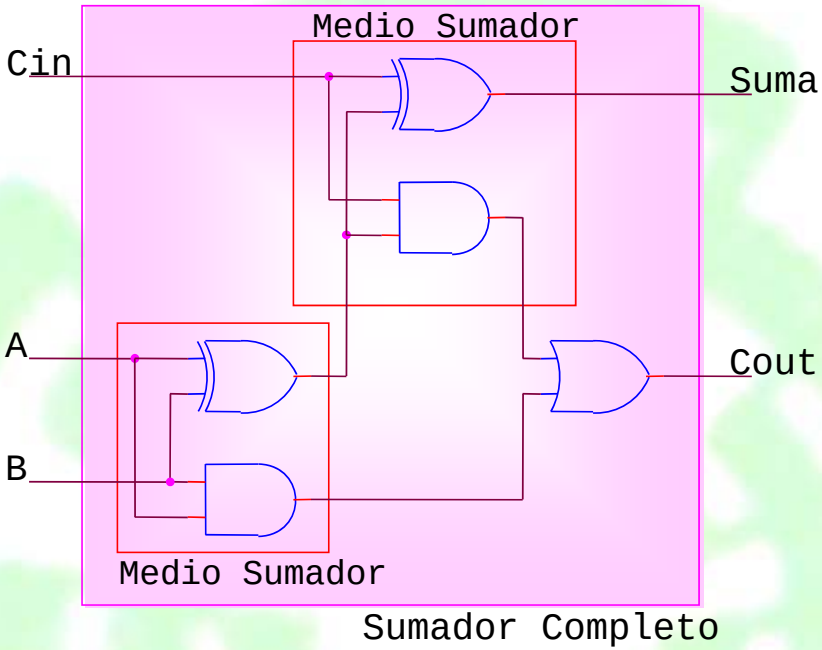
```
library ieee;
use ieee.std_logic_1164.all;
entity med_sum is
        port (A,B: in std_logic;
                Suma, Cout: out std_logic);
end med_sum;
architecture arq_sum of m_sum is
begin
        Suma <= A xor B;
        Cout <= A and B;
end arq_sum;
```

$$Suma = A \oplus B$$
$$Cout = A * B$$

Ejemplo: Sumador Completo



A	B	Cin	Suma	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$$\text{Suma} = \overline{A}\,\overline{B}\,Cin + \overline{A}\,B\overline{Cin} + A\overline{B}\,\overline{Cin} + ABCin = A \oplus B \oplus Cin$$

$$\text{Cout} = \overline{A}\,B\,Cin + A\,\overline{B}Cin + AB\,\overline{Cin} + ABCin = AB + (A \oplus B)Cin$$

Ejemplo: Sumador Completo

Cinvestav

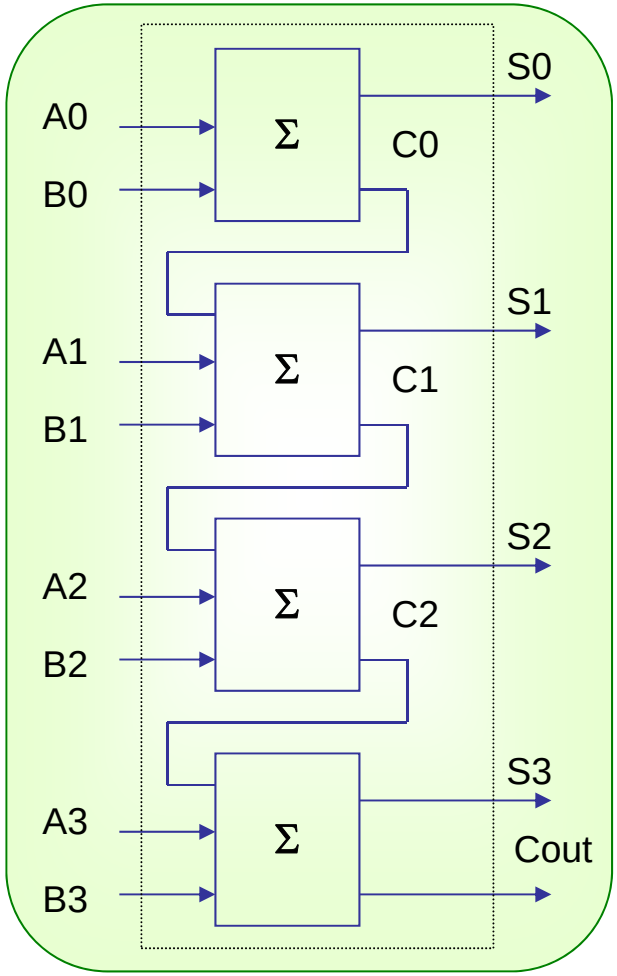
$$\text{Suma} = \overline{A}\,\overline{B}\,Cin + \overline{A}\,B\overline{Cin} + A\overline{B}\,\overline{Cin} + ABCin = A \oplus B \oplus Cin$$

$$\text{Cout} = \overline{A}\,B\,Cin + A\,\overline{B}Cin + AB\,\overline{Cin} + ABCin = AB + (A \oplus B)Cin$$

Ejemplo Nº 17 – Sumador Completo

```
library ieee;
use ieee.std_logic_1164.all;
entity sum is
        port (A, B, Cin: in std_logic;
                Suma, Cout: out std_logic);
end sum;
architecture arq_sum of sum is
begin
        Suma <= A xor B xor Cin;
        Cout <= (A and B) or ((A xor B) and Cin);
end arq_sum;
```

Ejemplo: Sumador Paralelo de 4 bits



Ejemplo Nº 18 – Sumador Paralelo de 4 bits

```
library ieee;
use ieee.std_logic_1164.all;
entity suma is
        port (A, B: in std_logic_vector (3 downto 0);
                S: out std_logic_vector (3 downto 0);
                Cout: out std_logic);
end suma;
architecture arqsuma of suma is
signal C: std_logic_vector (2 downto 0);
begin
        S(0) <= A(0) xor B(0);
        C(0) <= A(0) and B(0);
        S(1) <= (A(1) xor B(1)) xor C(0);
        C(1) <= (A(1) and B(1)) or (C(0) and (A(1) xor B(1)));
        S(2) <= (A(2) xor B(2)) xor C(1);
        C(2) <= (A(2) and B(2)) or (C(1) and (A(2) xor B(2)));
        S(3) <= (A(3) xor B(3)) xor C(2);
        Cout <= (A(3) and B(3)) or (C(2) and (A(3) xor B(3)));
end arqsuma;
```

Declaraciones de Señales (**signal**): Especifican señales que permiten conectar los diferentes tipos de enunciados concurrentes (*asignación de señales, bloques, procesos y llamadas a componentes o procedimientos*) de que consta una arquitectura.

Ejemplo: Sumador Paralelo de 4 bits

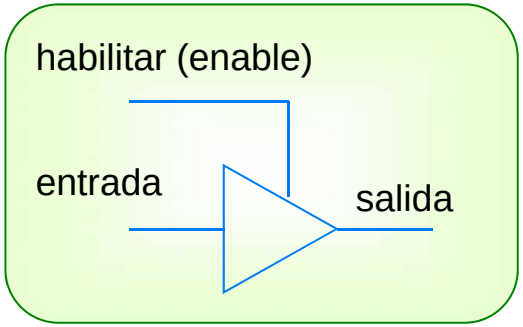
Cinvestav

Operadores Aritméticos	
Operador	Descripción
+	Suma
-	Resta
/	División
*	Multiplicación
**	Potencia

Ejemplo Nº 19 – Sumador Paralelo de 4 bits con Cout
(Uso Operador Aritmético '+')

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

entity sum4b_arit is
        port (A, B: in std_logic_vector (3 downto 0);
                S: out std_logic_vector (3 downto 0);
                Cout: out std_logic );
end sum4b_arit;
architecture arqsum of sum4b_arit is
signal sum: std_logic_vector (4 downto 0);
begin
        sum <= '0'&A + '0'& B;
        S <= sum (3 downto 0);
        Cout <= sum(4);
end arqsum;
```

Ejemplo: Buffer (salida de 3 estados)



Tipos Lógicos Estándares	
'U'	Valor No-Inicializado
'X'	Valor Fuerte Desconocido
'0'	0 Fuerte
'1'	1 Fuerte
'Z'	Alta Impedancia
'W'	Valor Débil Desconocido
'L'	0 Débil
'H'	1 Débil
'-'	No Importa (Don't Care)

Ejemplo Nº 20 – Buffer Salida de 3-Estados

```
library ieee;
use ieee.std_logic_1164.all;
entity tri_est is
        port (enable, entrada: in std_logic;
                salida: out std_logic);
end tri_est;
architecture arq_buffer of tri_est is
begin
process (enable, entrada)
begin
        if (enable = '0') then
                salida <= 'Z';
        else
                salida <= entrada;
        end if;
end process;
end arq_buffer;
```

El tipo de dato ***bit*** no soporta el valor 'Z', por lo que se debe utilizar el tipo ***std_logic***, que si lo soporta.

Resumen de Circuitos combinatorios

Cinvestav

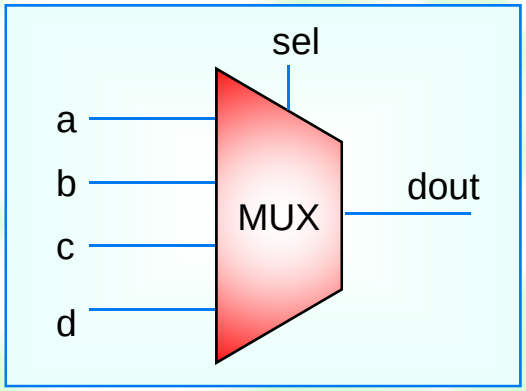
Multiplexores

```
with sel select
  dout <= a when “00”,
          b when “01”,
          c when “10”,
          d when “11”,
          (others => ‘x’) when others;
```

```
process(sel, a, b, c, d)
begin
  case sel is
    when “00”   => dout <= a;
    when “01”   => dout <= b;
    when “10”   => dout <= c;
    when “11”   => dout <= d;
    when others => dout <= (others => ‘X’);
  end case;
end process;
```

```
process(sel, a, b, c, d)
begin
  if (sel = “00”) then
    dout <= a;
  elsif (sel = “01”) then
    dout <= b;
  elsif (sel = “10”) then
    dout <= c;
  elsif (sel = “11”) then
    dout <= d;
  else
    dout <= (others => ‘X’);
  end if;
end process;
```

```
dout <= a when sel = “00” else
        b when sel = “01” else
        c when sel = “10” else
        d when sel = “11” else
        (others => ‘x’);
```



Resumen de Circuitos combinatorios

Cinvestav

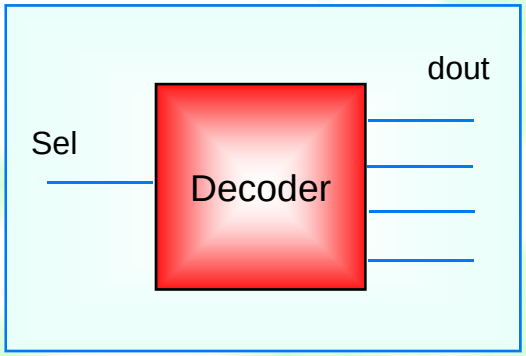
Decodificadores

```
dout <= "0001" when sel = "00" else
        "0010" when sel = "01" else
        "0100" when sel = "10" else
        "1000" when sel = "11" else
        (others => 'x');
```

```
process(sel)
begin
  dout <= "0000";
  case sel is
    when "00"   => dout(0) <= '1';
    when "01"   => dout(1) <= '1';
    when "10"   => dout(2) <= '1';
    when "11"   => dout(3) <= '1';
    when others => dout    <= "XXXX";
  end case;
end process;
```

```
process(sel)
begin
  dout <= "0000";
  if (sel = "00") then
    dout(0) <= '1';
  elsif (sel = "01") then
    dout(1) <= '1';
  elsif (sel = "10") then
    dout(2) <= '1';
  elsif (sel = "11") then
    dout(3) <= '1';
  else
    dout <= "XXXX";
  end if;
end process;
```

```
with sel select
  dout <= "0001" when "00",
          "0010" when "01",
          "0100" when "10",
          "1000" when "11",
          "XXXX" when others;
```



Resumen de Circuitos combinatorios

Cinvestav

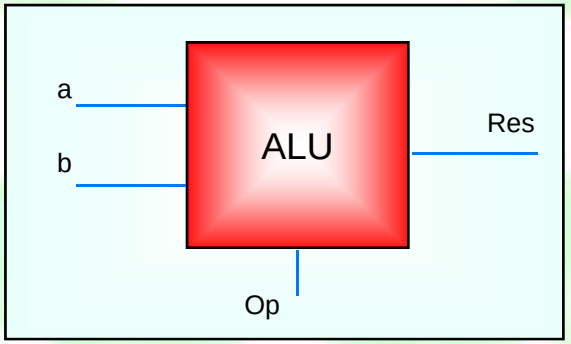
Unidad Aritmética Lógica

```
process(a, b, Op)
begin
  case Op is
    when suma  => Res <= a + b;
    when resta => Res <= a - b;
    when andl  => Res <= a and b;
    when orl   => Res <= a or b;
    when shl   => Res <= sll(a, 1);
    when shr   => Res <= srl(a, 1);
  end case;
end process;
```

```
Res <= a + b     when Op = suma  else
       a - b     when Op = resta else
       a and b   when Op = andl  else
       a or b    when Op = orl   else
       sll(a, 1) when Op = shl   else
       srl(a, 1);
```

```
with Op select
  Res <= a + b     when suma,
         a - b     when resta,
         a and b   when andl,
         a or b    when orl,
         sll(a, 1) when shl,
         srl(a, 1) when shr;
```

```
process(a, b, Op)
begin
  if (Op = suma)     then Res = a + b;
  elsif (Op = resta) then Res = a - b;
  elsif (Op = andl)  then Res = a and b;
  elsif (Op = orl)   then Res = a or b;
  elsif (Op = shl)   then Res = sll(a, 1);
  elsif (Op = shr)   then Res = srl(a, 1);
  end if;
end process;
```



Cinvestav

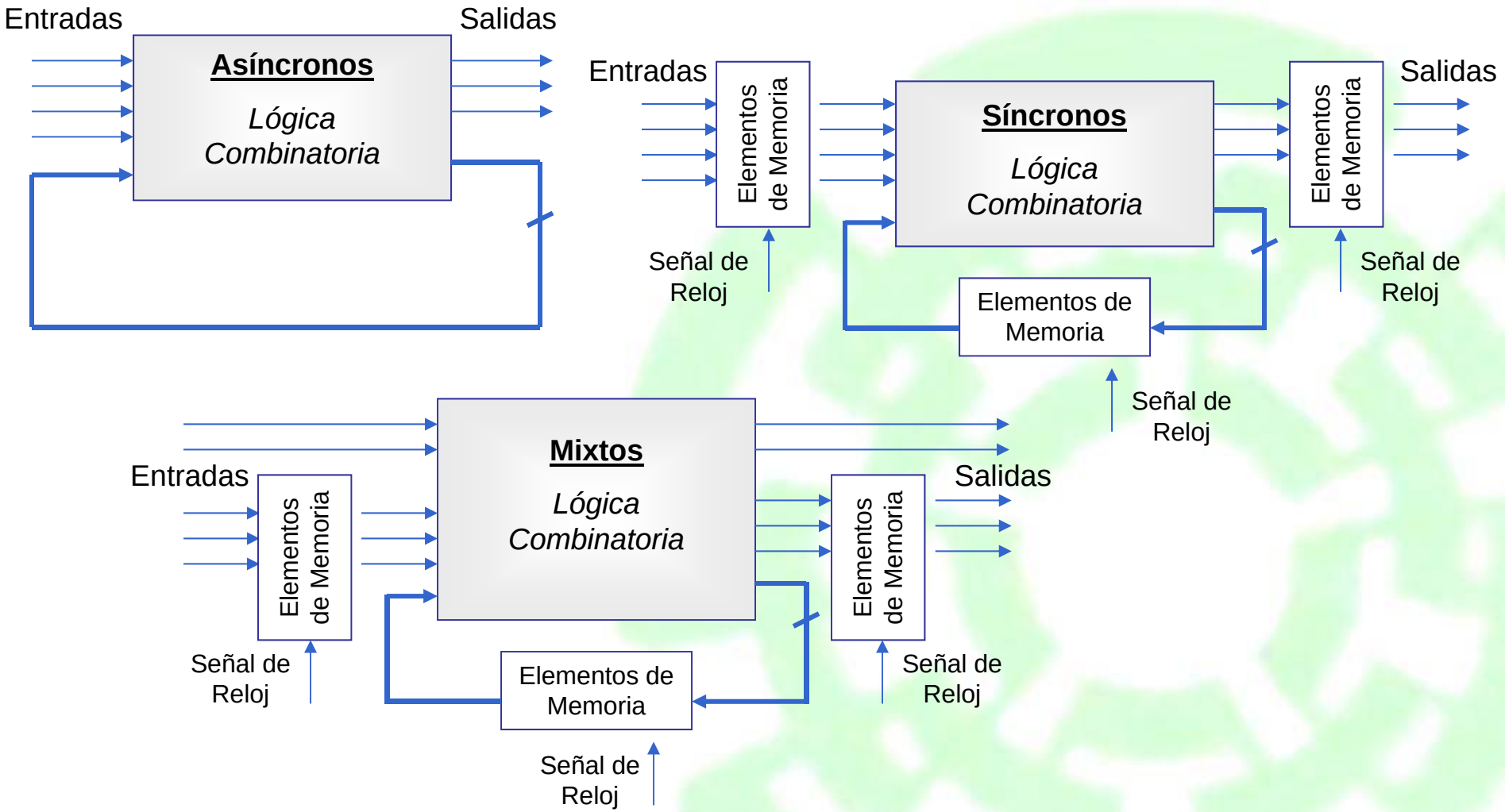
Capítulo 4

Circuitos Lógicos Secuenciales

¿Qué es un Circuito Lógico Secuencial?

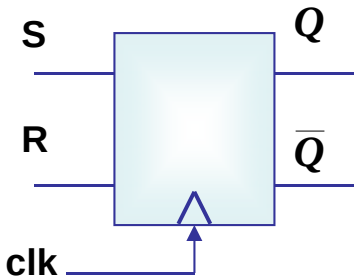
Cinvestav

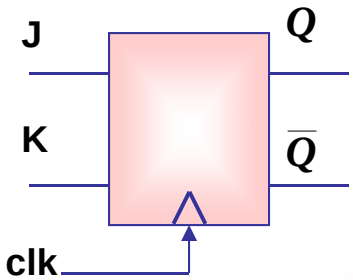
Clasificación

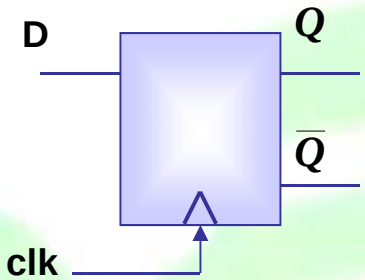


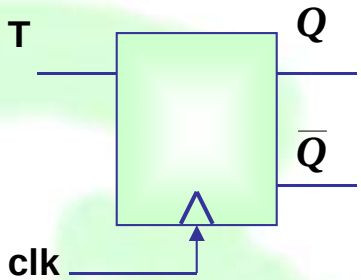
Elementos de Memoria: Flip-Flops

Cinvestav









S	R	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

J	K	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

D	Q_t	Q_{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

T	Q_t	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

Atributos

Cinvestav

Los elementos en VHDL, como señales, variables, etc., pueden tener información adicional llamada ***atributos.*** Estos atributos están asociados a estos elementos del lenguaje y se manejan en VHDL mediante comilla simple (**'**) .

SINTAXIS **name'*atributo***

Atributo : predefinido o definido por el usuario.

Ejemplos de algunos atributos predefinidos:

Suponiendo que **t** es un tipo enumerado, entero, flotante, o físico.

t'left Límite izquierdo del tipo **t**

t'low Límite inferior del tipo **t**

Suponiendo que **s** es una señal, se pueden utilizar los siguientes atributos.

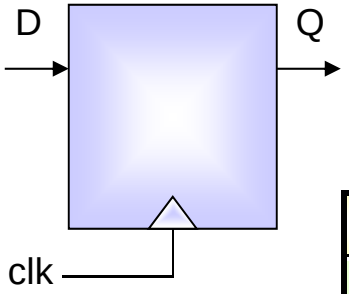
s'event Devuelve **true** si se ha producido un cambio en **s**

Flip-Flop tipo D

Cinvestav

En el *Diseño Secuencial* con VHDL, las construcciones:

if-then-else / if-then-elsif-then son las más utilizadas.



D	Q_t	Q_{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

Instrucciones equivalentes:

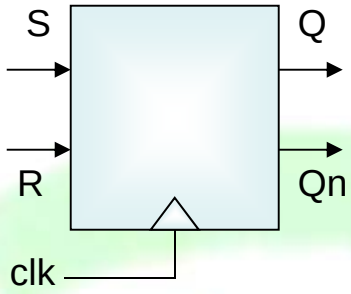
```
if rising_edge(clk)   – verdadero con el flanco de subida
if ( clk'event and clk= '1' and clk'last_value= '0' )

 if falling_edge(clk)   – verdadero con el flanco de bajada
if ( clk'event and clk= '0' and clk'last_value= '1' )
```

Ejemplo Nº 1 – Flip-Flop tipo D

```
library ieee;
use ieee.std_logic_1164.all;
entity ffd is
        port (D, clk: in std_logic;
                Q: out std_logic);
end ffd;
architecture arq_ffd of ffd is
begin
        process (clk,D)
        begin
        if (clk'event and clk='1') then
                Q <= D;
        end if;
        end process;
end arq_ffd;
```

Flip-Flop tipo SR

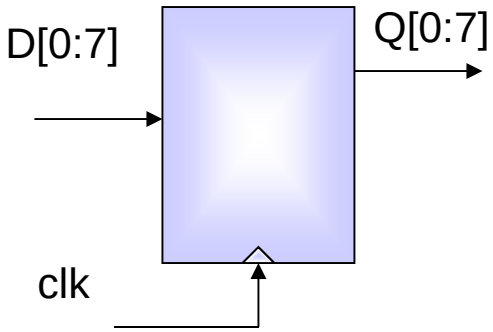
S	R	Q_t	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	-



Ejemplo Nº 2 – Flip-Flop tipo SR

```
library ieee;
use ieee.std_logic_1164.all;
entity ffsr is
        port (S, R, clk: in std_logic;
                Q, Qn: buffer std_logic);
end ffsr;
architecture arq_ffsr of ffsr is
begin
        process (clk, S, R)
        begin
        if (clk'event and clk='1') then
                if (S = '0' and R = '1')  then
                        Q <= '0';
                        Qn <= '1';
                elsif (S = '1' and R = '0')  then
                        Q <= '1';
                        Qn <= '0';
                elsif (S = '0' and R = '0')  then
                        Q <= Q;
                        Qn <= Qn;
                else
                        Q <= '-';
                        Qn <= '-';
                end if;
        end if;
        end process;
end arq_ffsr;
```

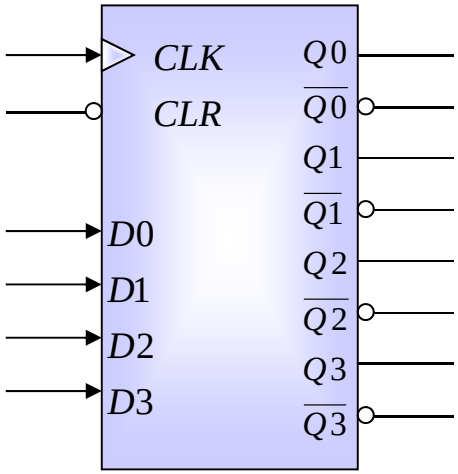
Registros Paralelo de 8 bits



Ejemplo Nº 3 – Registro Paralelo de 8-Bits

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
        port (D: in std_logic_vector (0 to 7);
                clk: in std_logic;
                Q: out std_logic_vector (0 to 7));
end reg;
architecture arqreg of reg is
begin
        process (clk,D)
        begin
        if (clk'event and clk='1') then
                Q <= D;
        end if;
        end process;
end arqreg;
```

Registros Paralelo de 4 bits con Clear

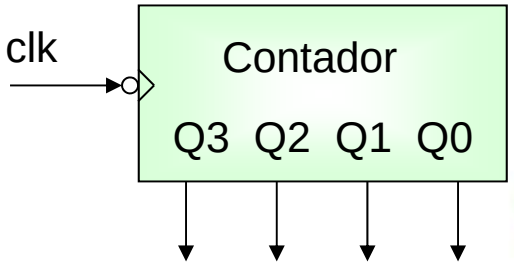


CLR	D	Q	Qn
0	'-'	0	1
1	D	D	Dn

Ejemplo Nº 4 – Registro Paralelo de 4-Bits con 'Clear'

```
library ieee;
use ieee.std_logic_1164.all;
entity reg4 is
        port (D: in std_logic_vector (3 downto 0);
                CLK, CLR: in std_logic;
                Q, Qn: out std_logic_vector (3 downto 0));
end reg4;
architecture arq_reg4 of reg4 is
begin
        process (CLK, CLR, D) begin
        if (CLK'event and CLK='1') then
                if (CLR = '1') then
                        Q <= D;
                        Qn <= not D;
                else
                        Q <= "0000";
                        Qn <= "1111";
                end if;
        end if;
        end process;
end arq_reg4;
```

Contadores



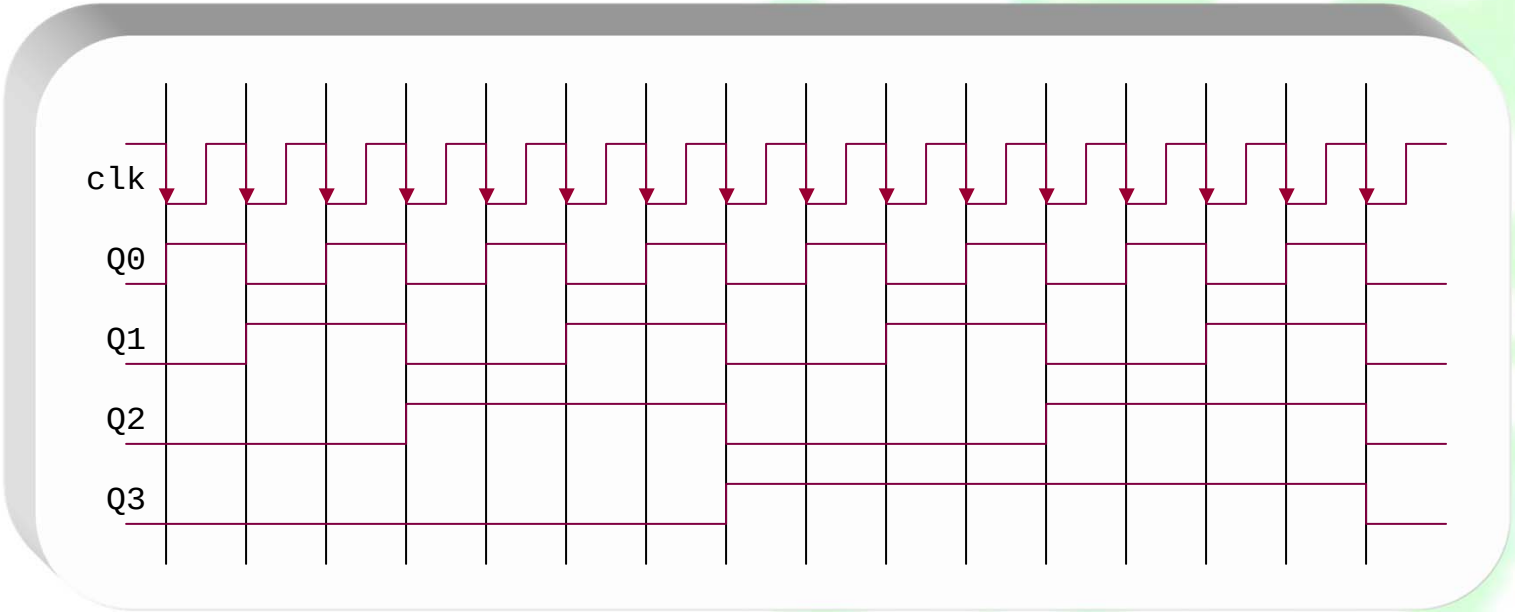


Diagrama de tiempo del contador de 4 bits

Contador Ascendente

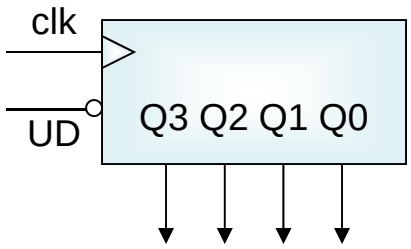
Cinvestav

Ejemplo Nº 5 – Contador de 4-Bits

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity cont4 is
        port (clk: in std_logic;
                Q: buffer std_logic_vector (3 downto 0));
end cont4;
architecture arqcont of cont4 is
begin
        process (clk)
                begin
                if (clk'event and clk = '0') then
                        Q <= Q +1;
                end if;
        end process;
end arqcont;
```

Es verdadera con el flanco de bajada de clk

Contador Ascendente/Descendente

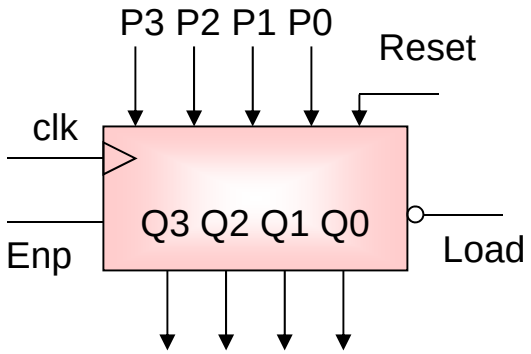


UD	Acción
0	Cuenta Ascendente
1	Cuenta Descendente

Ejemplo Nº 6 – Contador Ascendente/Descendente de 4-Bits

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity contador is
        port (clk: in std_logic;
                UD: in std_logic;
                Q: buffer std_logic_vector (3 downto 0));
end contador;
architecture arq_contador of contador is
begin
        process (UD, clk) begin
                if (clk'event and clk = '1') then
                        if (UD = '0') then
                                Q <= Q +1;
                        else
                                Q <= Q -1;
                        end if;
                end if;
        end process;
end arq_contador;
```

Contador con Reset y Carga Paralela



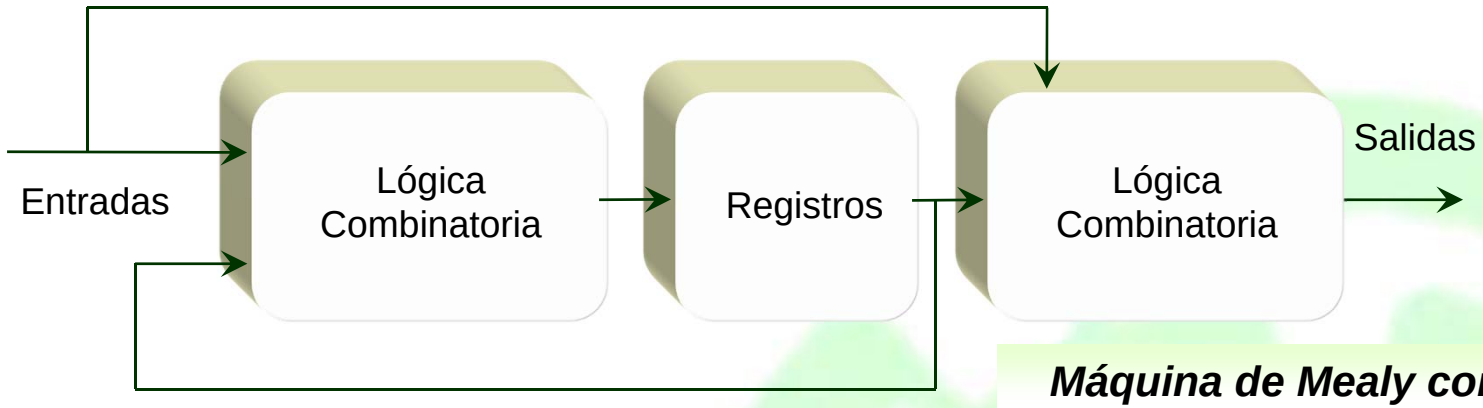
Enp	Load	Acción
0	0	Carga
0	1	Mantiene Estado
1	0	Carga
1	1	Cuenta

Ejemplo Nº 7 – Contador de 4-bits con reset y carga en paralelo

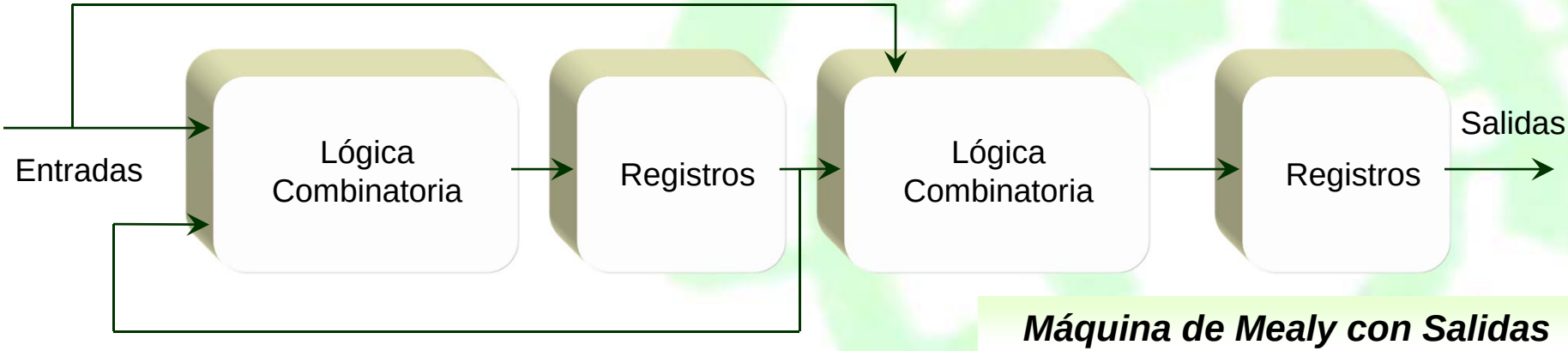
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity cont is
        port (P: in std_logic_vector (3 downto 0);
                clk, Load, Enp, Reset: in std_logic;
                Q: buffer std_logic_vector (3 downto 0));
end cont;
architecture arq_cont of cont is
begin
        process (clk, Reset, Load, Enp, P) begin
        if (Reset = '1') then
                Q <= "0000";
        elsif (clk'event and clk = '1') then
                        if (Load = '0' and Enp = '-') then
                                Q <= P;
                        elsif (Load = '1' and Enp = '0') then
                                Q <= Q;
                        elsif (Load = '1' and Enp = '1') then
                                Q <= Q + 1;
                        end if;
        end if;
        end process;
end arq_cont;
```

Operación Asíncrona

Máquinas Secuenciales



Máquina de Mealy con Salidas Asíncronas

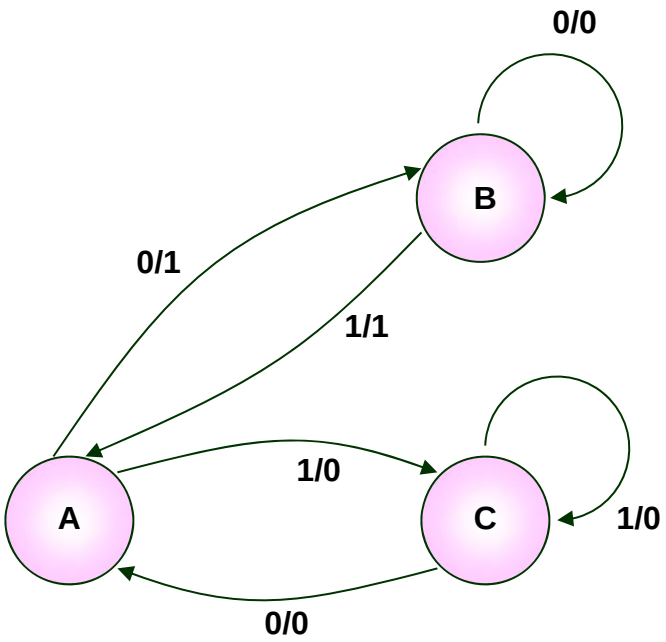


Máquina de Mealy con Salidas Síncronas

Máquinas Secuenciales

Ejemplo Nº 8

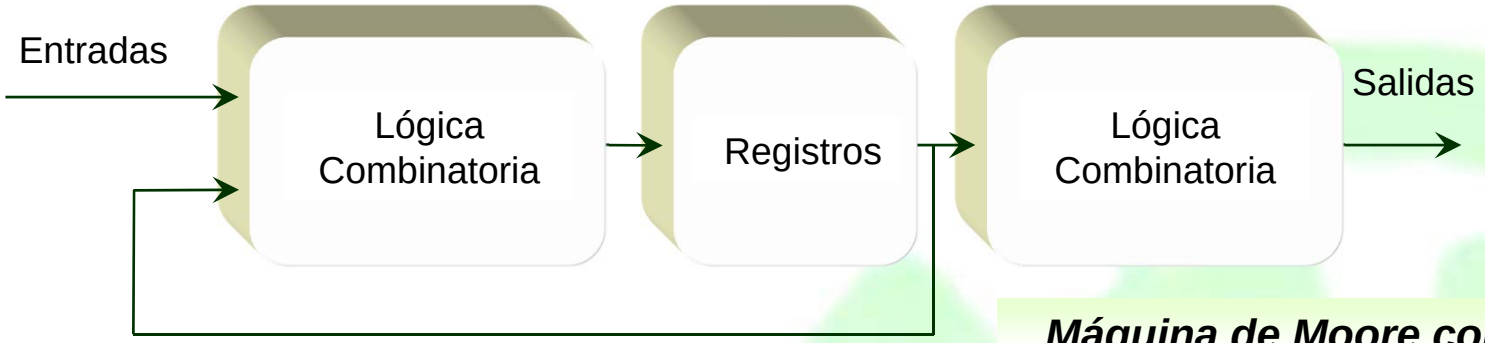
Representación de una Máquina de Mealy



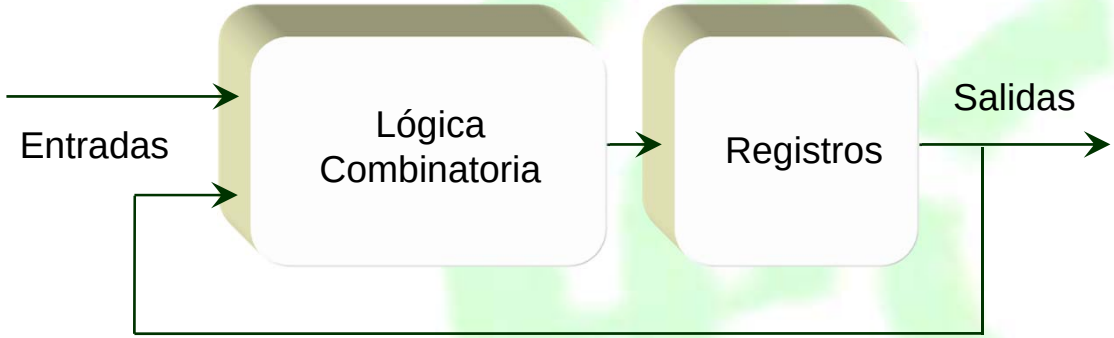
Estado Presente	Entrada R	
	0	1
A	B/1	C/0
B	B/0	A/1
C	A/0	C/0

Próximo Estado / Salida S

Máquinas Secuenciales



Máquina de Moore con Salidas Asíncronas

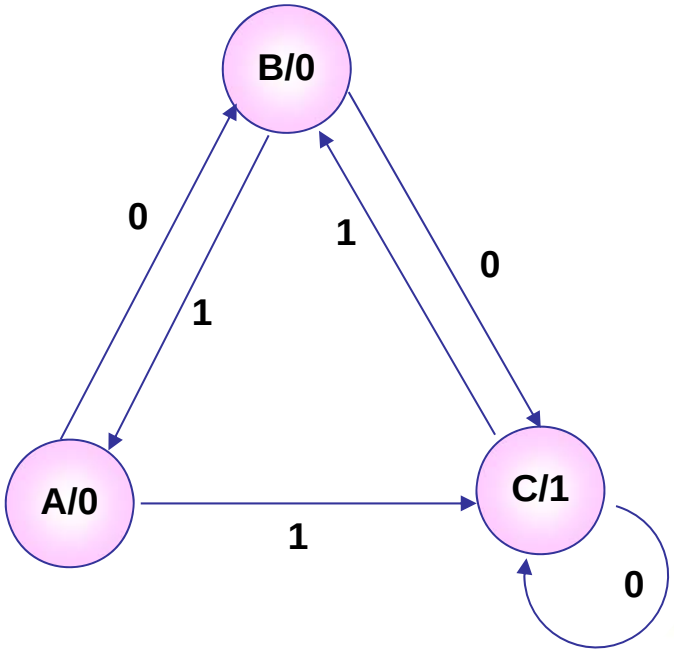


Máquina de Moore con Salidas Síncronas

Máquinas Secuenciales

Ejemplo Nº 9

Representación de una Máquina de Moore



Estado Presente	Entrada R		Salida S (Para el Estado Presente)
	0	1	
A	B	C	0
B	C	A	0
C	C	B	1

Próximo Estado

Diseño de Circuitos Secuenciales Síncronos

Cinvestav

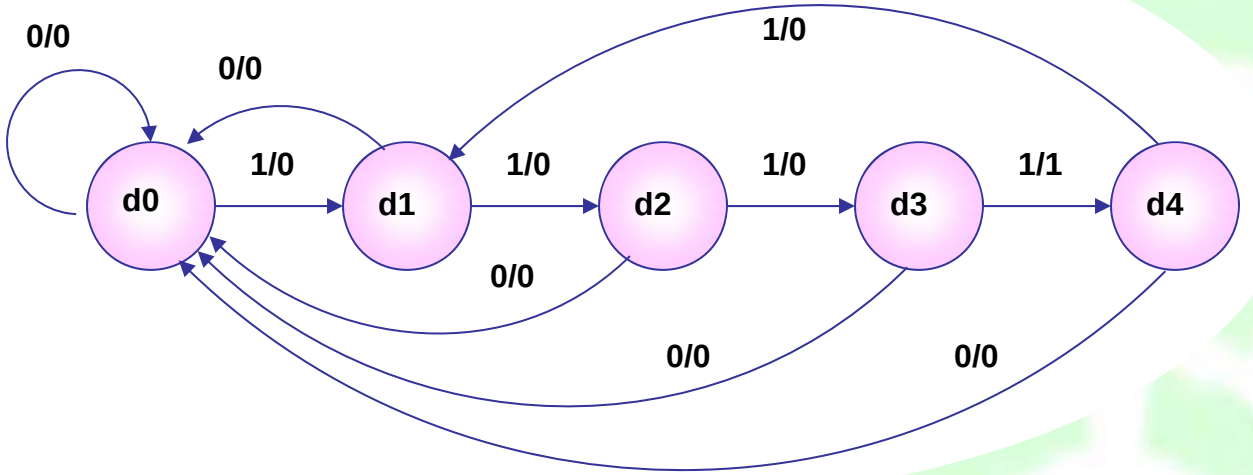
Flujo de Diseño de una Máquina de Estados Finitos (FSM)	
1	Dibujar el Diagrama de Transiciones de Estados.
2	Verificación del Diagrama de Estados: • Asegurarse que todos los estados están representados. • La función OR de todas las transiciones que dejan un estado = 1 (TRUE) – Esto permite determinar si existe una salida (por lo menos) de un estado dado, una vez que se ha llegado a él. • La función XOR de todas las transiciones-salida de un estado = 1 (TRUE) – Esto asegura que no existan condiciones en conflicto que conduzcan a tener más de una transición de salida activas en forma simultánea.
3	Asignación de Estados.
4	Descripción del Comportamiento: Uso de Lenguajes de Descripción de Hardware (HDL) – VHDL y Verilog
5	Compilación / Síntesis del Diseño – Generación de Lógica (Ecuaciones Lógicas) / Asignación de Estados
6	Simulación Funcional
7	Implementación / Realización del Diseño: Realización de la Lógica con una Tecnología y/o Dispositivos predefinidos, p.ej. Biblioteca de Celdas CMOS, Lógica Comercial, PLDs/CPLDs/FPGAs (para estos dispositivos, esta fase se le conoce como: 'Mapping, Place & Route'), Microcontroladores, etc.
8	Simulación Temporizada

Diseño de Circuitos Secuenciales Síncronos

Cinvestav

Circuito Secuencial que detecta 4-Unos (1's) consecutivos

Ejemplo Nº 10



Edo. Presente	Edo. Futuro		s (Salida)	
	a = 0	a = 1	a = 0	a = 1
d0	d0	d1	0	0
d1	d0	d2	0	0
d2	d0	d3	0	0
d3	d0	d4	0	1
d4	d0	d1	0	0

Diseño de Circuitos Secuenciales Síncronos

Cinvestav

¿Cómo describir o declarar los estados (usando VHDL) a partir del Diagrama de Estados?

Para entender el proceso de declaración de los estados, se comprenderá primeramente el siguiente grupo de declaraciones

```
type std_logic is ('0','1','U','X','Z','W','L','H','-');
signal a: std_logic;
signal b: std_logic_vector (3 downto 0);
```

a puede tener cualquiera de los valores:
'0','1','U','X','Z','W','L','H','-'

b está formado por 4-bits, cada uno de los cuales puede tener cualquiera de los valores:
'0','1','U','X','Z','W','L','H','-'

Diseño de Circuitos Secuenciales Síncronos

Cinvestav

Para declarar los estados de una Máquina de Estados Finitos se realiza lo siguiente:

¿Cómo son codificados: d0, d1, d2, d3, d4?

Valores que pueden tener el *edo.presente* y el *edo.futuro* en el ejemplo Nº 10:	
edo_presente	*edo_futuro*
d0	d0
d1	d1
d2	d2
d3	d3
d4	d4

estados

Declaración de Estados en una FSM

type estados **is** (d0, d1, d2, d3, d4);

signal edo_presente, edo_futuro: estados;

Tipos de Codificación utilizados:
- One-Hot
- Compact
- Secuencial
- Gray
- Johnson
- Definido por Usuario

El Número de Bits utilizados en la codificación está en función del Número de Estados **(Tarea realizada por el Compilador o Sintetizador)***

000	d0
001	d1
010	d2
011	d3
100	d4
101	101
110	110
111	111

estados es el nombre o identificador dado por el usuario al conjunto de datos conformado por *d0, d1,d2 d3, d4*. A este tipo de datos se le conoce como Tipo de Datos *Enumerados*

edo_presente, edo_futuro son señales (signal) que pueden adquirir cualquiera de los valores (*d0, d1, d2, d3, d4*) que describen al tipo de dato enumerado identificado con el nombre de *estados*. ***edo_presente*** y ***edo_futuro*** son también datos del tipo enumerado

Diseño de Circuitos Secuenciales Síncronos

Cinvestav

Ejemplo Nº 10 – Detector de Secuencia

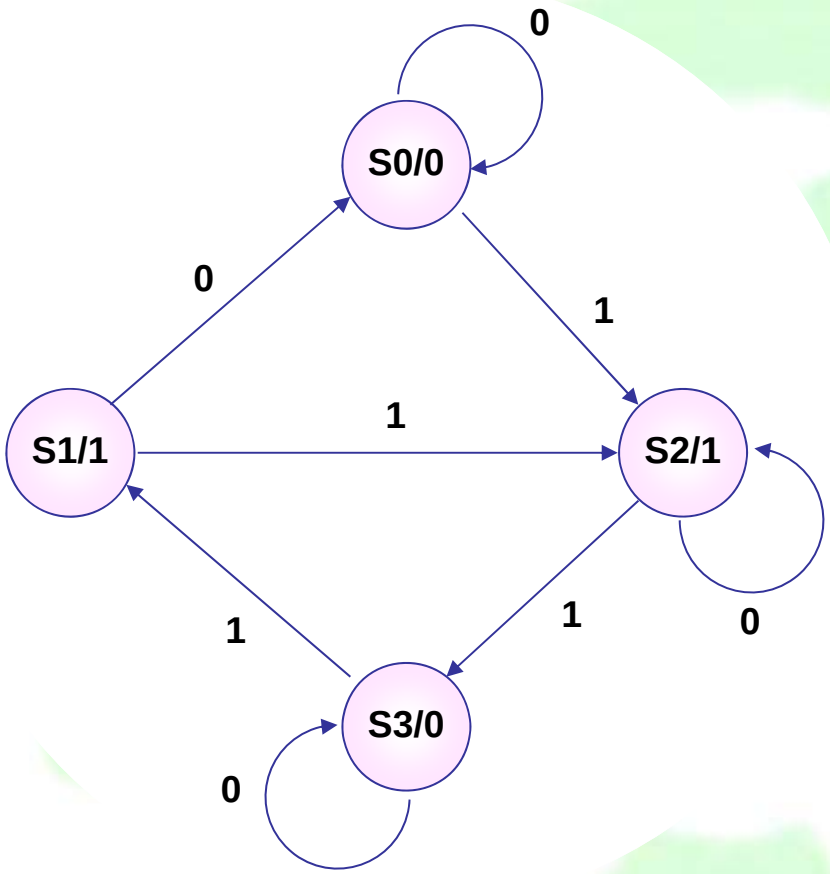
```
library ieee;
use ieee.std_logic_1164.all;
entity diagrama is
        port (clk, a: in std_logic;
                s: out std_logic);
end diagrama;

architecture arq_diagrama of diagrama is
type estados is (d0, d1, d2, d3, d4);
signal edo_presente, edo_futuro: estados;

begin
        proceso1: process (edo_presente, a) begin
        case edo_presente is
                when d0 =>
                if a = '1' then
                        edo_futuro <= d1;
                        s <= '0';
                else
                        edo_futuro <= d0;
                        s <= '0';
                end if;
                when d1 =>
                if a = '1' then
                        edo_futuro <= d2;
                        s <= '0';
                else
                        edo_futuro <= d0;
                        s <= '0';
                end if;
                when d2=>
                if a = '1' then
                        edo_futuro <= d3;
                        s <= '0';
                else
                        edo_futuro <= d0;
                        s <= '0';
                end if;
                when d3 =>
                if a = '1' then
                        edo_futuro <= d4;
                        s <= '1';
                else
                        edo_futuro <= d0;
                        s <= '0';
                end if;
                when d4 =>
                if a = '1' then
                        edo_futuro <= d1;
                        s <= '0';
                else
                        edo_futuro <= d0;
                        s <= '0';
                end if;
end case;
end process  proceso1;
        proceso2: process (clk) begin
                if (clk'event and clk = '1') then
                        edo_presente <= edo_futuro;
                end if;
        end process proceso2;
 end arq_diagrama;
```

Diseño de Circuitos Secuenciales Síncronos

Cinvestav

Ejemplo Nº 11

Máquina de Moore



Diseño de Circuitos Secuenciales Síncronos

Cinvestav

Ejemplo Nº 11 – Máquina de Moore

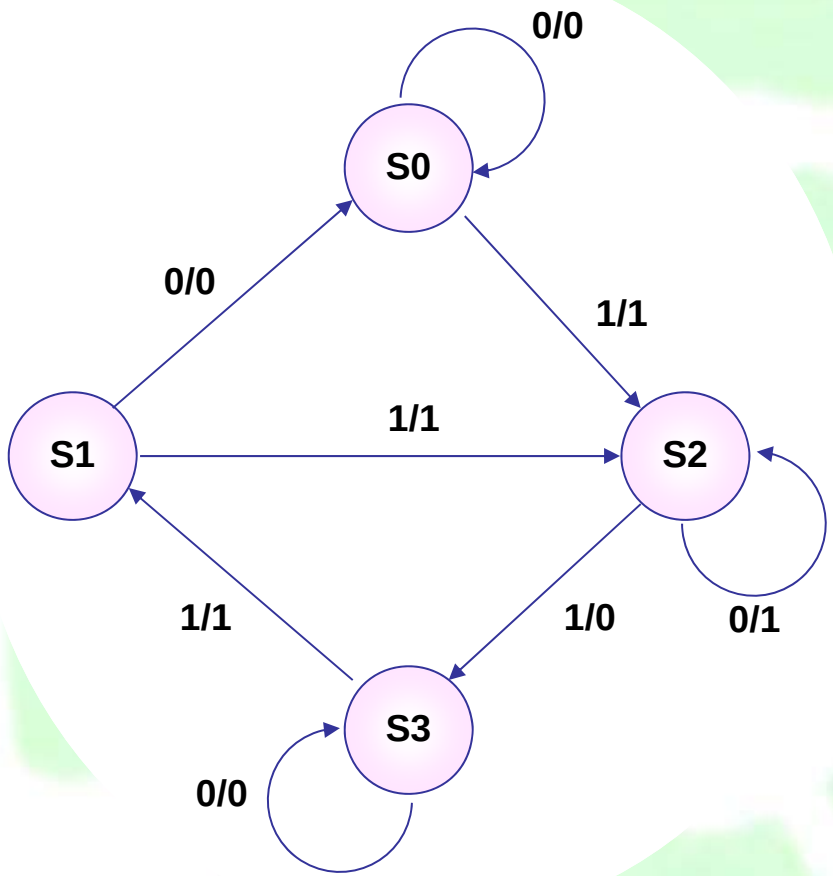
```
library ieee;
use ieee.std_logic_1164.all;
entity MOORE is
        port (A, CLK: in std_logic;
                Sal: out std_logic);
end MOORE;
architecture ARQ_MOORE of MOORE is
type Estados is (S0, S1, S2, S3);
signal Edo_Pres, Edo_Fut: Estados;
begin
        proceso1: process (Edo_Pres, A) begin
        case Edo_Pres is
                when S0 => Sal<= '0';
                if A = '0' then
                        Edo_Fut <= S0;
                else
                        Edo_Fut <= S2;
                end if;
                when S1 => Sal <= '1';
                if A = '0' then
                        Edo_Fut <= S0;
                else
                        Edo_Fut <= S2;
                end if;
                when S2=> Sal <= '1';
                if A = '0' then
                        Edo_Fut <= S2;
                else
                        Edo_Fut <= S3;
                end if;
                when S3 => Sal <= '0';
                if A = '0' then
                        Edo_Fut <= S3;
                else
                        Edo_Fut <= S1;
                end if;
        end case;
        end process  proceso1;
        proceso2: process (CLK) begin
                if (CLK'event and CLK = '1') then
                        Edo_Pres <= Edo_Fut;
                end if;
        end process proceso2;
end ARQ_MOORE;
```

Diseño de Circuitos Secuenciales Síncronos

Cinvestav

Ejemplo Nº 12

Máquina de Mealy



Diseño de Circuitos Secuenciales Síncronos

Cinvestav

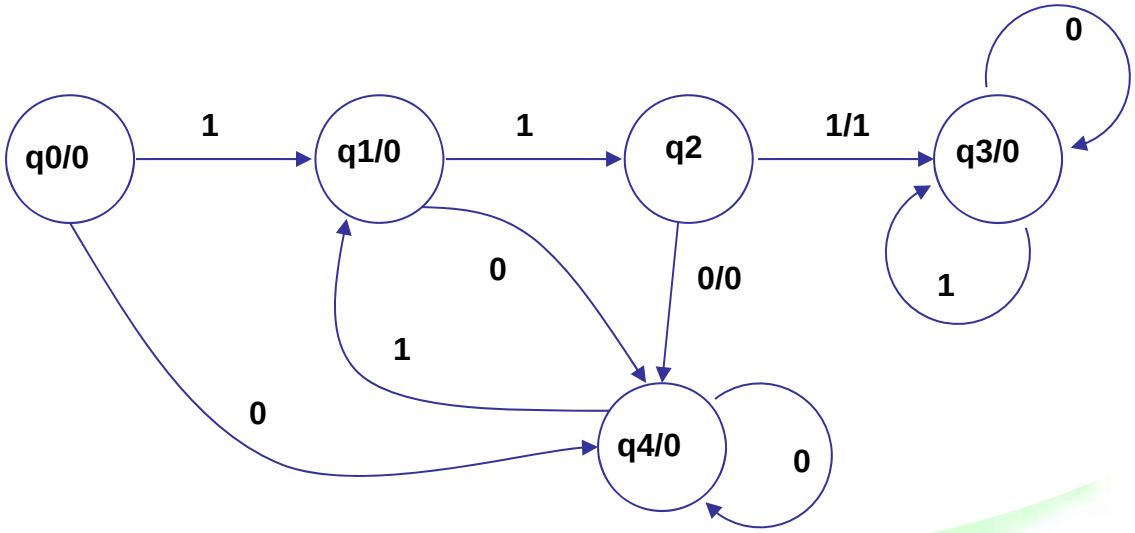
Ejemplo Nº 12 – Máquina de Mealy

```
library ieee;
use ieee.std_logic_1164.all;
entity MEALY is
        port (clk, a: in std_logic;
                sal: out std_logic);
end MEALY;
architecture ARQ_MEALY of MEALY is
type Estados is (S0, S1, S2, S3);
signal Edo_Pres, Edo_Fut: Estados;
begin
        proceso1: process (Edo_Pres, a) begin
        case Edo_Pres is
                when S0 =>
                if a = '0' then
                        sal <= '0';
                        Edo_Fut <= S0;
                else
                        sal <= '1';
                        Edo_Fut <= S2;
                end if;
                when S1 =>
                if a = '0' then
                        sal <= '0';
                        Edo_Fut <= S0;
                else
                        sal <= '1';
                        Edo_Fut <= S2;
                end if;
                when S2=>
                if a = '0' then
                        sal <= '1';
                        Edo_Fut <= S2;
                else
                        sal <= '0';
                        Edo_Fut <= S3;
                end if;
                when S3 =>
                if a = '0' then
                        sal <= '0';
                        Edo_Fut <= S3;
                else
                        sal <= '1';
                        Edo_Fut <= S1;
                end if;
        end case;
        end process  proceso1;
        proceso2: process (clk) begin
                if (clk'event and clk = '1') then
                        Edo_Pres <= Edo_Fut;
                end if;
        end process proceso2;
end ARQ_MEALY;
```

Diseño de Circuitos Secuenciales Síncronos

Cinvestav

Ejemplo Nº 13



¿Qué tipo de Máquina es?

¿Es correcta su Descripción en VHDL?

¿Cumple las Reglas de Verificación de un Diagrama de Estados?

Diseño de Circuitos Secuenciales Síncronos

Cinvestav

Ejemplo Nº 13 - Máquina Mixta

```
library ieee;
use ieee.std_logic_1164.all;
entity diag is
        port (clk, a: in std_logic;
                sal: out std_logic);
end diag;
architecture arq_diag of diag is
type estados is (q0, q1, q2, q3, q4);
signal edo_pres, edo_fut: estados;
begin
        proceso1: process (edo_pres, a) begin
        case edo_pres is
                when q0 => sal <= '0';
                if a = '0' then
                        edo_fut <= q4;
                else
                        edo_fut <= q1;
                end if;
                when q1 => sal <= '0';
                if a = '0' then
                        edo_fut <= q4;
                else
                        edo_fut <= q2;
                end if;
                when q2=>
                if a = '0' then
                        edo_fut <= q4;
                        sal <= '0';
                else
                        edo_fut <= q3;
                        sal <= '1';
                end if;
                when q3 => sal <= '0';
                if a = '0' then
                        edo_fut <= q3;
                else
                        edo_fut <= q3;
                end if;
                when q4 => sal <= '0';
                if a = '0' then
                        edo_fut <= q4;
                else
                        edo_fut <= q1;
                end if;
        end case;
        end process  proceso1;
        proceso2: process (clk) begin
                if (clk'event and clk = '1') then
                        edo_pres <= edo_fut;
                end if;
        end process proceso2;
end arq_diag;
```

Cinvestav

Capítulo 5

Diseño jerárquico en VHDL

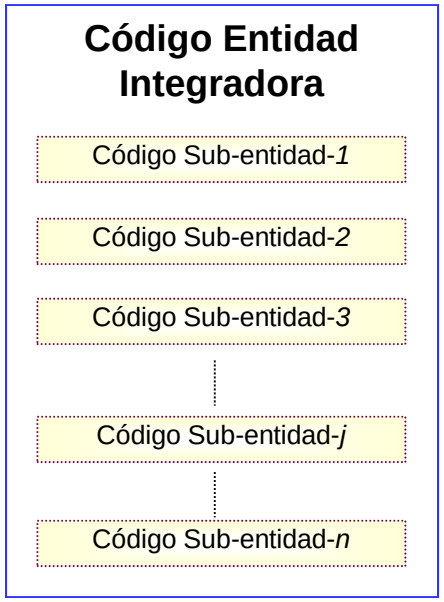
¿Cómo integrar Entidades?

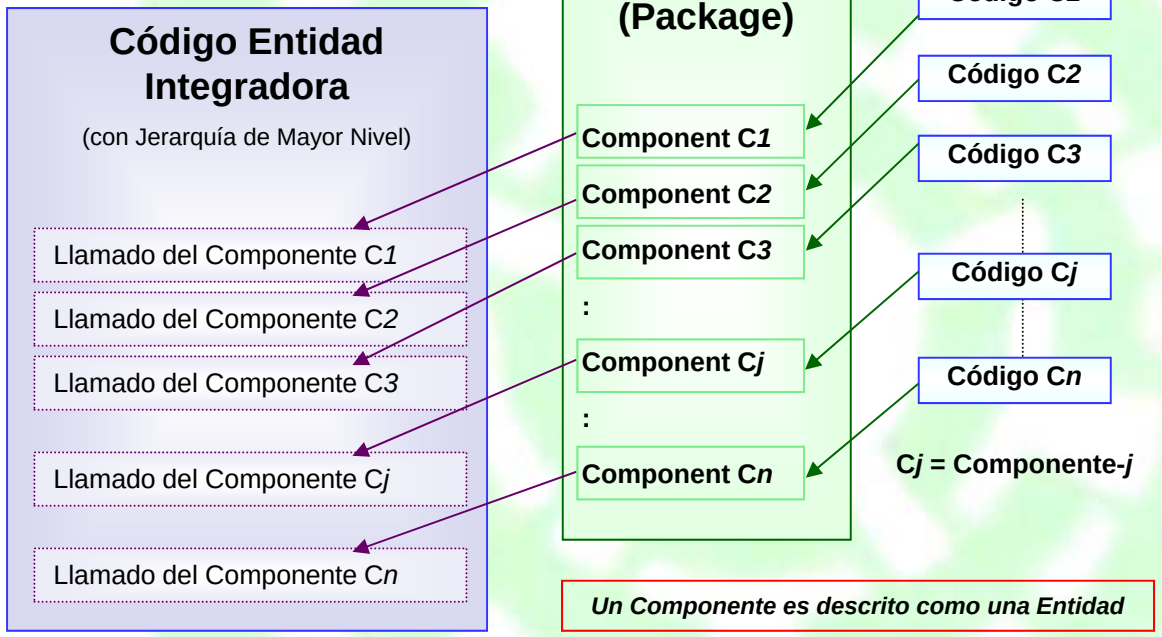
Cinvestav

¿ Cómo integrar dos o más entidades en una sola entidad con el fin de formar un Sistema más complejo?

Integración de Entidades en una sola Entidad

Integración de Estructuras Jerárquicas Uso de Componentes (Components)



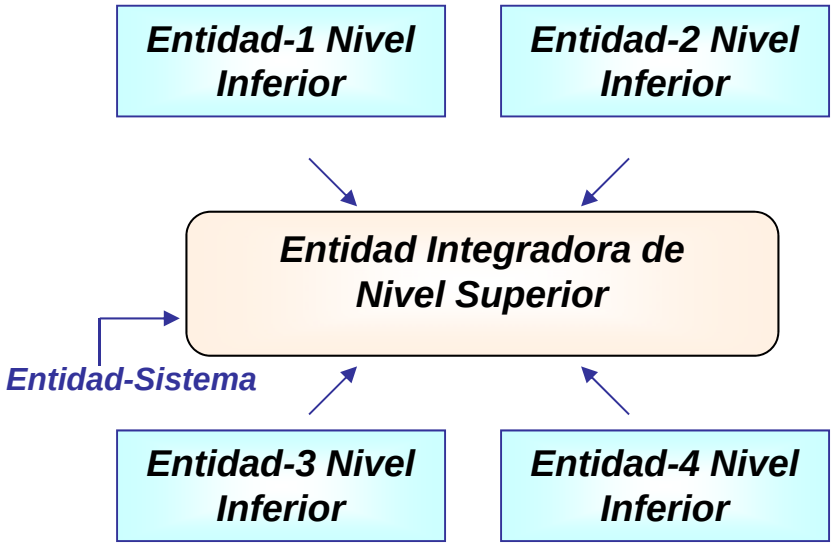


¿Qué es el Diseño Jerárquico?

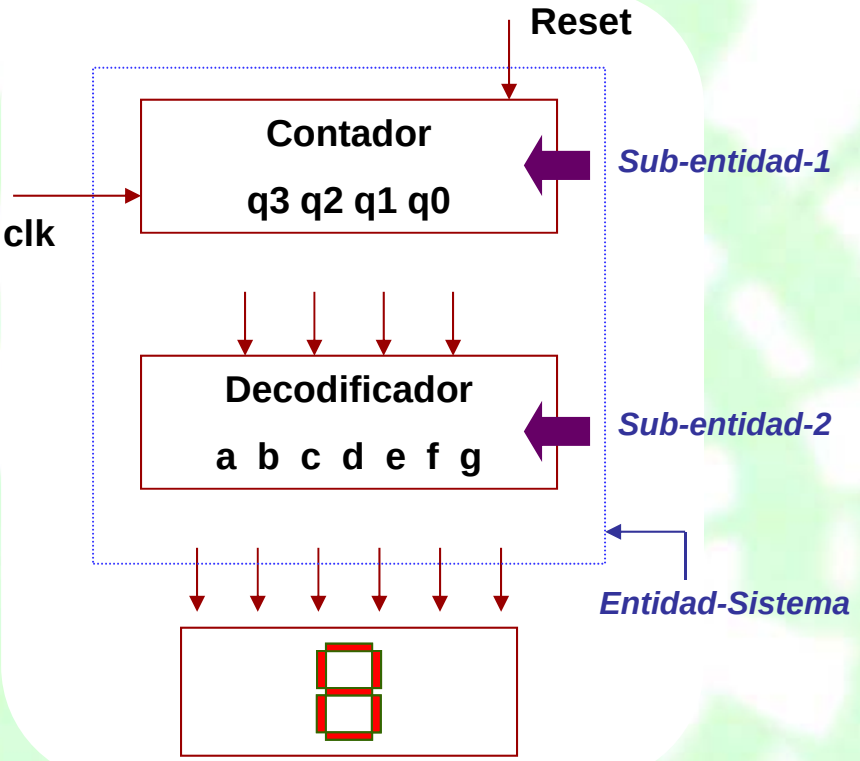
Cinvestav

¿ Cómo diseñar un Sistema complejo mediante la unión de bloques o módulos (entidades) diseñados en forma independiente ?

Estructuras Jerárquicas



Integración de Entidades



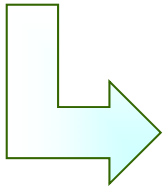
- Una Entidad de Nivel Inferior se puede ver como un Componente independiente.
- En este caso, los códigos asociados a cada entidad o componente de nivel inferior no son combinados dentro del código de la Entidad-Sistema

Metodología de Diseño

Cinvestav

Metodología de Diseño de Estructuras Jerárquicas

- **Analizar el Sistema a diseñar y dividirlo en bloques jerárquicos (Componentes)**
- **Describir, simular y sintetizar los módulos o componentes.**
- **Crear un Paquete de Componentes (Package) – Código VHDL**
- **Describir la Entidad Integradora de Nivel Superior (con Mayor Jerarquía) que representará al Sistema completo – Código VHDL**

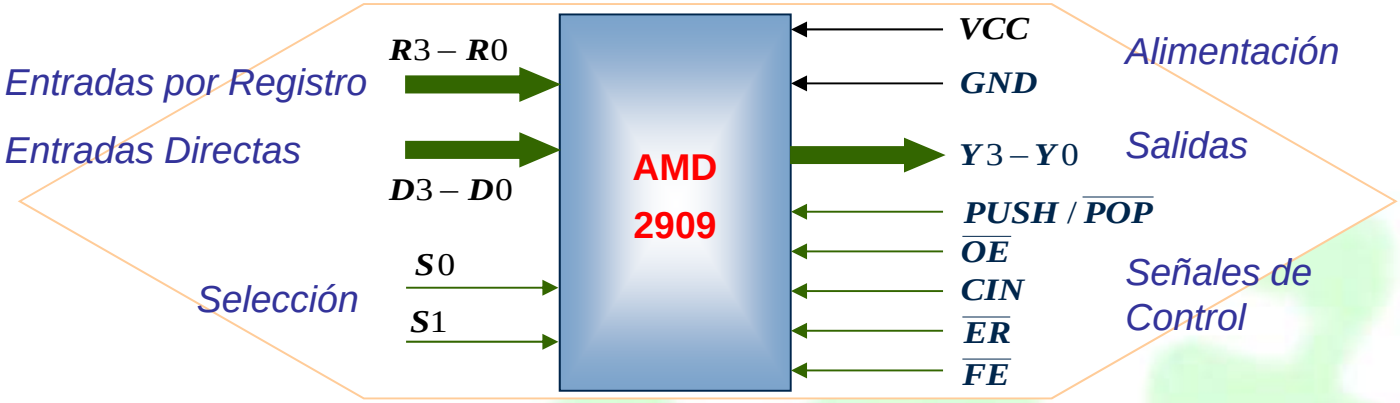


Los puntos con letra azul pueden también ser realizados a través de métodos esquemáticos (p.ej. con WebPack de Xilinx)

Ejemplo 1: Sistema a diseñar

Cinvestav

Secuenciador AMD2909

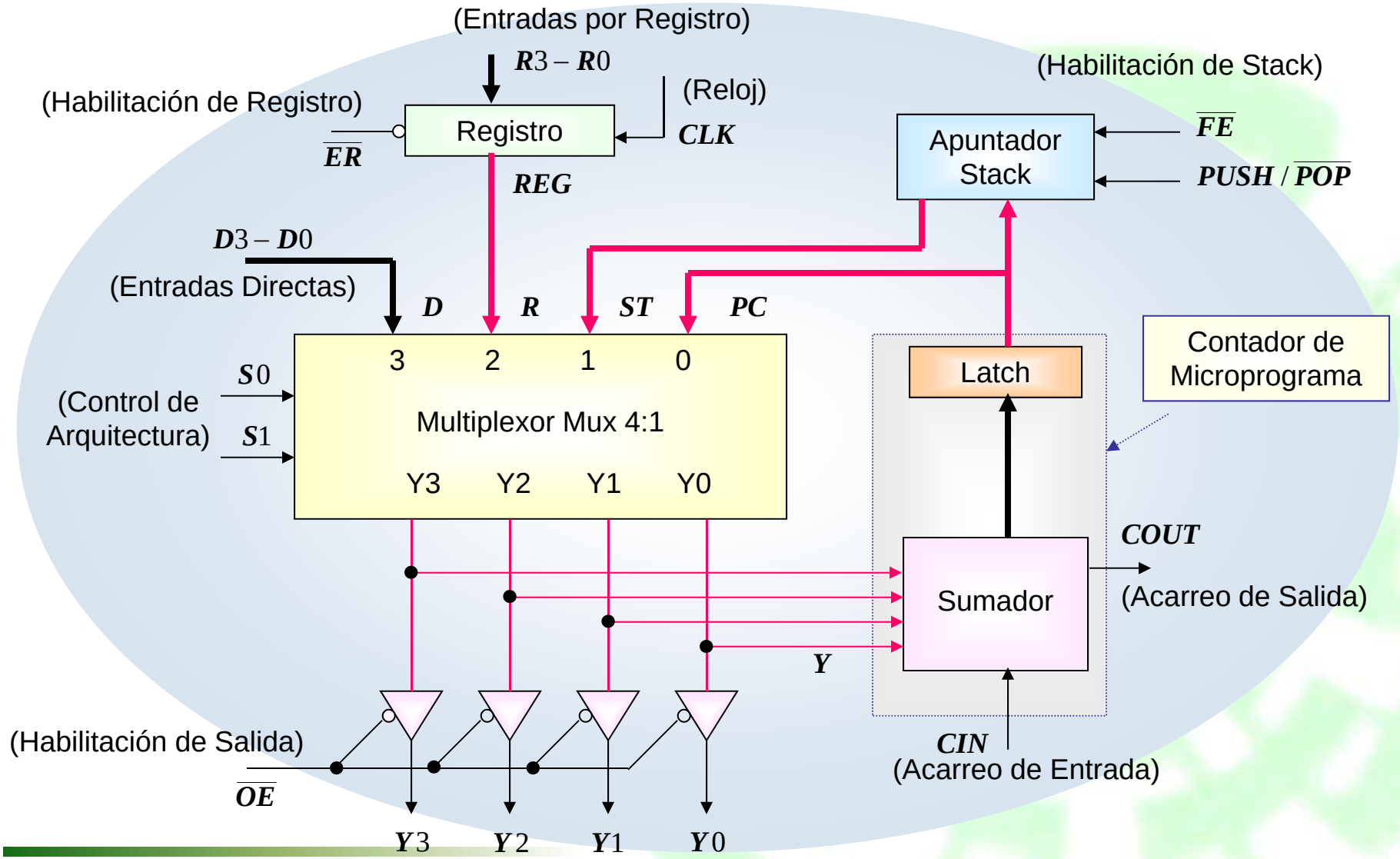


Terminales	Función
Entradas por Registro ***R3-R0***	A través de ellas se permite sostener (hold) una dirección.
Entradas Directas ***D3-D0***	Entradas del Secuenciador que permiten realizar un cambio de dirección en la lógica del programa.
Entrada $\overline{ER}$	Habilitación del Registro R
Entrada $\overline{FE}$	Habilitación del Apuntador de Pila (Stack Pointer: ST)
Entrada ***CIN***	Acarreo de Entrada
Entrada $\overline{OE}$	Habilitación de Salidas
Entrada $PUSH / \overline{POP}$	Control de Direccionamiento de Subrutinas
Entradas ***S0 S1***	Líneas de Selección
Salidas ***Y3-Y0***	Salidas del Secuenciador
Salida ***COUT***	Acarreo de Salida
VCC y **GND**	Alimentación del Circuito

Análisis del Sistema (Definición de Componentes)

Cinvestav

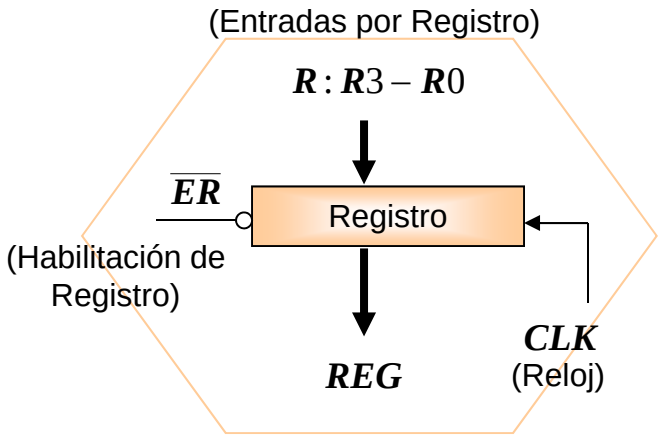
Descripción Interna del Circuito Secuenciador AMD2909



Diseño y programación de componentes

Cinvestav

Diseño del Registro



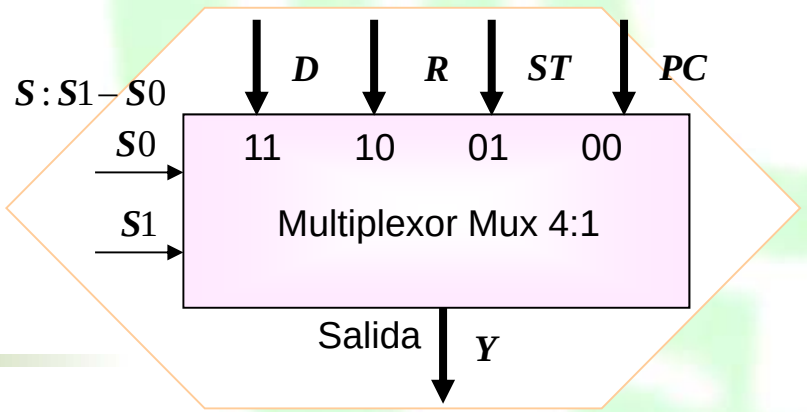

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity registro is
7       Port ( R : in std_logic_vector(3 downto 0);
8              ER, CLK : in std_logic;
9              REG : inout std_logic_vector(3 downto 0));
10  end registro;
11
12  architecture arq_reg of registro is
13
14  begin
15     process (CLK,ER,REG,R) begin
16        if (CLK'event and CLK='1') then
17           if ER='0' then
18              REG <= R;
19           else
20              REG <= REG;
21           end if;
22        end if;
23     end process;
24  end arq_reg;
```

Diseño y programación de componentes

Cinvestav

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity mux_4 is
7       Port ( D,R,ST,PC : in std_logic_vector(3 downto 0);
8              S : in std_logic_vector(1 downto 0);
9              Y : inout std_logic_vector(3 downto 0));
10  end mux_4;
11
12  architecture arq_mux of mux_4 is
13
14  begin
15     with S select
16        Y <=  PC when "00",
17              ST when "01",
18              R  when "10",
19              D  when others;
20  end arq_mux;
```

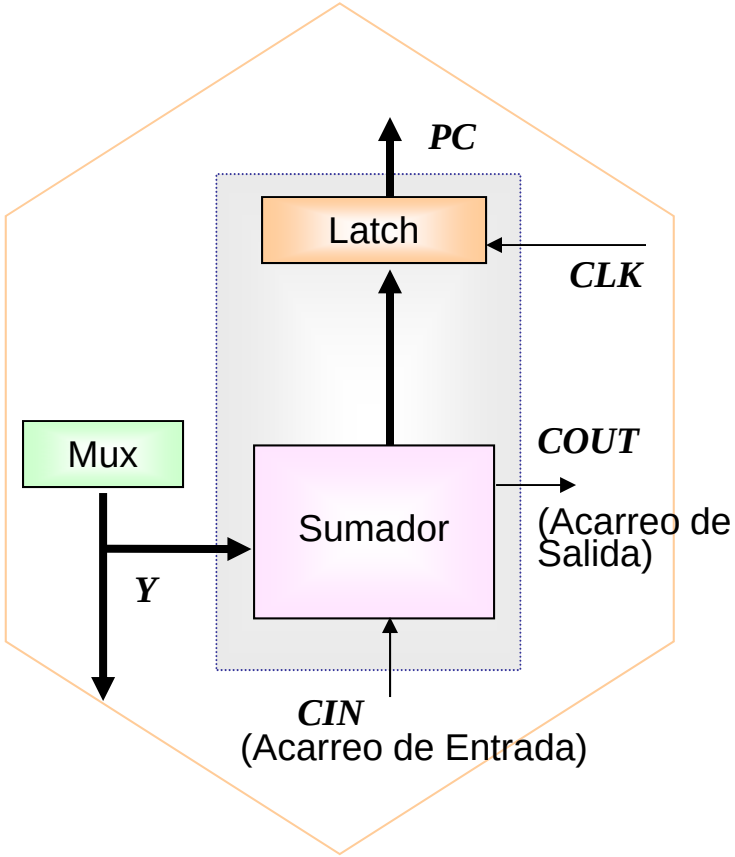
Diseño del Multiplexor 4 a 1



Diseño y programación de componentes

Cinvestav

Diseño del Contador de Microprograma

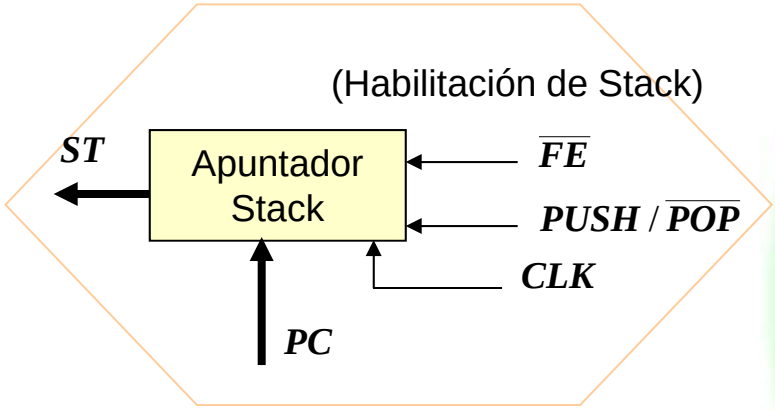



```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity mpc is
7      Port ( CIN,CLK : in std_logic;
8             Y : in std_logic_vector(3 downto 0);
9             COUT : out std_logic;
10            PC : inout std_logic_vector(3 downto 0));
11 end mpc;
12
13 architecture arq_mpc of mpc is
14
15 begin
16    process (CLK,Y,CIN) begin
17       if (CLK'event and CLK='1') then
18          if (CIN='1') then
19             PC <= Y+1;
20          else
21             PC <= Y;
22          end if;
23       end if;
24    end process;
25    COUT <= (CIN and Y(0) and Y(1) and Y(2) and Y(3));
26 end arq_mpc;
```

Diseño y programación de componentes

Cinvestav

Diseño de la Pila (Stack)



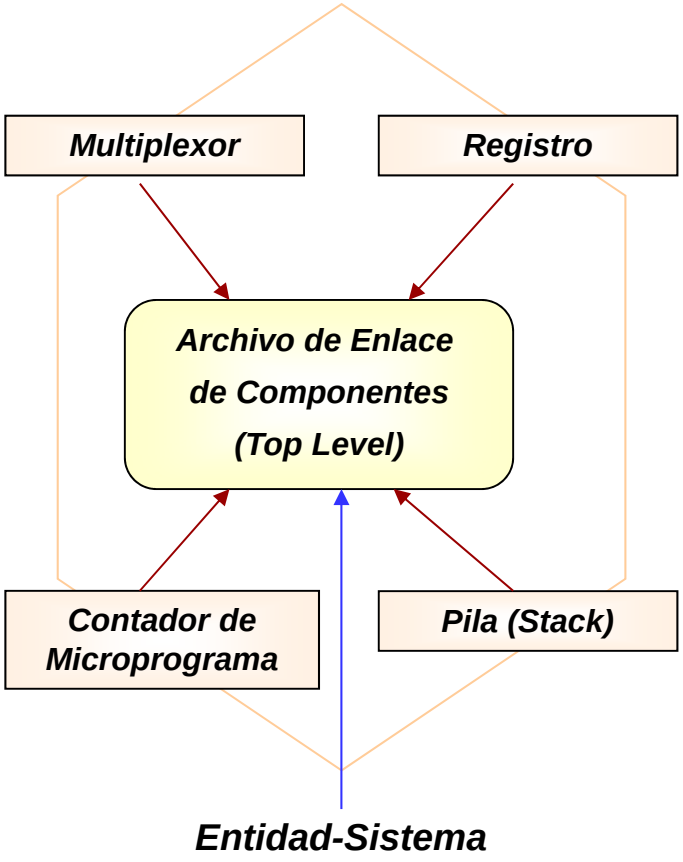

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity stack is
7      Port ( CLK,FE,PUSH_POP : in std_logic;
8             PC : in std_logic_vector(3 downto 0);
9             ST : inout std_logic_vector(3 downto 0));
10 end stack;
11
12 architecture arq_stack of stack is
13 signal var: std_logic_vector(3 downto 0) := "0000";
14 begin
15    process (FE,CLK,PUSH_POP,PC,ST)
16       variable x: std_logic_vector (3 downto 0):= "0000";
17    begin
18     if (CLK'event and CLK='1') then
19       if (FE='0') then
20          if (PUSH_POP='1') then
21             x := PC; -- almacena dato
22             var <= x;
23          else
24             ST <= var; -- extrae dato
25          end if;
26       else
27          ST <= ST;
28       end if;
29    end if;
30    end process;
31 end arq_stack;
```

Creación del Paquete de Componentes

Cinvestav

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  package comp_sec is
5
6  component registro port(
7        R: in std_logic_vector (3 downto 0);
8        ER,CLK: in std_logic;
9        REG: inout std_logic_vector (3 downto 0));
10 end component;
11
12 component mpc port(
13       CIN,CLK: in std_logic;
14       Y: in std_logic_vector(3 downto 0);
15       COUT: out std_logic;
16       PC: inout std_logic_vector (3 downto 0));
17 end component;
18
19 component stack port(
20       CLK,FE,PUSH_POP: in std_logic;
21       PC: inout std_logic_vector (3 downto 0);
22       ST: inout std_logic_vector (3 downto 0));
23 end component;
24
25 component mux_4 port(
26       D,R,ST,PC: in std_logic_vector (3 downto 0);
27       S: in std_logic_vector (1 downto 0);
28       Y: inout std_logic_vector (3 downto 0));
29 end component;
30
31 end comp_sec;
```

Realización del Programa de Alto Nivel (Top Level)



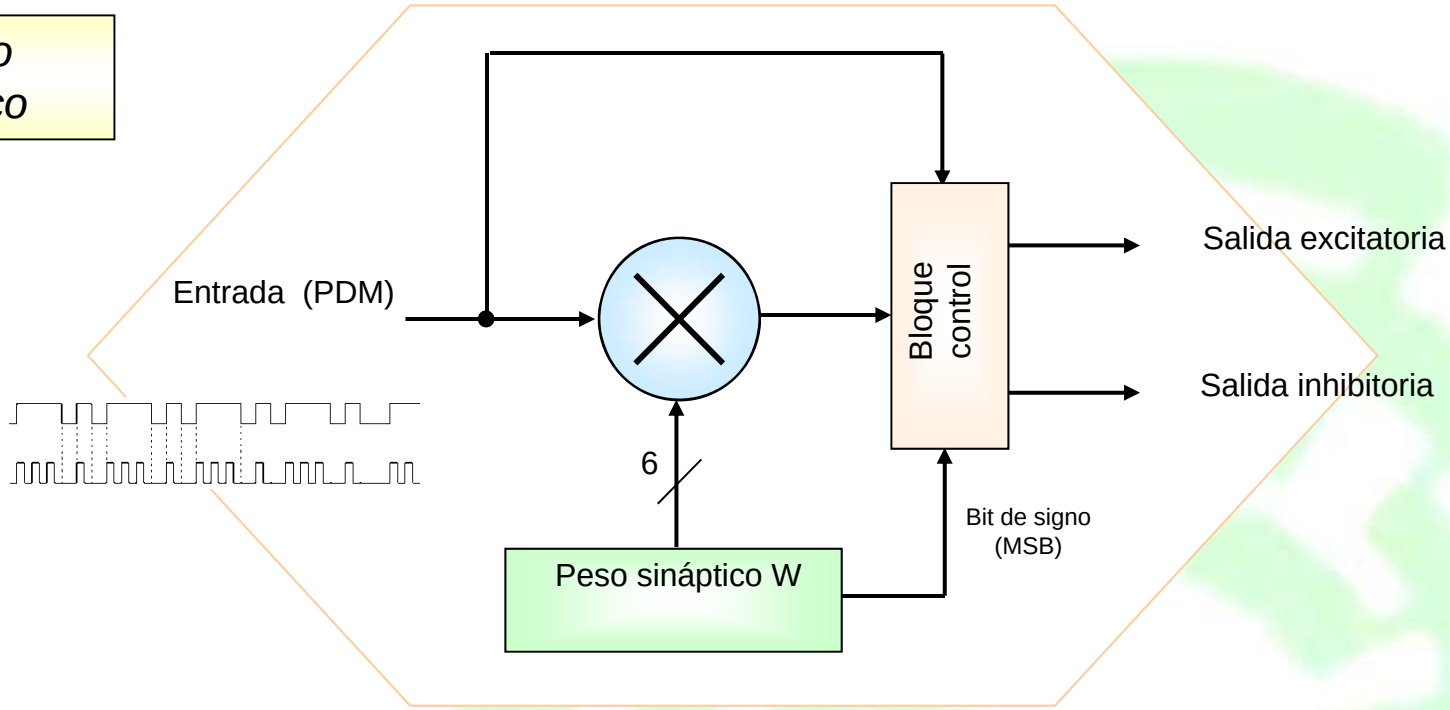
Verificar y Simular el Sistema

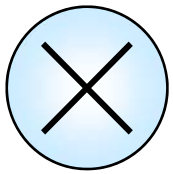
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5  use work.comp_sec.ALL;
6
7  entity amd2909 is
8      Port ( R : in std_logic_vector(3 downto 0);
9             D : in std_logic_vector(3 downto 0);
10            ER : in std_logic;
11            CLK : in std_logic;
12            S : in std_logic_vector (1 downto 0);
13            FE : in std_logic;
14            PUSH_POP : in std_logic;
15            CIN : in std_logic;
16            COUT : out std_logic;
17            Y : inout std_logic_vector (3 downto 0));
18 end amd2909;
19
20 architecture arq_amd2909 of amd2909 is
21
22    signal REG: std_logic_vector (3 downto 0);
23    signal ST: std_logic_vector (3 downto 0);
24    signal PC: std_logic_vector (3 downto 0);
25
26 begin
27
28 -- Inicia interconexión de componentes
29
30 u1: registro port map (CLK=>CLK, ER=>ER, REG=>REG, R=>R);
31 u2: mpc port map (CIN=>CIN, COUT=>COUT, CLK=>CLK, Y=>Y, PC=>PC);
32 u3: stack port map (CLK=>CLK, FE=>FE, PUSH_POP=>PUSH_POP, PC=>PC, ST=>ST);
33 u4: mux_4 port map (D=>D, R=>REG, ST=>ST, PC=>PC, S=>S, Y=>Y);
34
35 end arq_amd2909;
```

Ejemplo 2: Sistema a diseñar

Cinvestav

Circuito Sináptico





Multiplicador de frecuencia

Peso sináptico W

Registro de 7 bits del peso sináptico

Bloque control

Circuito de selección y muestreador

Tabla de funcionamiento del multiplicador de frecuencia

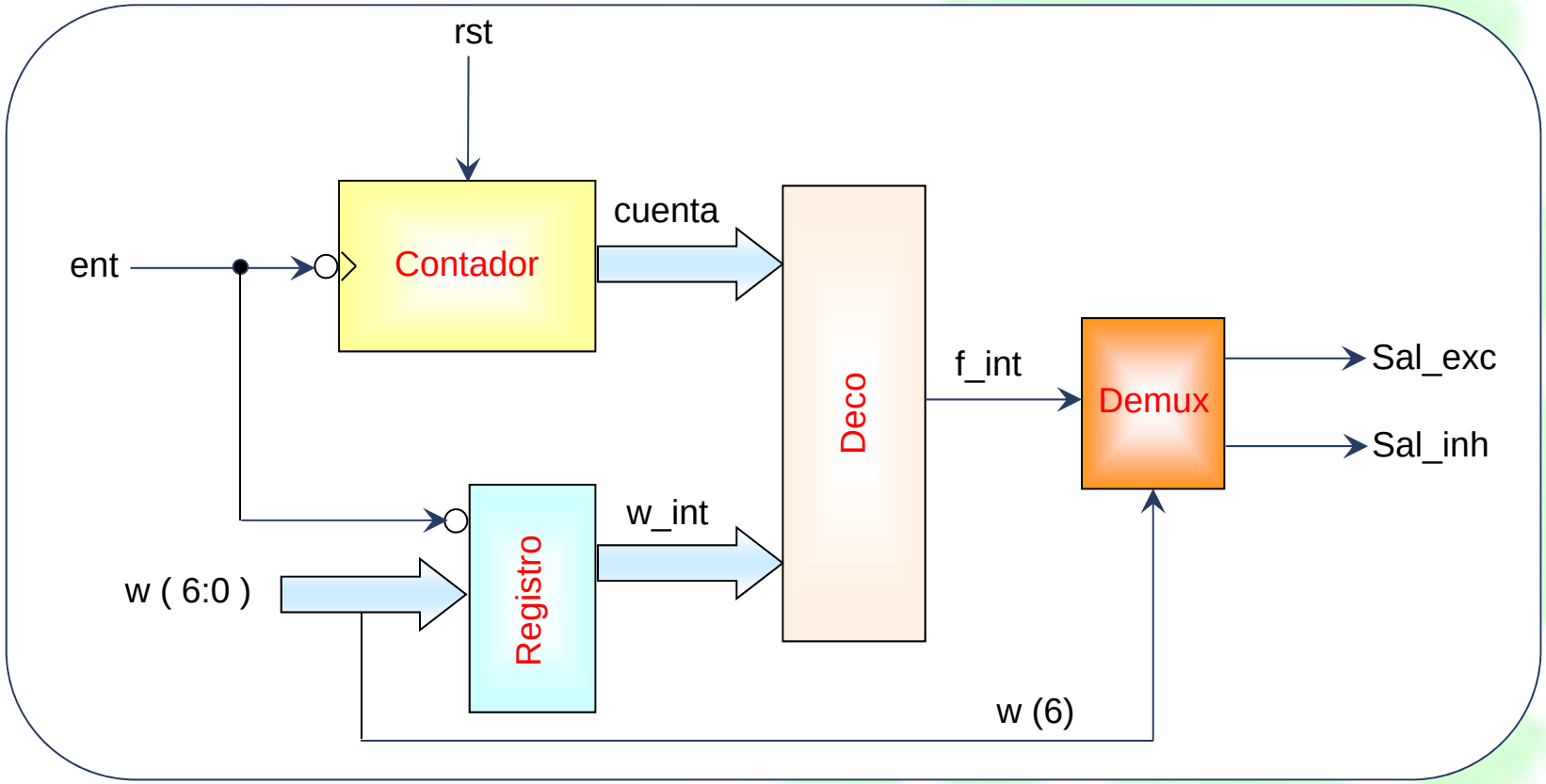
Cinvestav

		Estado							
		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
Peso (w)	000	0	0	0	0	0	0	0	0
	001	0	0	0	0	1	0	0	0
	010	0	0	1	0	0	0	1	0
	011	0	0	1	0	1	0	1	0
	100	0	1	0	1	0	1	0	1
	101	0	1	0	1	1	1	0	1
	110	0	1	1	1	0	1	1	1
	111	0	1	1	1	1	1	1	1

Análisis del Sistema (Definición de Componentes)

Cinvestav

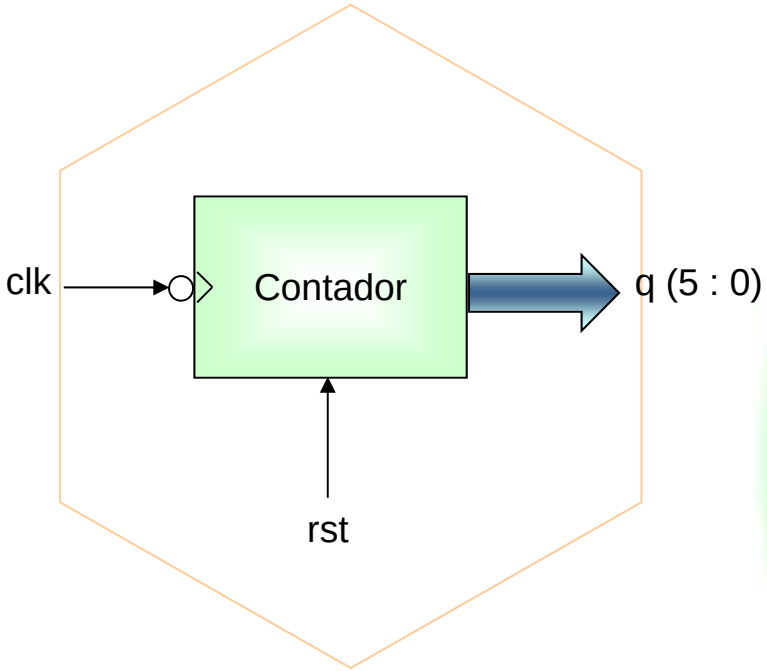
Descripción Interna del circuito Sináptico



Diseño y programación de componentes

Cinvestav

Diseño del contador



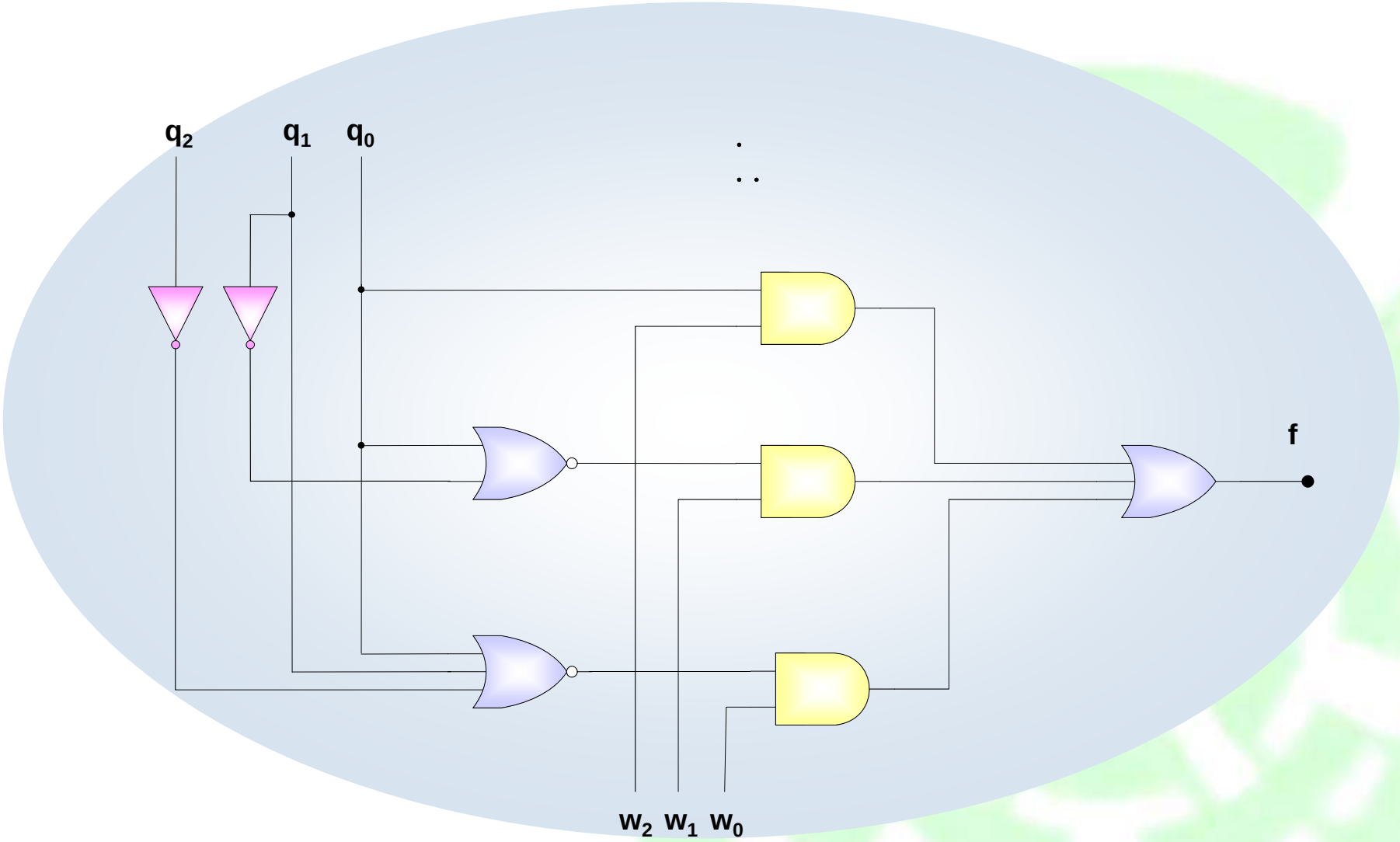

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity contador is
7      Port ( clk : in std_logic;
8             rst : in std_logic;
9             q :   inout std_logic_vector(5 downto 0));
10 end contador;
11
12 architecture arq_contador of contador is
13
14 begin
15
16    process(clk,rst)
17    begin
18       if rst = '1' then
19          q <= "000000";
20       elsif (clk'event and clk='0') then
21          q <= q + 1;
22       end if;
23    end process;
24
25 end arq_contador;
```

Encontrar la relación de distribución (solucion 1)

Cinvestav

AND		Estado							
		0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
Peso (w)	0 0 0	0	0	0	0	0	0	0	0
	1 0 0	0	0	0	0	1	0	0	0
	0 1 0	0	0	1	0	0	0	1	0
	1 1 0	0	0	1	0	1	0	1	0
	0 0 1	0	1	0	1	0	1	0	1
	1 0 1	0	1	0	1	1	1	0	1
	0 1 1	0	1	1	1	0	1	1	1
	1 1 1	0	1	1	1	1	1	1	1

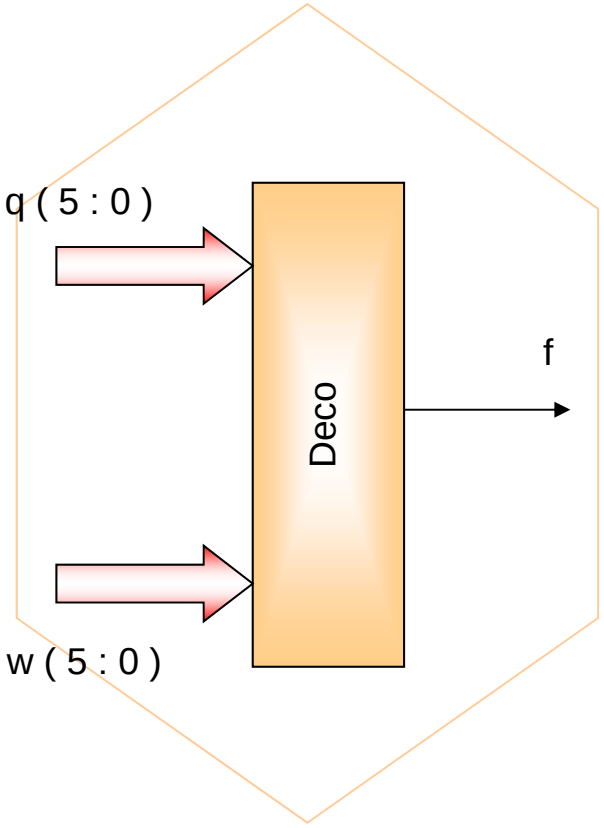
Circuito Lógico resultante



Diseño y programación de componentes

Cinvestav

Diseño del decodificador

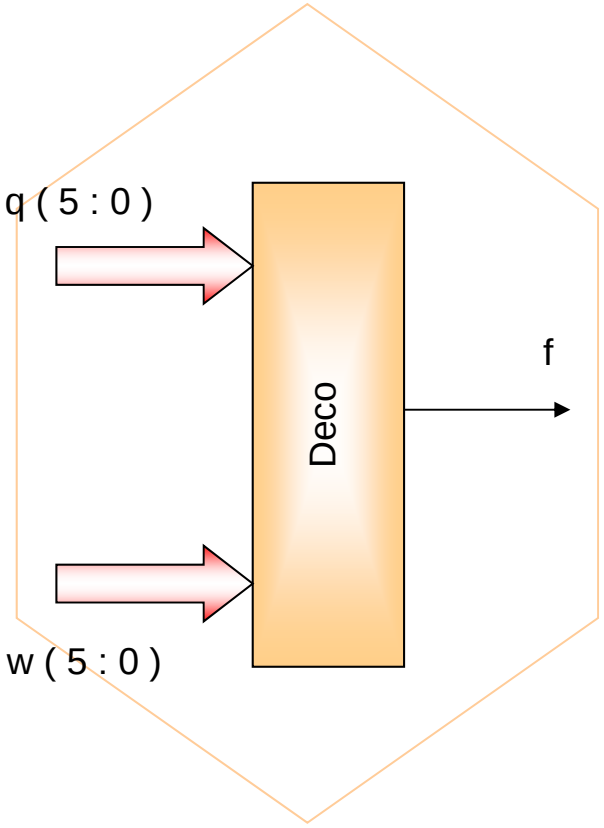



```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.STD_LOGIC_ARITH.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  entity deco is
 7      Port ( q : in std_logic_vector(5 downto 0);
 8             w : in std_logic_vector(5 downto 0);
 9             f : out std_logic);
10  end deco ;
11
12  architecture arq_deco of deco is
13  signal r: std_logic_vector(5 downto 0);
14
15  begin
16  r(0) <=  q(0);
17  r(1) <= not(not q(1) or q(0));
18  r(2) <= not(not q(2) or q(1) or q(0));
19
20
21
22
23  f <= (r(0) and w(5)) or
24       (r(1) and w(4)) or
25       (r(2) and w(3)) or
26
27
28
29
30  end arq_deco ;
```

Diseño y programación de componentes

Cinvestav

Diseño del decodificador

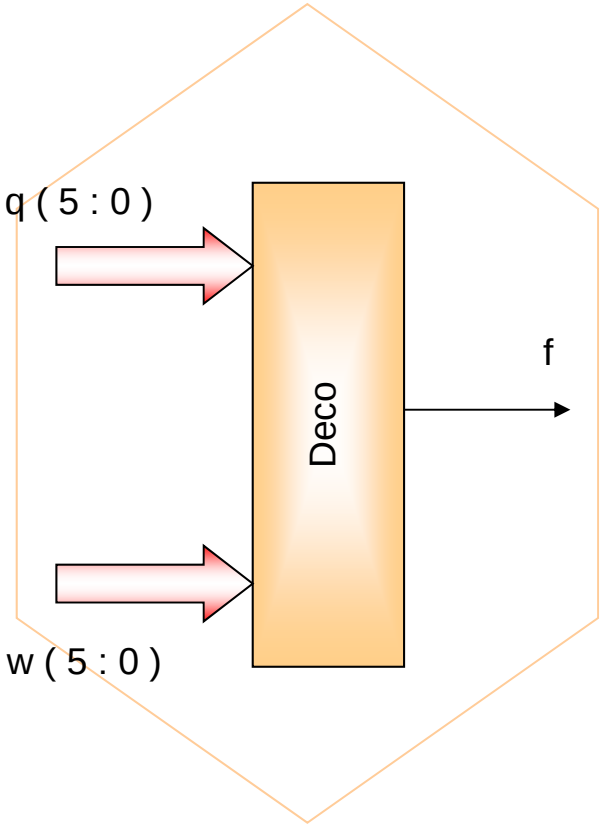



```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.STD_LOGIC_ARITH.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  entity deco is
 7      Port ( q : in std_logic_vector(5 downto 0);
 8             w : in std_logic_vector(5 downto 0);
 9             f : out std_logic);
10  end deco ;
11
12  architecture arq_deco of deco is
13  signal r: std_logic_vector(5 downto 0);
14
15  begin
16  r(0) <=  q(0);
17  r(1) <= not(not q(1) or q(0));
18  r(2) <= not(not q(2) or q(1) or q(0));
19  r(3) <= not(not q(3) or q(2) or q(1) or q(0));
20
21
22
23  f <= (r(0) and w(5)) or
24       (r(1) and w(4)) or
25       (r(2) and w(3)) or
26       (r(3) and w(2)) or
27
28
29
30  end arq_deco ;
```

Diseño y programación de componentes

Cinvestav

Diseño del decodificador

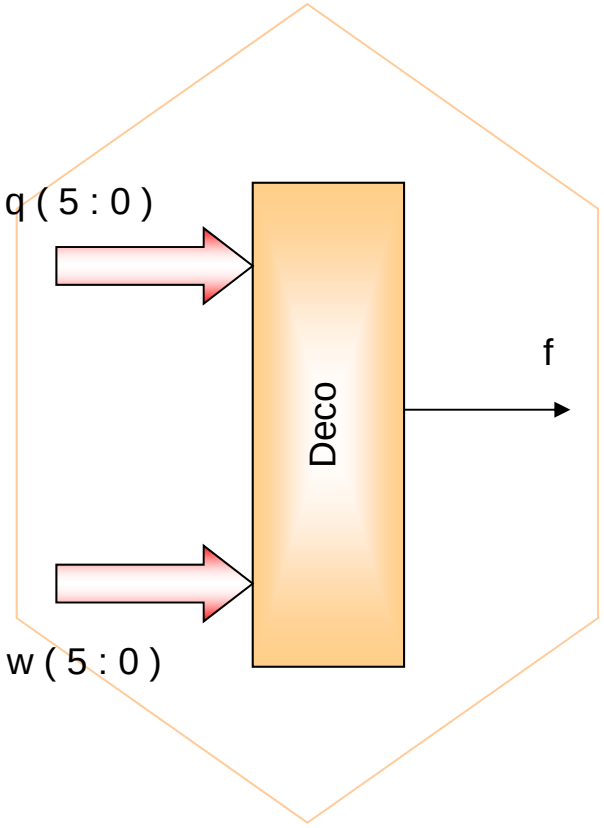



```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.STD_LOGIC_ARITH.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  entity deco is
 7      Port ( q : in std_logic_vector(5 downto 0);
 8             w : in std_logic_vector(5 downto 0);
 9             f : out std_logic);
10  end deco ;
11
12  architecture arq_deco of deco is
13  signal r: std_logic_vector(5 downto 0);
14
15  begin
16  r(0) <=  q(0);
17  r(1) <= not(not q(1) or q(0));
18  r(2) <= not(not q(2) or q(1) or q(0));
19  r(3) <= not(not q(3) or q(2) or q(1) or q(0));
20  r(4) <= not(not q(4) or q(3) or q(2) or q(1) or q(0));
21
22
23  f <= (r(0) and w(5)) or
24       (r(1) and w(4)) or
25       (r(2) and w(3)) or
26       (r(3) and w(2)) or
27       (r(4) and w(1)) or
28
29
30  end arq_deco ;
```

Diseño y programación de componentes

Cinvestav

Diseño del decodificador




```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.STD_LOGIC_ARITH.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  entity deco is
 7      Port ( q : in std_logic_vector(5 downto 0);
 8             w : in std_logic_vector(5 downto 0);
 9             f : out std_logic);
10  end deco ;
11
12  architecture arq_deco of deco is
13  signal r: std_logic_vector(5 downto 0);
14
15  begin
16  r(0) <=  q(0);
17  r(1) <= not(not q(1) or q(0));
18  r(2) <= not(not q(2) or q(1) or q(0));
19  r(3) <= not(not q(3) or q(2) or q(1) or q(0));
20  r(4) <= not(not q(4) or q(3) or q(2) or q(1) or q(0));
21  r(5) <= not(not q(5) or q(4) or q(3) or q(2) or q(1) or q(0));
22
23  f <= (r(0) and w(5)) or
24       (r(1) and w(4)) or
25       (r(2) and w(3)) or
26       (r(3) and w(2)) or
27       (r(4) and w(1)) or
28       (r(5) and w(0));
29
30  end arq_deco ;
```

Encontrar la relación de distribución (solución 2)

Cinvestav

Peso W			Estados del contador (Q_2 Q_1 Q_0)							
w_2	w_1	w_0	000	001	010	011	100	101	110	111
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0
0	1	0	0	0	1	0	0	0	1	0
0	1	1	0	0	1	0	1	0	1	0
1	0	0	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	1	1	0	1
1	1	0	0	1	1	1	0	1	1	1
1	1	1	0	1	1	1	1	1	1	1

Q_2 Q_1 Q_0	f
X X 1	w_2
X 1 0	w_1
1 0 0	w_0
0 0 0	0

$$f = Q_0 w_2 + Q_1 \overline{Q_0} w_1 + Q_2 \overline{Q_1} \overline{Q_0} w_0$$

Solución 2 : Encontrar la relación de distribución

Cinvestav

Generalización a N bits

$Q_3\ Q_2\ Q_1\ Q_0$	f
X X X 1	w_3
X X 1 0	w_2
X 1 0 0	w_1
1 0 0 0	w_0
0 0 0 0	0

$Q_4\ Q_3\ Q_2\ Q_1\ Q_0$	f
X X X X 1	w_4
X X X 1 0	w_3
X X 1 0 0	w_2
X 1 0 0 0	w_1
1 0 0 0 0	w_0
0 0 0 0 0	0

$Q_5\ Q_4\ Q_3\ Q_2\ Q_1\ Q_0$	f
X X X X X 1	w_5
X X X X 1 0	w_4
X X X 1 0 0	w_3
X X 1 0 0 0	w_2
X 1 0 0 0 0	w_1
1 0 0 0 0 0	w_0
0 0 0 0 0 0	0

$$f = Q_0 w_3 + Q_1 \overline{Q_0} w_2 + Q_2 \overline{Q_1}\,\overline{Q_0} w_1 + Q_3 \overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} w_0$$

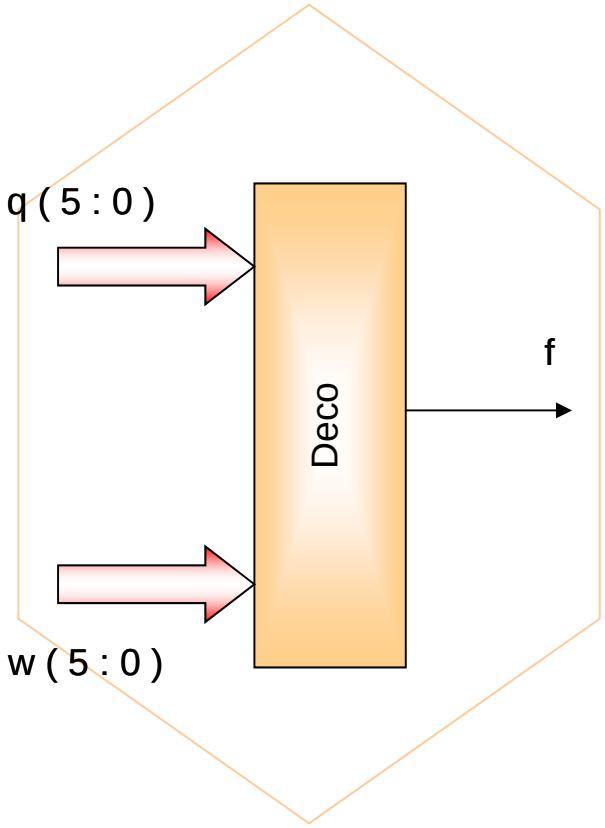
$$f = Q_0 w_4 + Q_1 \overline{Q_0} w_3 + Q_2 \overline{Q_1}\,\overline{Q_0} w_2 + Q_3 \overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} w_1 + Q_4 \overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} w_0$$

$$f = Q_0 w_5 + Q_1 \overline{Q_0} w_4 + Q_2 \overline{Q_1}\,\overline{Q_0} w_3 + Q_3 \overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} w_2 + Q_4 \overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} w_1 + Q_5 \overline{Q_4}\,\overline{Q_3}\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0} w_0$$

Diseño y programación de componentes

Cinvestav

Diseño del decodificador




```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.STD_LOGIC_ARITH.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6  entity deco is
 7      Port ( q : in std_logic_vector(5 downto 0);
 8             w : in std_logic_vector(5 downto 0);
 9             f : out std_logic);
10  end deco ;
11
12  architecture arq_deco of deco is
13
14  begin
15
16  with q select
17
18  f <=   w(5) when "-----1",
19         w(4) when "----10",
20         w(3) when "---100",
21         w(2) when "--1000",
22         w(1) when "-10000",
23         w(0) when "100000",
24         '0'  when others;
25
26  end arq_deco ;
```

Creación del Paquete de Componentes

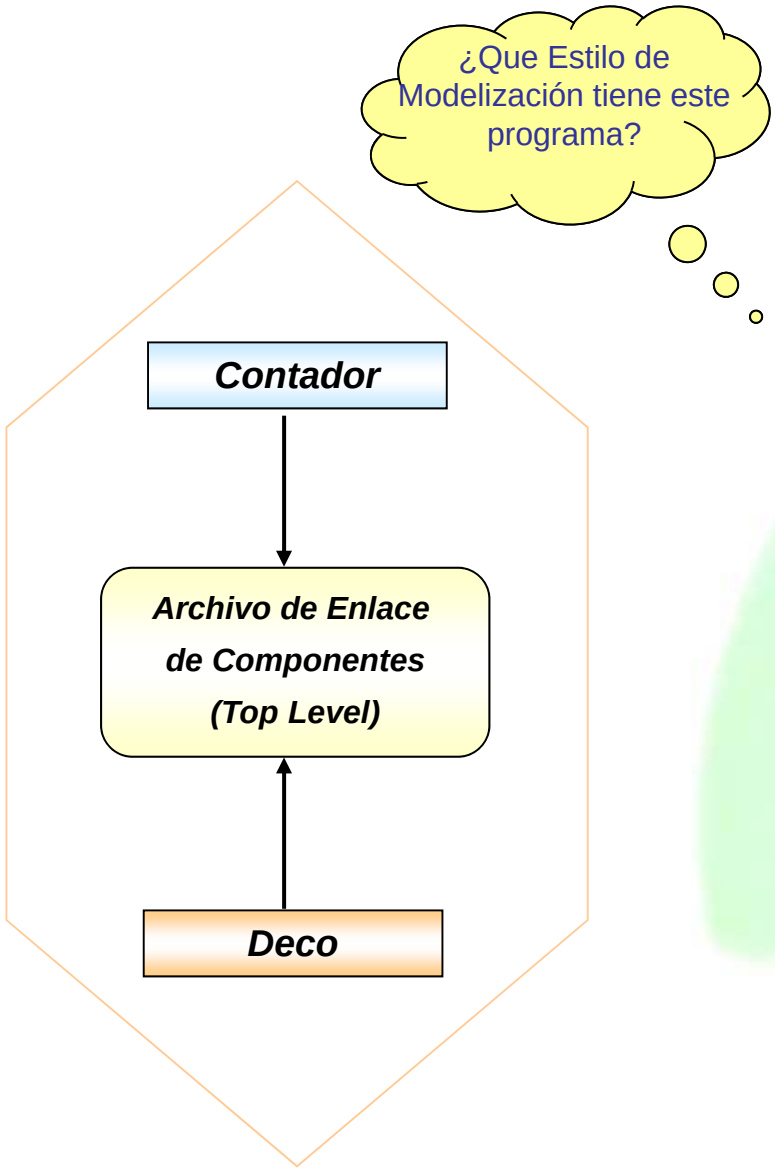
Cinvestav

Creación del Paquete de Componentes

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.all;
3
4   package comp_mult is
5
6   component deco
7      Port (  q : in std_logic_vector(5 downto 0);
8              w : in std_logic_vector(5 downto 0);
9              f : out std_logic);
10  end component;
11
12  component contador
13      Port ( clk : in std_logic;
14             rst : in std_logic;
15             q :   inout std_logic_vector(5 downto 0));
16  end component;
17
18  end comp_mult;
```

Realización del Programa de Alto Nivel (Top Level)

Cinvestav




```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.STD_LOGIC_ARITH.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5  use work.comp_mult.ALL;
 6
 7  entity sinap is
 8      Port ( ent,rst : in std_logic;
 9             w : in std_logic_vector(6 downto 0);
10             sal_exc,sal_inh : out std_logic );
11  end sinap;
12
13  architecture arq_sinap of sinap is
14
15  signal w_int, cuenta : std_logic_vector(5 downto 0);
16  signal f_int : std_logic;
17
18  begin
19  u1: contador port map(clk=>ent, rst=>rst, q=>cuenta);
20  u2: deco     port map(q=>cuenta, w=>w_int, f=>f_int);
21
22  sal_exc <= f_int  and not w(6);
23  sal_inh <= f_int  and w(6);
24
25  process(ent, w)
26     begin
27        if ent='0' then
28           w_int <= w(5 downto 0);
29        end if;
30  end process;
31  end arq_sinap;
```

Estructuras de repetición

Cinvestav

GENERATE crea cero o más copias de un conjunto cerrado de instrucciones concurrentes. Existen dos clases de generate.

for … generate: el número de copias está determinado por un rango discreto.

SINTAXIS

label: **for** *identifier* **in** *range* **generate**
{*concurrent_statment***}**
end generate[*label***];**

DESCRIPCION

label: **nombre de esta instrucción la cual sirve para construir instrucciones *generate* anidadas.**

identifier: **es específico de la instrucción *for…generate*.**

range: **número entero calculable de la cantidad de copias; por ejemplo:**
integer_expression **to** *integer_expression*, **o bien,**
integer_expression **downto** *integer_expression*

Ejemplo de la instrucción for … generate

Cinvestav

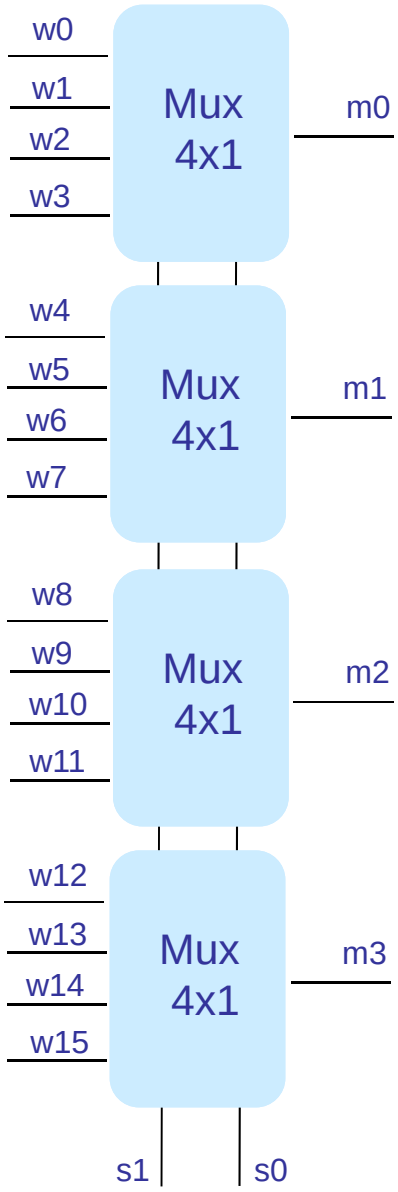
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity mux4tol is
7      Port ( w0,w1,w2,w3: in std_logic;
8             s: in std_logic_vector (1 downto 0);
9             f : out std_logic);
10 end mux4tol;
11
12 architecture arq_mux of mux4tol is
13
14 begin
15    with s select
16    f<= w0 when "00",
17        w1 when "01",
18        w2 when "10",
19        w3 when others;
20 end arq_mux;
```

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  Package mux4tol_package is
5
6  component mux4tol is
7      Port ( w0,w1,w2,w3: in std_logic;
8             s: in std_logic_vector (1 downto 0);
9             f : out std_logic);
10 end component;
11
12 end mux4tol_package;
```

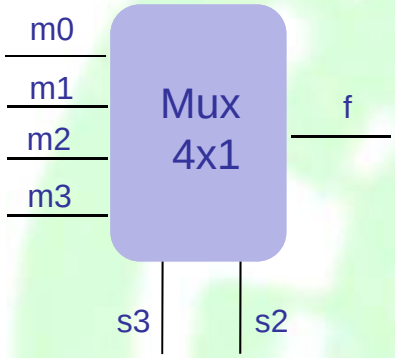
Diseño de un Multiplexor de 16 a 1

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use work.mux4tol_package.all;
4
5  entity mux16tol is
6      Port ( w: in std_logic_vector(0 to 15);
7             s: in std_logic_vector (3 downto 0);
8             f : out std_logic);
9  end mux16tol;
10
11 architecture arq_mux of mux16tol is
12
13 signal m: std_logic_vector (0 to 3);
14
15 begin
16    G1: for i in 0 to 3 generate
17       Muxes: mux4tol port map(
18       w(4*i),w(4*i+1),w(4*i+2),w(4*i+3),s(1 downto 0),m(i));
19       end generate;
20       mux5:mux4tol port map (m(0),m(1),m(2),m(3),s(3 downto 2), f);
21
22 end arq_mux;
```

Multiplexor de 16 a 1




```
G1: for  i  in 0 to 3 generate
      Muxes: mux4to1 port  map (
      w(4*i), w(4*i+1), w(4*i+2), w(4*i+3), s(1 downto 0), m(i));
      end generate;
      Mux5: mux4to1 port  map (m(0), m(1), m(2), m(3), s(3 downto 2), f);
```



s(3)	s(2)	s(1)	s(0)	w
0	0	0	0	w0
0	0	0	1	w1
0	0	1	0	w2
0	0	1	1	w3
0	1	0	0	w4
0	1	0	1	w5
0	1	1	0	w6
0	1	1	1	w7
1	0	0	0	w8
1	0	0	1	w9
1	0	1	0	w10
1	0	1	1	w11
1	1	0	0	w12
1	1	0	1	w13
1	1	1	0	w14
1	1	1	1	w15

Estructuras de repetición

Cinvestav

if … generate: realiza cero o una copia de manera condicional. Sirve para generar una estructura regular que tiene un principio distinto, el cuerpo medio constante y un final también diferente.

SINTAXIS

label: **if** *expression* **generate**
{*concurrent_statment***}**
end generate[*label***];**

DESCRIPCION

label: **nombre de esta instrucción.**
expression: **cualquier expresión que evalue un valor del tipo *Boolean*.**
concurrent_statment: **cunjunto de instrucciones concurrentes.**

Ejemplo de la instrucción if … generate

Cinvestav

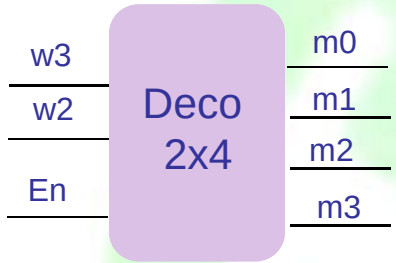
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity dec2to4 is
7      Port ( w: in std_logic_vector(1 downto 0);
8             En: in std_logic;
9             y : out std_logic_vector(0 to 3));
10 end dec2to4;
11
12 architecture arq_deco of dec2to4 is
13
14 signal Enw : std_logic_vector(2 downto 0);
15
16 begin
17    Enw<= En & w;
18
19     with Enw select
20     y <= "1000" when "100",
21          "0100" when "101",
22          "0010" when "110",
23          "0001" when "111",
24          "0000" when others;
25
26 end arq_deco;
```

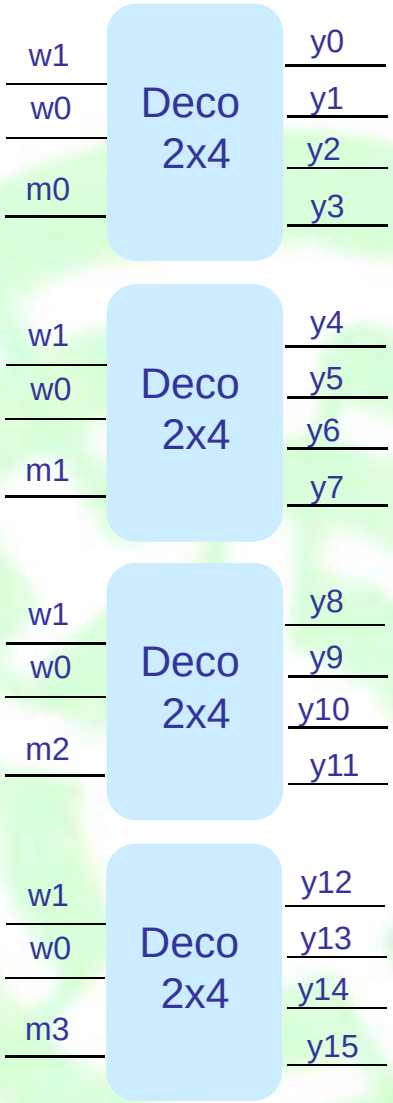
Diseño de un Decodificador de 4 a 16

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity dec4to16 is
7      Port ( w: in std_logic_vector(3 downto 0);
8             En: in std_logic;
9             y : out std_logic_vector(0 to 15));
10 end dec4to16;
11
12 architecture arq_deco of dec4to16 is
13
14 component dec2to4 is
15     Port ( w: in std_logic_vector(1 downto 0);
16            En: in std_logic;
17            y : out std_logic_vector(0 to 3 ));
18 end component;
19
20 signal m : std_logic_vector(0 to 3);
21
22 begin
23    G1: for i in 0 to 3 generate
24        dec_ri: dec2to4 port map(w(1 downto 0),m(i),y(4*i to 4*i+3));
25        G2: if i=3 generate
26        dec_left: dec2to4 port map (w(i downto i-1),En,m);
27        end generate;
28    end generate;
29
30 end arq_deco;
```

Decodificador de 4 a 16

```
G1: for  i  in 0 to 3 generate
      dec_ri: dec2to4 port map (w(1 downto 0), m(i), y(4*i to 4*i+3));
      G2: if i=3 generate
           dec_left: dec2to4 port map (w(i downto i-1), En, m);
       end generate;
end generate;
```





PROCEDIMIENTO

Cinvestav

PROCEDURE

El procedimiento sólo puede devolver valores a través de los parámetros que se le pasen, los argumentos pueden ser de entrada, de salida o bidireccional, tiene efectos colaterales, es decir, puede provocar cambios en objetos externos a él, puede tener instrucciones WAIT.

SINTAXIS

PROCEDURE nombre [(parámetros)] **IS**

[declaraciones]

BEGIN

[sentencias_serie]

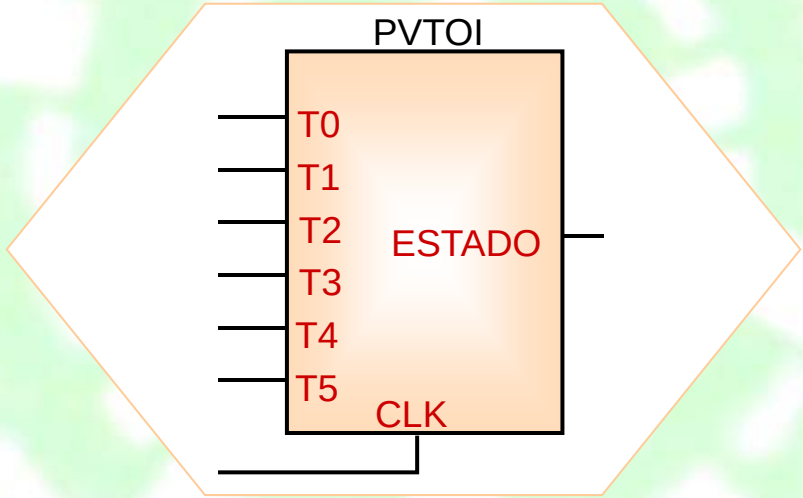
END [PROCEDURE] [nombre];

Ejemplo de un PROCEDIMIENTO

Cinvestav

```
-- convertir un vector a entero (versión procedimiento)
library IEEE;
use ieee.std_logic_1164.all;

entity PVTOI is
port (CLK: in std_logic;
       T0,T1,T2,T3,T4,T5: in std_logic;
       ESTADO: out integer range 0 to 63 );
end PVTOI;

architecture RTL of PVTOI is
signal temporal1: std_logic_vector(5 downto 0);
begin
process(CLK,temporal1)
variable temporal2: integer range 0 to 63;
begin
 if (CLK'event AND CLK='1') then
    temporal1<=T5&T4&T3&T2&T1&T0;
  end if;
  -- se llama al procedimiento VTOI
vtoi(temporal1,6,temporal2);
ESTADO<=temporal2;
end process;
end RTL;
```

```
procedure vtoi(vin: in std_logic_vector;
                    nbits: in integer;
                  salida: out integer) is
 variable temp: integer range 0 to 63;
begin
   temp:=0;
  for i in 0 to nbits-1 loop
  if (vin(i)='1') then
     temp:=temp + (2**i);
  end if;
  end loop;
  salida:=temp;
end;
```



FUNCION

Cinvestav

FUNCTION

Una función devuelve un valor; los argumentos son siempre de entrada; no tiene efectos colaterales; como devuelve un valor se usa en expresiones; debe contener la palabra clave RETURN seguida de una expresión; jamás debe tener la instrucción WAIT.

SINTAXIS

[PURE | IMPURE]

[FUNCTION nombre [(parámetros)] **RETURN** tipo **IS**

[declaraciones]

BEGIN

[sentencias_serie] -- debe incluir al menos un RETURN

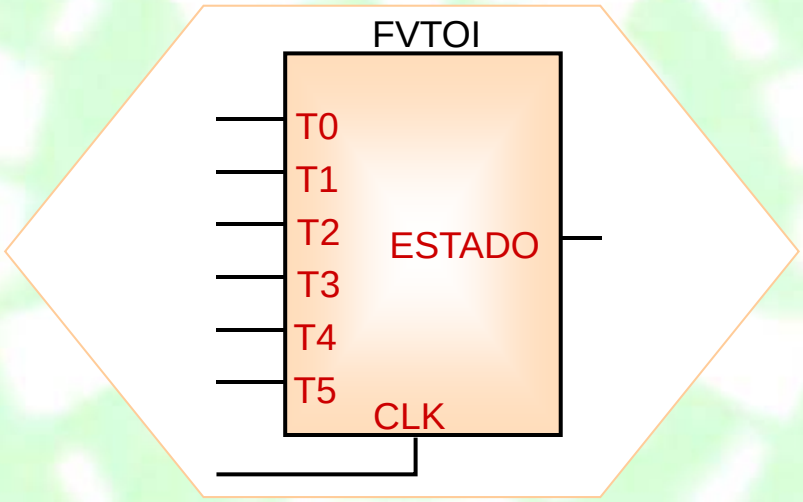
END [FUNCTION] [nombre];

Ejemplo de una FUNCION

Cinvestav

```
-- convertir vector a entero (versión función)
library IEEE;
use ieee.std_logic_1164.all;

entity FVTOI is
port (  CLK: in std_logic;
        T0,T1,T2,T3,T4,T5:  in std_logic;
         ESTADO: out integer range 0 to 63 );
end FVTOI;

architecture RTL of FVTOI is
signal temporal1: std_logic_vector(5 downto 0);
begin
process(CLK,temporal1)
 begin
  if (CLK'event AND CLK='1') then
     temporal1<=T5&T4&T3&T2&T1&T0;
  end if;
   -- llamada a la funcion vtoi
      ESTADO <= vtoi(temporal1,6);
 end process;
end RTL;
```

```
function vtoi(vin: in std_logic_vector; nbits: in
integer) return integer is
 variable temp: integer range 0 to 63;
begin
   temp:=0;
  for i in 0 to nbits-1 loop
        if (vin(i)='1') then
           temp:=temp + (2**i);
        end if;
  end loop;
  return(temp);
end;
```



Diferencias en Procedimientos y Funciones

Cinvestav

1. Una función siempre devuelve un valor, mientras un procedimiento sólo puede devolver valores a través de los parámetros que se le pasen.

2. Los argumentos de una función son siempre de entrada, por lo que sólo se pueden leer dentro de la función. En el procedimiento pueden ser de entrada, salida o de entrada/salida, por lo que pueden sufrir modificaciones.

3. Una función no tiene efectos colaterales, pero un procedimiento sí, es decir, puede provocar cambios en objetos externos a él debido a que se pueden cambiar las señales aunque no se hubieran especificado en el argumento. Es decir, en los procedimientos se pueden realizar asignaciones sobre señales declaradas en la arquitectura y, por lo tanto, externas al procedimiento.

4. Las funciones, como devuelven un valor, se usan en expresiones, mientras que los procedimientos se llaman como una sentencia secuencial o concurrente.

5. La función debe contener la palabra clave **RETURN** seguida de una expresión puesto que siempre devuelve un valor, mientras que en el procedimiento no es necesario.

6. Una función jamás puede tener la instrucción **WAIT**, mientras que un procedimiento sí.

Cinvestav

Capítulo 6

VHDL para simulación

VHDL para simulación

Cinvestav

El lenguaje VHDL sirve también para la descripción de modelos para simulación, sus principales características son:

- *No tiene demasiadas restricciones.*
- *Solo se requiere un interprete de las instrucciones.*
- *No importa el nivel de abstracción.*

Hay una serie de elementos que solo tiene significado en un entorno de simulación, estos son:

- *Especificación de Retardos.*
- *Notificación de sucesos.*
- *Descripción del banco de pruebas.*

Especificación de retardos

Cinvestav

Especificación de retardos

Para indicar un retardo en las asignaciones se emplea la palabra **AFTER**

señal <= ‘0’ **AFTER** 15 ns;

Cuando se le asigna un valor a una señal, no se le asigna de forma inmediata, sino que se le asigna a su *fuente* (driver).

En la sentencia se ha especificado el tiempo en el cual se realizará realmente la asignación, de esta manera la información de la *fuente* pasa a la señal cuando la simulación llega a este tiempo.

Gracias al concepto de *evento* es fácil entender que en una asignación se pueden programar varios eventos o sucesos que tendrán en el futuro.

señal <= ‘1’ **AFTER** 4 ns, ‘0’ **AFTER** 20 ns;

Retardos inerciales y transportados

Cinvestav

Retardos inerciales

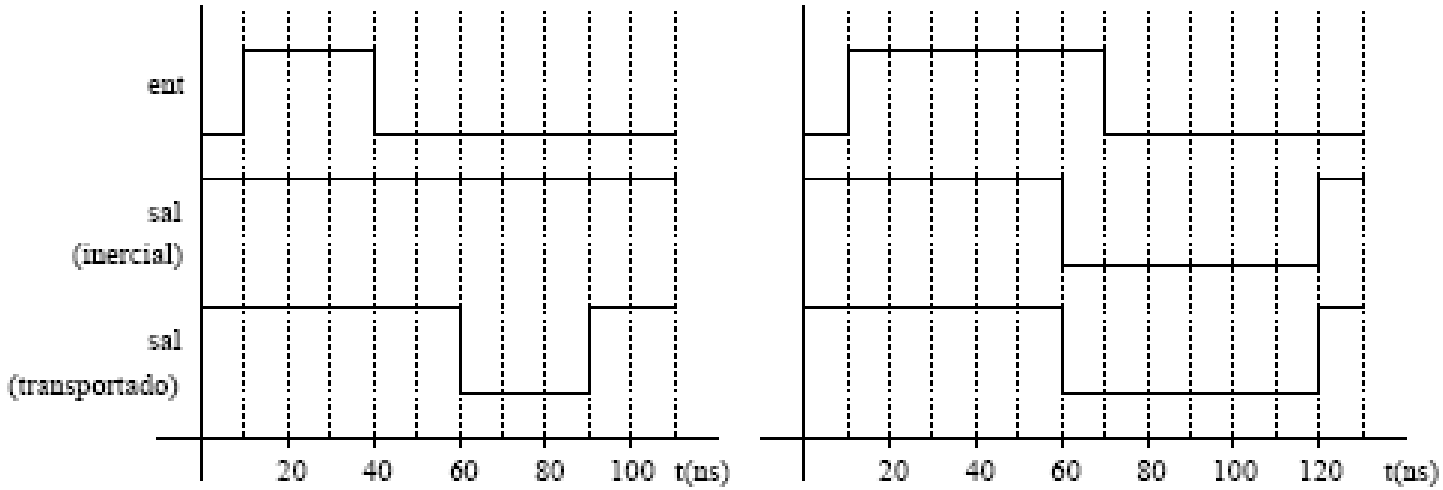
Elimina el evento que hubiera en la lista y lo sustituye con el nuevo evento.

Retardos transportados

Lo que se hace es simplemente introducir el evento en la lista.

sal <= **INERTIAL not** ent **AFTER** 50 ns;

sal <= **TRANSPORT not** ent **AFTER** 50 ns;



Nota: El retraso inercial es el de defecto, en el caso de asignaciones múltiples sólo la primera es inercial

sal <= **REJECT** 10 ns **INERTIAL not** ent **AFTER** 50 ns;

Modelo de simulación

Cinvestav

La Memoria ROM

Realizar el modelo de simulación de una memoria ROM simple. La ROM tiene una entrada de selección activa a nivel bajo, de manera que cuando está activa, la salida es el contenido de la posición indicada por la dirección de entrada, si no está activa, la salida es alta impedancia. El tiempo que pasa entre que cambia la selección y la salida es de 60 ns. El tiempo que pasa entre que cambia la dirección y cambia la salida es de 100 ns. En el caso de cambio en la dirección, la salida mantiene su valor anterior durante 10 ns y luego pasa a desconocido.

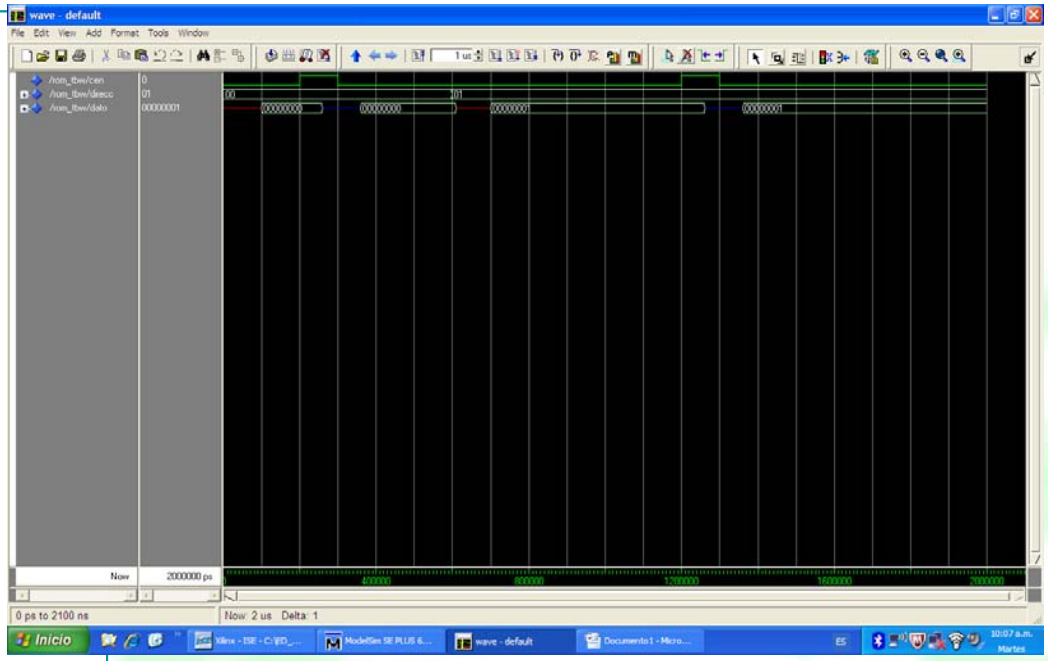
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ENTITY rom IS
7  PORT( cen: IN std_logic;
8  direcc: IN std_logic_vector(1 DOWNTO 0);
9  dato: OUT std_logic_vector(7 DOWNTO 0));
10 END rom;
11
12 ARCHITECTURE modelo OF rom IS
13 SIGNAL salida: std_logic_vector(7 DOWNTO 0);
14 SIGNAL cenr: std_logic:
15
16 BEGIN
17 PROCESS(direcc)
18 BEGIN
19 salida<="XXXXXXXX" AFTER 10 ns;
20    CASE direcc IS
21    WHEN "00"=>salida<=TRANSPORT "00000000" AFTER 100 ns;
22    WHEN "01"=>salida<=TRANSPORT "00000001" AFTER 100 ns;
23    WHEN "10"=>salida<=TRANSPORT "01010101" AFTER 100 ns;
24    WHEN "11"=>salida<=TRANSPORT "10101010" AFTER 100 ns;
25    WHEN OTHERS=> NULL;
26    END CASE;
27 END PROCESS;
28 dato<=salida   WHEN cenr='0' ELSE
29 (OTHERS =>'Z') WHEN cenr='1' ELSE
30 (OTHERS =>'X');
31 cenr<=cen AFTER 60 ns;
32 END modelo;
```

Modelo de simulación

Cinvestav

La Memoria ROM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ENTITY rom IS
7  PORT( cen: IN std_logic;
8  direcc: IN std_logic_vector(1 DOWNTO 0);
9  dato: OUT std_logic_vector(7 DOWNTO 0));
10 END rom;
11
12 ARCHITECTURE modelo OF rom IS
13 SIGNAL salida: std_logic_vector(7 DOWNTO 0);
14 SIGNAL cenr: std_logic:
15
16 BEGIN
17 PROCESS(direcc)
18 BEGIN
19 salida<="XXXXXXXX" AFTER 10 ns;
20    CASE direcc IS
21    WHEN "00"=>salida<=TRANSPORT "00000000" AFTER 100 ns;
22    WHEN "01"=>salida<=TRANSPORT "00000001" AFTER 100 ns;
23    WHEN "10"=>salida<=TRANSPORT "01010101" AFTER 100 ns;
24    WHEN "11"=>salida<=TRANSPORT "10101010" AFTER 100 ns;
25    WHEN OTHERS=> NULL;
26    END CASE;
27 END PROCESS;
28 dato<=salida   WHEN cenr='0' ELSE
29 (OTHERS =>'Z') WHEN cenr='1' ELSE
30 (OTHERS =>'X');
31 cenr<=cen AFTER 60 ns;
32 END modelo;
```

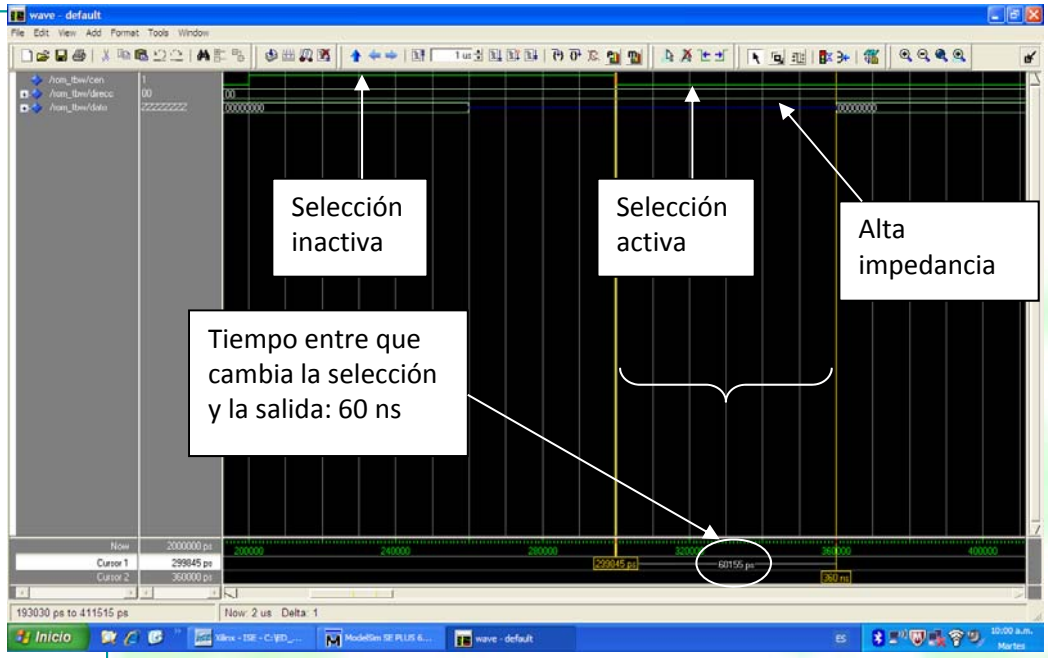


Modelo de simulación

Cinvestav

La Memoria ROM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ENTITY rom IS
7  PORT( cen: IN std_logic;
8  direcc: IN std_logic_vector(1 DOWNTO 0);
9  dato: OUT std_logic_vector(7 DOWNTO 0));
10 END rom;
11
12 ARCHITECTURE modelo OF rom IS
13 SIGNAL salida: std_logic_vector(7 DOWNTO 0);
14 SIGNAL cenr: std_logic:
15
16 BEGIN
17 PROCESS(direcc)
18 BEGIN
19 salida<="XXXXXXXX" AFTER 10 ns;
20    CASE direcc IS
21    WHEN "00"=>salida<=TRANSPORT "00000000" AFTER 100 ns;
22    WHEN "01"=>salida<=TRANSPORT "00000001" AFTER 100 ns;
23    WHEN "10"=>salida<=TRANSPORT "01010101" AFTER 100 ns;
24    WHEN "11"=>salida<=TRANSPORT "10101010" AFTER 100 ns;
25    WHEN OTHERS=> NULL;
26    END CASE;
27 END PROCESS;
28 dato<=salida   WHEN cenr='0' ELSE
29 (OTHERS =>'Z') WHEN cenr='1' ELSE
30 (OTHERS =>'X');
31 cenr<=cen AFTER 60 ns;
32 END modelo;
```

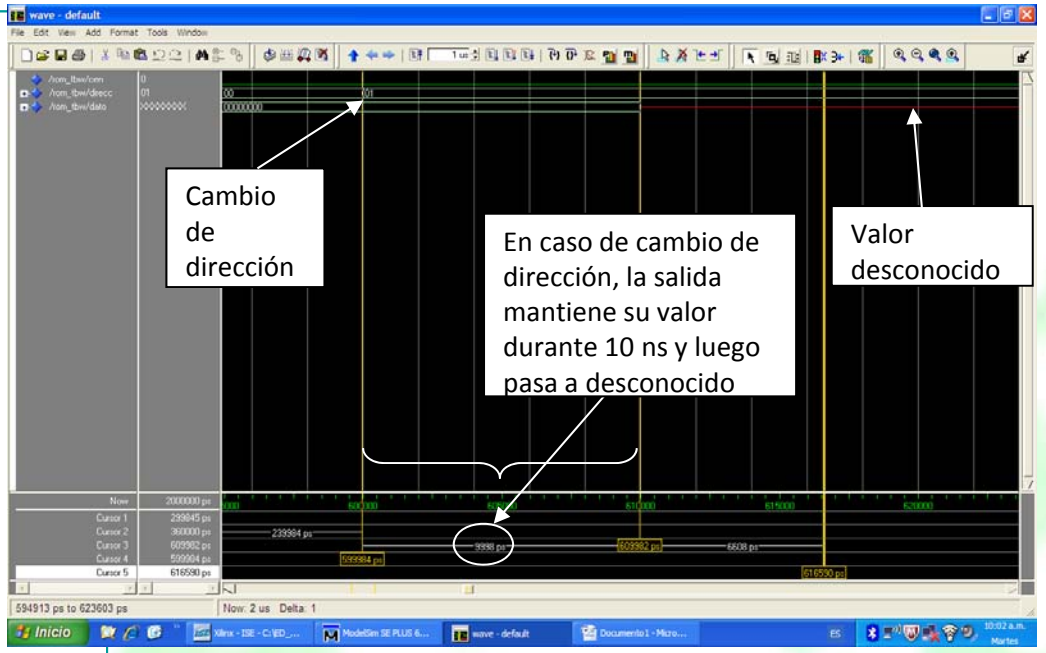


Modelo de simulación

Cinvestav

La Memoria ROM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ENTITY rom IS
7  PORT( cen: IN std_logic;
8  direcc: IN std_logic_vector(1 DOWNTO 0);
9  dato: OUT std_logic_vector(7 DOWNTO 0));
10 END rom;
11
12 ARCHITECTURE modelo OF rom IS
13 SIGNAL salida: std_logic_vector(7 DOWNTO 0);
14 SIGNAL cenr: std_logic:
15
16 BEGIN
17 PROCESS(direcc)
18 BEGIN
19 salida<="XXXXXXXX" AFTER 10 ns;
20    CASE direcc IS
21    WHEN "00"=>salida<=TRANSPORT "00000000" AFTER 100 ns;
22    WHEN "01"=>salida<=TRANSPORT "00000001" AFTER 100 ns;
23    WHEN "10"=>salida<=TRANSPORT "01010101" AFTER 100 ns;
24    WHEN "11"=>salida<=TRANSPORT "10101010" AFTER 100 ns;
25    WHEN OTHERS=> NULL;
26    END CASE;
27 END PROCESS;
28 dato<=salida   WHEN cenr='0' ELSE
29 (OTHERS =>'Z') WHEN cenr='1' ELSE
30 (OTHERS =>'X');
31 cenr<=cen AFTER 60 ns;
32 END modelo;
```



Modelo de simulación

Cinvestav

La Memoria ROM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ENTITY rom IS
7  PORT( cen: IN std_logic;
8  direcc: IN std_logic_vector(1 DOWNTO 0);
9  dato: OUT std_logic_vector(7 DOWNTO 0));
10 END rom;
11
12 ARCHITECTURE modelo OF rom IS
13 SIGNAL salida: std_logic_vector(7 DOWNTO 0);
14 SIGNAL cenr: std_logic:
15
16 BEGIN
17 PROCESS(direcc)
18 BEGIN
19 salida<="XXXXXXXX" AFTER 10 ns;
20    CASE direcc IS
21    WHEN "00"=>salida<=TRANSPORT "00000000" AFTER 100 ns;
22    WHEN "01"=>salida<=TRANSPORT "00000001" AFTER 100 ns;
23    WHEN "10"=>salida<=TRANSPORT "01010101" AFTER 100 ns;
24    WHEN "11"=>salida<=TRANSPORT "10101010" AFTER 100 ns;
25    WHEN OTHERS=> NULL;
26    END CASE;
27 END PROCESS;
28 dato<=salida   WHEN cenr='0' ELSE
29 (OTHERS =>'Z') WHEN cenr='1' ELSE
30 (OTHERS =>'X');
31 cenr<=cen AFTER 60 ns;
32 END modelo;
```

Notificación de sucesos

Cinvestav

Notificación de sucesos

Durante la simulación de un circuito descrito en VHDL, se pueden notificar ciertos sucesos mediante la utilización de la palabra clave **ASSERT,** que tiene como elemento de activación una condición:

ASSERT condición **REPORT** mensaje **SEVERITY** nivel gravedad

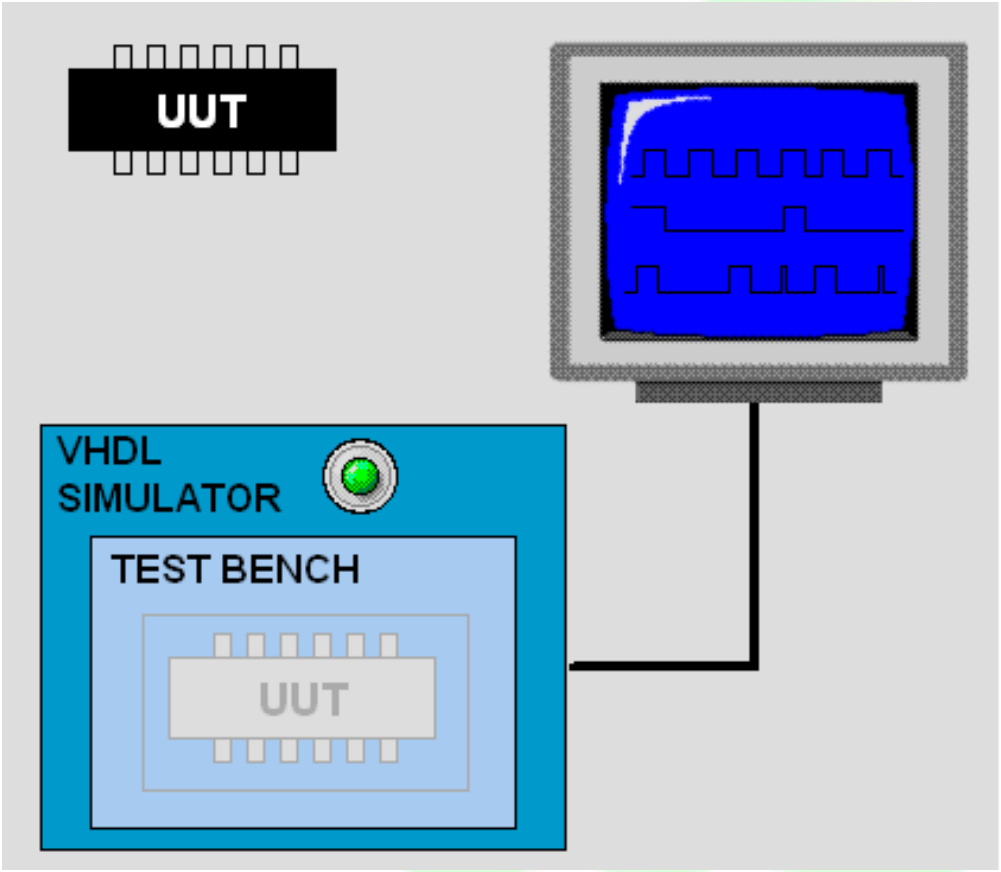
Si no se cumple la condición especificada entonces se saca el mensaje especificado por pantalla y se da además un nivel de gravedad. Tanto el mensaje como el nivel de gravedad son opcionales. Si no se especifica ningún mensaje aparece la cadena "Assertion Violation".
Los niveles de gravedad que hay son: **NOTE**, **WARNING**, **ERROR** y **FAILURE**, y si no se especifica nada el valor por defecto es ERROR.

¿Qué es un Banco de Pruebas?

Cinvestav

Banco de pruebas (Test Bench)

Un banco de pruebas no es más que la definición de un conjunto de entradas llamadas patrones de prueba, con las que se verifica el funcionamiento del circuito.



Funciones del Banco de Pruebas

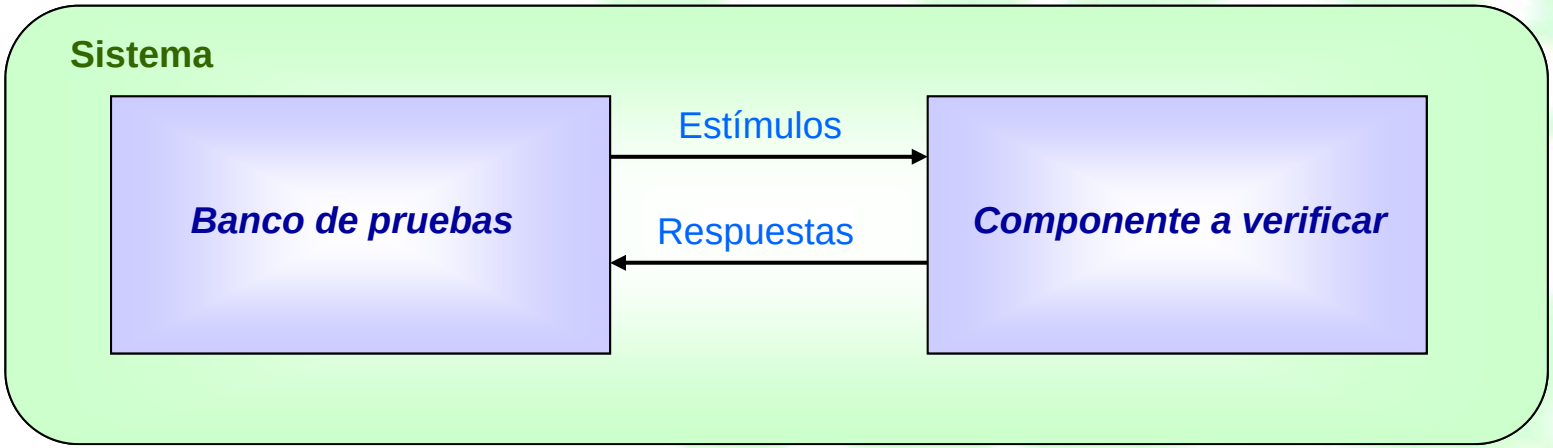
Cinvestav

Funciones del banco de pruebas

El banco de pruebas se conecta con el componente a verificar mediante dos tipos de señales:

- Estímulos
- Respuestas

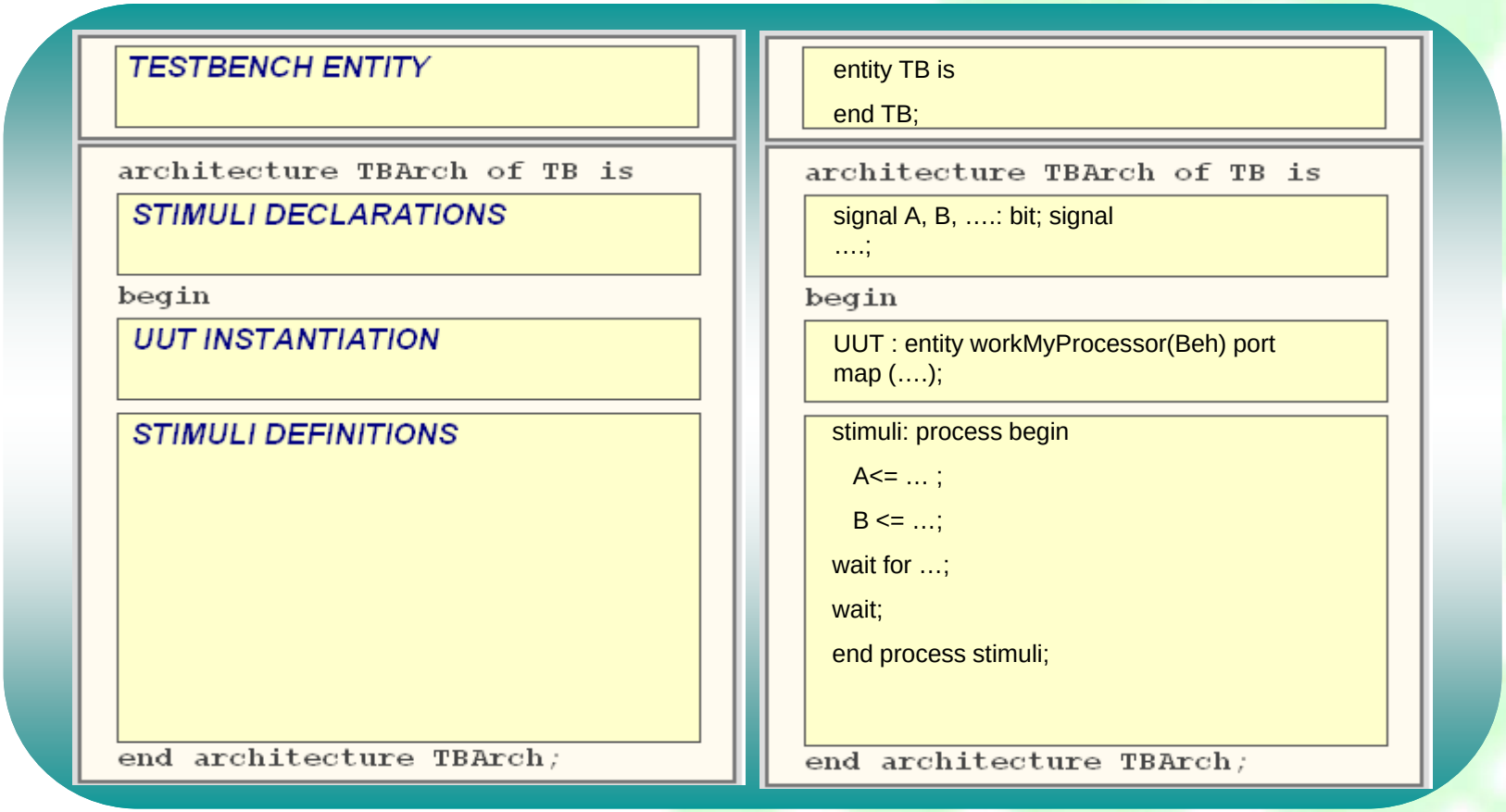
Sus funciones son la generación de estímulos y el análisis de las respuestas.



Estructura del Banco de Pruebas

Cinvestav

El banco de pruebas consta de una entidad sin puertos. La arquitectura es de tipo estructural, tiene como señales internas las entradas y salidas del circuito, el único llamado a componente es el correspondiente a la entidad que se desea simular y por último una sección de definición de estímulos.



Metodologías para un Banco de Pruebas

Cinvestav

Metodologías para un banco de pruebas

La construcción de un banco de pruebas para un circuito determinado puede abordarse de distintas maneras. Según la forma de generación de los vectores de prueba, estas metodologías se pueden clasificar en:

Método Tabular

Utilización de Archivos

Metodología Algorítmica

WAIT

Cinvestav

WAIT condición usada para detener un proceso**.**

En VHDL se usan tres tipos de enunciados ***wait***.

- ***wait for*** tipo_expresión -- espera por algún lapso de tiempo

```
wait for 10ns;
wait for periodo/2;
```

- ***wait until*** condición -- espera hasta que la condición booleana se cumple

```
wait until A and B;
wait until clk='1';
```

- ***wait on*** lista_sensitiva -- espera por un cambio de valor de la señal

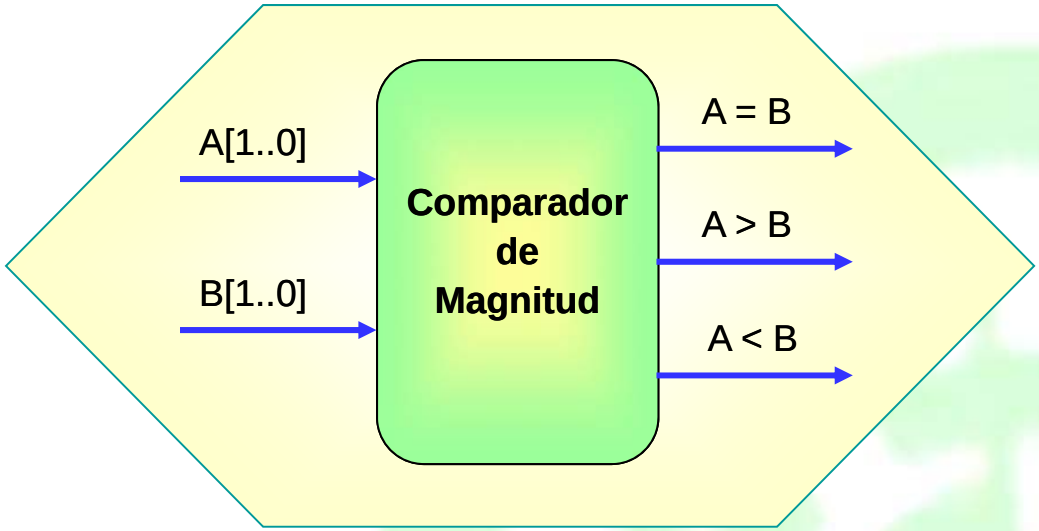
```
wait on clk;
wait on enable,data;
```

- Se pueden realizar condiciones *complejas*, combinando las diferentes formas anteriores. En este caso se realiza el *and* de las dos primeras *or* la tercera.

```
wait on a,b until (a='1'and b='0') for 10ns;
```

Ejemplo: Comparador de magnitud de 2 bits

Cinvestav



Función	sal2	sal1	sal0
A = B	1	0	0
A > B	0	1	0
A < B	0	0	1

Solución: Comparador de magnitud de 2 bits

Cinvestav

Descripción en VHDL del Comparador de Magnitud

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity Comparador is
7       Port ( A : in std_logic_vector(1 downto 0);
8              B : in std_logic_vector(1 downto 0);
9              Sal : out std_logic_vector(2 downto 0));
10  end Comparador;
11
12  architecture Arq_comparador of Comparador is
13
14  begin
15
16     process (A,B)
17        begin
18        if (A = B) then
19           Sal <= "100";
20        elsif (A > B) then
21           Sal <="010";
22        else
23           Sal <= "001";
24        end if;
25     end process;
26
27  end Arq_comparador;
```

Metodologías para un banco de pruebas

Cinvestav

Método tabular

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.numeric_std.ALL;
4
5   ENTITY testbench IS
6   END testbench;
7
8   ARCHITECTURE behavior OF testbench IS
9
10     COMPONENT comparador
11     PORT(
12        a : IN std_logic_vector(1 downto 0);
13        b : IN std_logic_vector(1 downto 0);
14        sal : OUT std_logic_vector(2 downto 0)
15        );
16     END COMPONENT;
17
18     SIGNAL a :  std_logic_vector(1 downto 0);
19     SIGNAL b :  std_logic_vector(1 downto 0);
20     SIGNAL sal : std_logic_vector(2 downto 0);
21
22  BEGIN
23
24     uut: comparador PORT MAP(
25        a => a,
26        b => b,
27        sal => sal
28     );
```

```
29
30
31  -- *** Test Bench - User Defined Section ***
32     tb : PROCESS
33     BEGIN
34
35     A<="00";  B<="00"; wait for 100 ns;
36     A<="00";  B<="01"; wait for 100 ns;
37     A<="00";  B<="10"; wait for 100 ns;
38     A<="00";  B<="11"; wait for 100 ns;
39     A<="01";  B<="00"; wait for 100 ns;
40     A<="01";  B<="01"; wait for 100 ns;
41     A<="01";  B<="10"; wait for 100 ns;
42     A<="01";  B<="11"; wait for 100 ns;
43     A<="10";  B<="00"; wait for 100 ns;
44     A<="10";  B<="01"; wait for 100 ns;
45     A<="10";  B<="10"; wait for 100 ns;
46     A<="10";  B<="11"; wait for 100 ns;
47     A<="11";  B<="00"; wait for 100 ns;
48     A<="11";  B<="01"; wait for 100 ns;
49     A<="11";  B<="10"; wait for 100 ns;
50     A<="11";  B<="11"; wait for 100 ns;
51
52     wait;
53     END PROCESS;
54  -- *** End Test Bench - User Defined Section ***
55
56  END;
```

Metodologías para un banco de pruebas

Cinvestav

Utilización de Archivos

```
1   00 00 100
2   00 01 001
3   00 10 001
4   00 11 001
5   01 00 010
6   01 01 100
7   01 10 001
8   01 11 001
9   10 00 010
10  10 01 010
11  10 10 100
12  10 11 001
13  11 00 010
14  11 01 010
15  11 10 010
16  11 11 010
```

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  USE std.textio.all;
5
6  ENTITY testbench IS
7  END testbench;
8
9  ARCHITECTURE behavior OF testbench IS
10
11     COMPONENT comparador
12     PORT(
13        a : IN bit_vector(1 downto 0);
14        b : IN bit_vector(1 downto 0);
15        sal : OUT bit_vector(2 downto 0)
16        );
17     END COMPONENT;
18
19     SIGNAL a :  bit_vector(1 downto 0);
20     SIGNAL b :  bit_vector(1 downto 0);
21     SIGNAL sal :  bit_vector(2 downto 0);
22
23 BEGIN
24
25     uut: comparador PORT MAP(
26        a => a,
27        b => b,
28        sal => sal
29     );
```

```
30
31
32 -- *** Test Bench - User Defined Section ***
33     tb : PROCESS
34     FILE archivo : TEXT OPEN read_mode IS "datos.txt";
35     VARIABLE invector : LINE;
36     VARIABLE vSal : BIT_VECTOR(2 DOWNTO 0);
37     VARIABLE vA, vB : BIT_VECTOR(1 DOWNTO 0);
38     BEGIN
39     WHILE NOT ENDFILE(archivo) LOOP
40       READLINE(archivo,invector);   --Se lee una linea del archivo
41       READ(invector,vA);
42       READ(invector,vB);
43       READ(invector,vSal);
44       --Se aplica el vector de test
45       A <= vA; B <= vB;
46       WAIT FOR 100 ns;
47       ASSERT Sal=vSal REPORT "Salida incorrecta. " SEVERITY error;
48     END LOOP;
49        wait;
50     END PROCESS;
51 -- *** End Test Bench - User Defined Section ***
52
53 END;
```

Metodologías para un banco de pruebas

Cinvestav

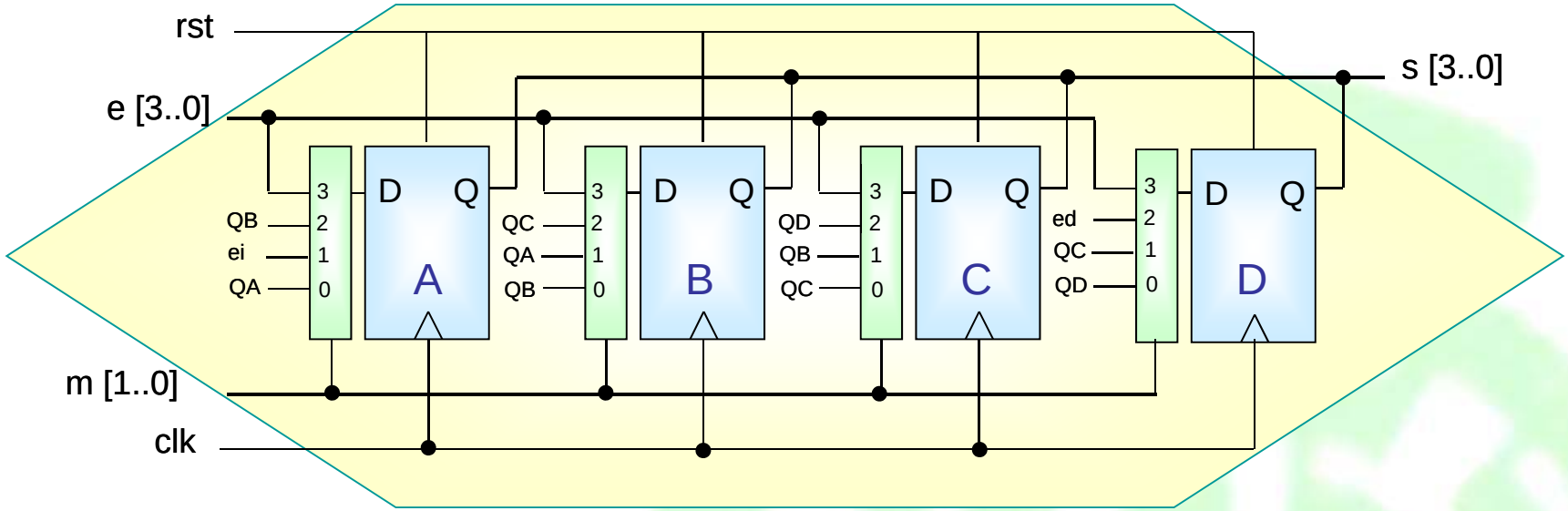
Metodología Algorítmica

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  USE ieee.std_logic_arith.ALL;
5
6  ENTITY testbench IS
7  END testbench;
8
9  ARCHITECTURE behavior OF testbench IS
10
11    COMPONENT comparador
12    PORT(
13       a : IN std_logic_vector(1 downto 0);
14       b : IN std_logic_vector(1 downto 0);
15       sal : OUT std_logic_vector(2 downto 0)
16       );
17    END COMPONENT;
18
19    SIGNAL a :  std_logic_vector(1 downto 0);
20    SIGNAL b :  std_logic_vector(1 downto 0);
21    SIGNAL sal :  std_logic_vector(2 downto 0);
22
23 BEGIN
24
25    uut: comparador PORT MAP(
26       a => a,
27       b => b,
28       sal => sal
29    );
```

```
30
31
32 -- *** Test Bench - User Defined Section ***
33    tb : PROCESS
34
35    BEGIN
36    for i in 0 to 3 loop
37    A <= CONV_STD_LOGIC_VECTOR(i,2);
38       for j in 0 to 3 loop
39       B <= CONV_STD_LOGIC_VECTOR(j,2);
40          wait for 100 ns;
41       end loop;
42    end loop;
43    wait;
44    END PROCESS;
45 -- *** End Test Bench - User Defined Section ***
46
47 END;
48
```

Ejemplo: Registro de desplazamiento síncrono de 4 bits

Cinvestav



m1 m0	s3 s2 s1 s0	Función
0 0	QA QB QC QD	Mantiene
0 1	ei QA QB QC	Desplaz. derecha
1 0	QB QC QD ed	Desplaz. izquierda
1 1	e3 e2 e1 e0	Carga

Solución: Registro de desplazamiento síncrono de 4 bits

Cinvestav

Descripción en VHDL del Registro de Desplazamiento

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.all;
3
4  ENTITY desplazamiento IS PORT(
5             clk, rst, ed, ei : IN BIT;
6             m : IN BIT_vector(1 DOWNTO 0);
7             e : IN BIT_vector(3 DOWNTO 0);
8             sal : OUT BIT_vector(3 DOWNTO 0));
9  END desplazamiento;
10
11 ARCHITECTURE comportamiento OF desplazamiento IS
12 --Señal auxiliar para leer la salida:
13 SIGNAL s: BIT_vector (3 DOWNTO 0);
14 BEGIN
15   sal <= s;  -- Así se puede leer la salida.
16   PROCESS (clk,rst)
17   BEGIN
18    IF rst='1' THEN s<="0000";
19    ELSIF (clk'EVENT AND clk='1') THEN
20       CASE  m IS
21         WHEN   "01"   => s(3)<=ei;   s(2)<=s(3);   --Desplazamiento a
22                          s(1)<=s(2); s(0)<=s(1);   --la derecha
23         WHEN   "10"   => s(3)<=s(2); s(2)<=s(1);   --Desplazamiento a
24                          s(1)<=s(0); s(0)<=ed;     --la izquierda
25         WHEN   "11"   => s<=e;                     --Carga en paralelo
26         WHEN   OTHERS => NULL;                     --La salida no cambia
27       END CASE;
28    END IF;
29   END PROCESS;
30 END comportamiento;
31
```

Metodologías para un banco de pruebas

Cinvestav

Método tabular

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  ENTITY testbench IS
6  END testbench;
7
8  ARCHITECTURE behavior OF testbench IS
9
10    COMPONENT desplazamiento
11    PORT(
12       clk, rst, ed, ei : IN std_logic;
13       m : IN std_logic_vector(1 downto 0);
14       e : IN std_logic_vector(3 downto 0);
15       sal : OUT std_logic_vector(3 downto 0)
16       );
17    END COMPONENT;
18
19    SIGNAL clk, rst, ed, ei :  std_logic;
20    SIGNAL m :  std_logic_vector(1 downto 0);
21    SIGNAL e :  std_logic_vector(3 downto 0);
22    SIGNAL sal :  std_logic_vector(3 downto 0);
23
24 BEGIN
25
26    uut: desplazamiento PORT MAP(
27       clk => clk,
28       rst => rst,
29       ed => ed,
30       ei => ei,
31       m => m,
32       e => e,
33       sal => sal
34    );
```

```
35
36
37    tb : PROCESS
38
39     BEGIN
40    --Se da reset
41    clk<='0'; rst<='1'; ed<='-'; ei<='-'; m<="--"; e<="----";wait for 25 ns;
42    clk<='1'; rst<='1'; ed<='-'; ei<='-'; m<="--"; e<="----";wait for 25 ns;
43    clk<='0'; rst<='0'; ed<='-'; ei<='-'; m<="00"; e<="----";wait for 25 ns;
44    ASSERT sal="0000" REPORT "Salida incorrecta. " SEVERITY error;
45
46
47    --Se comprueba que no se carga nada en el modo 0
48    clk<='1'; rst<='0'; ed<='-'; ei<='-'; m<="00"; e<="1011";wait for 25 ns;
49    clk<='0'; rst<='0'; ed<='-'; ei<='-'; m<="00"; e<="1011";wait for 25 ns;
50    ASSERT sal="0000" REPORT "Salida incorrecta. " SEVERITY error;
51
52    --Se carga el código 1010 en paralelo
53    clk<='1'; rst<='0'; ed<='-'; ei<='-'; m<="11"; e<="1010";wait for 25 ns;
54    clk<='0'; rst<='0'; ed<='-'; ei<='-'; m<="11"; e<="1010";wait for 25 ns;
55    ASSERT sal="1010" REPORT "Salida incorrecta. " SEVERITY error;
56
57    --Se desplaza hacia la derecha 0->101
58    clk<='1'; rst<='0'; ed<='-'; ei<='0'; m<="01"; e<="----";wait for 25 ns;
59    clk<='0'; rst<='0'; ed<='-'; ei<='0'; m<="01"; e<="----";wait for 25 ns;
60    ASSERT sal="0101" REPORT "Salida incorrecta. " SEVERITY error;
61
62    --Se desplaza hacia la izquierda 101<-1
63    clk<='1'; rst<='0'; ed<='1'; ei<='-'; m<="10"; e<="----";wait for 25 ns;
64    clk<='0'; rst<='0'; ed<='1'; ei<='-'; m<="10"; e<="----";wait for 25 ns;
65    ASSERT sal="1011" REPORT "Salida incorrecta. " SEVERITY error;
66    WAIT;
67
68    END PROCESS;
69
70 END;
71
```

Metodologías para un banco de pruebas

Cinvestav

Utilización de Archivos

```
0   1   0   0   00   0000   0000
1   1   0   0   00   0000   0000
0   0   0   0   00   0000   0000
1   0   0   0   00   1011   0000
0   0   0   0   00   1011   0000
1   0   0   0   11   1011   1011
0   0   0   0   11   1011   1011
1   0   0   0   01   0000   0101
0   0   0   0   01   0000   0101
1   0   1   0   10   0000   1011
0   0   1   0   10   0000   1011
```

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4  USE std.textio.all;
5
6  ENTITY testbench IS
7  END testbench;
8
9  ARCHITECTURE behavior OF testbench IS
10
11    COMPONENT desplazamiento
12    PORT(
13       clk, rst, ed, ei : IN BIT;
14       m : IN BIT_vector(1 downto 0);
15       e : IN BIT_vector(3 downto 0);
16       sal : OUT BIT_vector(3 downto 0)
17       );
18    END COMPONENT;
19
20    SIGNAL clk, rst,ed,ei : BIT;
21    SIGNAL m :  BIT_vector(1 downto 0);
22    SIGNAL e :  BIT_vector(3 downto 0);
23    SIGNAL sal :  BIT_vector(3 downto 0);
24
25 BEGIN
26
27    uut: desplazamiento PORT MAP(
28       clk => clk,
29       rst => rst,
30       ed => ed,
31       ei => ei,
32       m => m,
33       e => e,
34       sal => sal
35    );
36
37
```

```
38 -- *** Test Bench - User Defined Section ***
39    tb : PROCESS
40    FILE archivo : TEXT OPEN read_mode IS "datos.txt";
41    VARIABLE invector : LINE;
42    VARIABLE vclk, vrst, ved, vei : BIT;
43    VARIABLE vm : BIT_VECTOR(1 DOWNTO 0);
44    VARIABLE ve, vs : BIT_VECTOR(3 DOWNTO 0);
45    BEGIN
46    WHILE NOT ENDFILE(archivo) LOOP
47      READLINE(archivo,invector);   --Se lee una linea del archivo
48      READ(invector,vclk);
49      READ(invector,vrst);
50      READ(invector,ved);
51      READ(invector,vei);
52      READ(invector,vm);
53      READ(invector,ve);
54      READ(invector,vs);
55      --Se aplica el vector de test
56      clk<=vclk; rst<=vrst; ed<=ved; ei<=vei; m<=vm; e<=ve;
57      WAIT FOR 25 ns;
58      ASSERT sal=vs REPORT "Salida incorrecta. " SEVERITY error;
59    END LOOP;
60    wait; -- will wait forever
61    END PROCESS;
62 -- *** End Test Bench - User Defined Section ***
63
64 END;
65
```

Metodologías para un banco de pruebas

Cinvestav

Metodología Algorítmica

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  ENTITY testbench IS
6  END testbench;
7
8  ARCHITECTURE behavior OF testbench IS
9
10    COMPONENT desplazamiento
11    PORT(
12       clk,rst,ed,ei : IN std_logic;
13       m : IN std_logic_vector(1 downto 0);
14       e : IN std_logic_vector(3 downto 0);
15       sal : OUT std_logic_vector(3 downto 0)
16       );
17    END COMPONENT;
18
19    SIGNAL clk,rst,ed,ei :  std_logic;
20    SIGNAL m :  std_logic_vector(1 downto 0);
21    SIGNAL e :  std_logic_vector(3 downto 0);
22    SIGNAL sal :  std_logic_vector(3 downto 0);
23
24
25 BEGIN
26
27    uut: desplazamiento PORT MAP(
28       clk => clk,
29       rst => rst,
30       ed => ed,
31       ei => ei,
32       m => m,
33       e => e,
34       sal => sal
35    );
36
```

```
38 -- Se genera la prueba y se verifica el resultado
39    tb : PROCESS
40    VARIABLE reg_aux : STD_LOGIC_VECTOR(3 DOWNTO 0):="0000";
41
42    BEGIN
43    --Inicialmente se hace un reset
44    rst<='1'; WAIT FOR 20 ns;
45    ASSERT (sal=reg_aux) REPORT "Reset incorrecto" SEVERITY ERROR;
46    WAIT FOR 20 ns;
47
48    --Se desplaza un '1' hacia la izquierda
49    FOR i IN 0 TO 3 LOOP
50       clk<='0'; rst<='0'; ed<='1'; m<="10";
51       WAIT FOR 10 ns;
52       clk<='1'; rst<='0'; ed<='1'; m<="10"; reg_aux:=reg_aux(2 DOWNTO 0) & '1';
53       WAIT FOR 10 ns;
54       ASSERT (sal=reg_aux) REPORT "Desplazamiento a la izquierda incorrecto"
55       SEVERITY ERROR;
56    END LOOP;
57
58    --Se desplaza un '0' hacia la derecha
59    FOR i IN 0 TO 3 LOOP
60       clk<='0'; rst<='0'; ei<='0'; m<="01";
61       WAIT FOR 10 ns;
62       clk<='1'; rst<='0'; ei<='0'; m<="01"; reg_aux:='0' & reg_aux(3 DOWNTO 1);
63       WAIT FOR 10 ns;
64       ASSERT (sal=reg_aux) REPORT "Desplazamiento a la derecha incorrecto"
65       SEVERITY ERROR;
66    END LOOP;
67
68    --Se prueba la carga en paralelo
69    clk<='0'; rst<='0'; e<="1011"; m<="11";
70    WAIT FOR 10 ns;
71    clk<='1'; rst<='0'; e<="1011"; m<="11";
72    WAIT FOR 10 ns;
73    ASSERT (sal="1011") REPORT "Carga incorrecta" SEVERITY ERROR;
74
75    ASSERT FALSE REPORT "Test finalizado" SEVERITY NOTE;
76
77    END PROCESS;
78 END;
```

Proceso: enunciados secuenciales

Cinvestav

Proceso

Enunciados Secuenciales

- Enunciados de Asignación de Variables
- Enunciados de Asignación de Señales
- Enunciados ***if***
- Enunciados ***case***
- Enunciados ***next***
- Enunciados ***exit***
- Enunciados ***null***
- Enunciados ***return***
- Enunciados ***loop***
- Enunciados ***wait***
- Enunciados de Subprogramas (***Procedure*** *y* ***function***)

NEXT, EXIT, NULL, RETURN

Cinvestav

NEXT salta instrucciones y continúa con otra iteración.

SINTAXIS

next [etiqueta**] when [**condición**];**

EXIT salta instrucciones y continúa fuera de una iteración.

SINTAXIS

exit [etiqueta**] when [**condición**];**

NULL no ejecuta ninguna acción.

SINTAXIS

null;

RETURN termina un subprograma (función o procedimiento).

SINTAXIS

return *expresión*; -- *para función*

return; -- *para procedimiento*.

LOOP

Cinvestav

LOOP ejecuta repetidamente una secuencia de instrucciones.

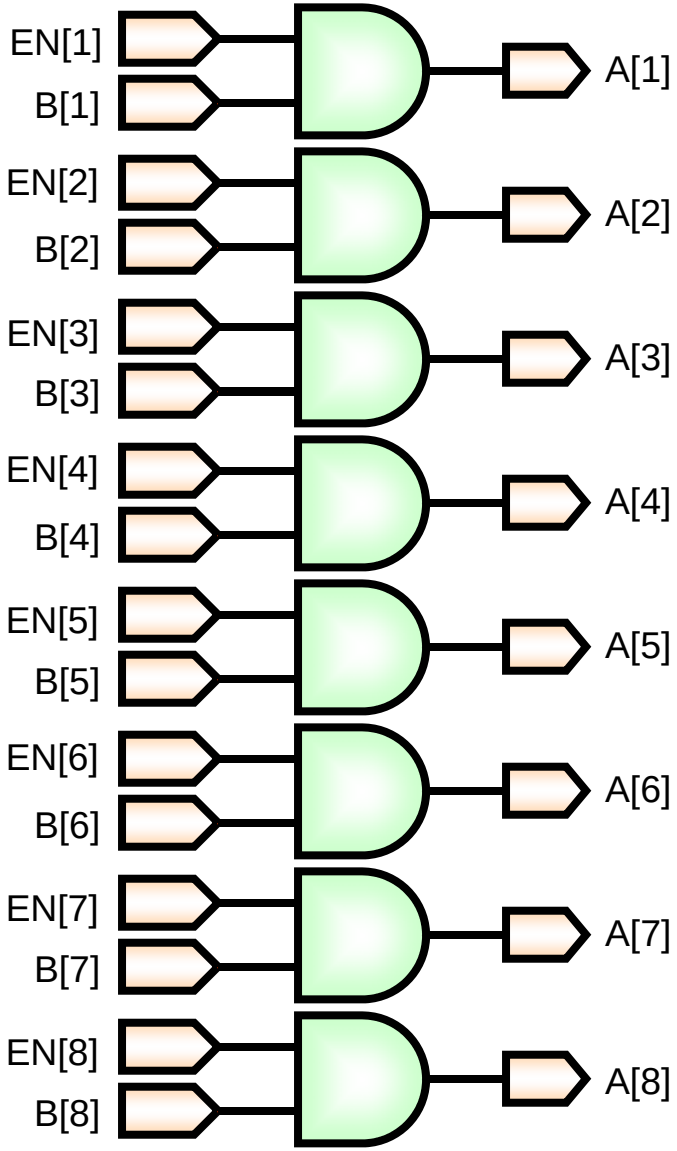
SINTAXIS

[*etiqueta***:]** **[***esquema de iteración***] loop**
{*enunciados secuenciales***}**
{next [*etiqueta***] when [***condición***];}**
{exit [*etiqueta***] when [***condición***];}**
end loop [*etiqueta***];**

DESCRIPCION

etiqueta: nombre opcional del lazo, útil para lazos anidados.
esquema de iteración: puede ser ***while*** o ***for***.
condición: expresión boolena.
next: salta instrucciones y continúa con otra iteración.
exit: salta instrucciones y continúa fuera de LOOP.

Ejemplo: Habilitador de ducto



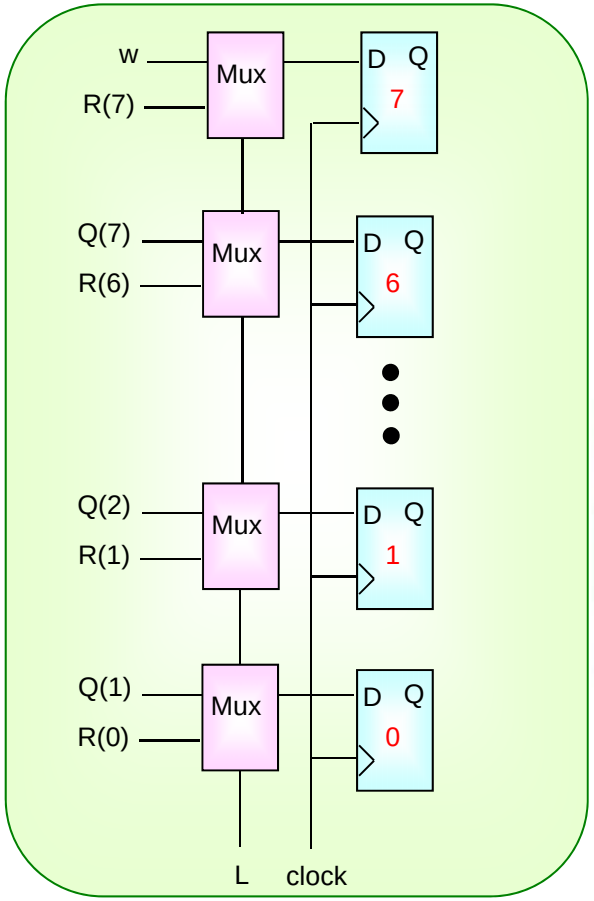
Ejemplo: Habilitador de ducto

```
library ieee;

use ieee.std_logic_1164.all;

entity habilitador is
port( B, EN: in bit_vector(1 to 8);
              A: out bit_vector(1 to 8));
end habilitador;

architecture RTL of habilitador is
 begin
   process( B, EN )
     begin
     A <= "00000000";
     for i in 1 to 8 loop
          next when EN( i ) = '0';
          A( i ) <= B( i );
     end loop;
   end process;
end RTL;
```

Ejemplo de la instrucción LOOP

Cinvestav

Uso de las instrucción LOOP




```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity reg_param is
7      Generic ( n : integer := 8);
8      Port ( R : in std_logic_vector(n-1 downto 0);
9             clock : in std_logic;
10            L, w : in std_logic;
11            Q : inout std_logic_vector(n-1 downto 0));
12 end reg_param;
13
14 architecture arq_reg of reg_param is
15
16 begin
17
18    PROCESS
19    begin
20       wait until clock'event and clock='1';
21       IF L='1' then
22          Q <= R;
23       else
24          for i in 0 to n-2 loop
25             Q(i) <= Q(i+1);
26          end loop;
27             Q(n-1) <= w;
28       end if;
29    end process;
30
31 end arq_reg;
32
```

Cinvestav

Muchas gracias por su atención para cualquier aclaración:

e-mail : mreyes@cinvestav.mx