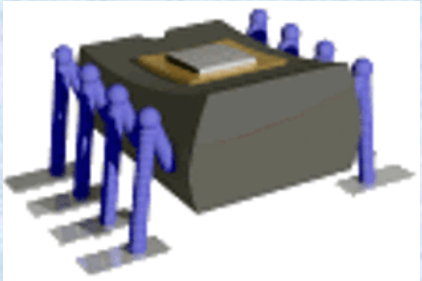
Lenguaje de descripción de Hardware VHSIC

VHDL



Laboratorio de diseño digital

Ingeniería Electrónica

Contenido

- Estilos de diseño
- Lenguaje VHDL: Evolución y características
- Objetos VHDL
- Modelo de Estructura: Entidad y Arquitectura
- Descripción circuitos combinacionales
- Estilos de descripción VHDL
  - Descripción Estructural
  - Descripción RTL
  - Descripción Algorítmica
- Modelo de concurrencia/tiempo
- Descripción circuitos secuenciales

Bibliografía

- Pardo Carpio F. - *Tecnología Informática* - Universidad de Valencia, España - 1997.
- Villar, Teres, Olcoz & Torroja - *VHDL Lenguaje estánda de diseño electrónico* - Mc Graw Hill - 1998
- Wakerly J. F. - *Diseño digital: principios y prácticas* Prentice Hall - 2003
- Aguirre, Tombs & Muñoz - *Diseño de sistemas digitales mediante Lenguajes de Descripción de Hardware* - 2004

Diseño digital

1. Ecuaciones Booleanas → Una ecuación para cada entrada de FF y para cada bloque de puertas.

2. Modelado algorítmico → Redes de Petri

3. Esquemáticos
   - Introducen otros elementos constructivo
   - Expresividad de los métodos gráficos
   - Soportado por herramientas EDA

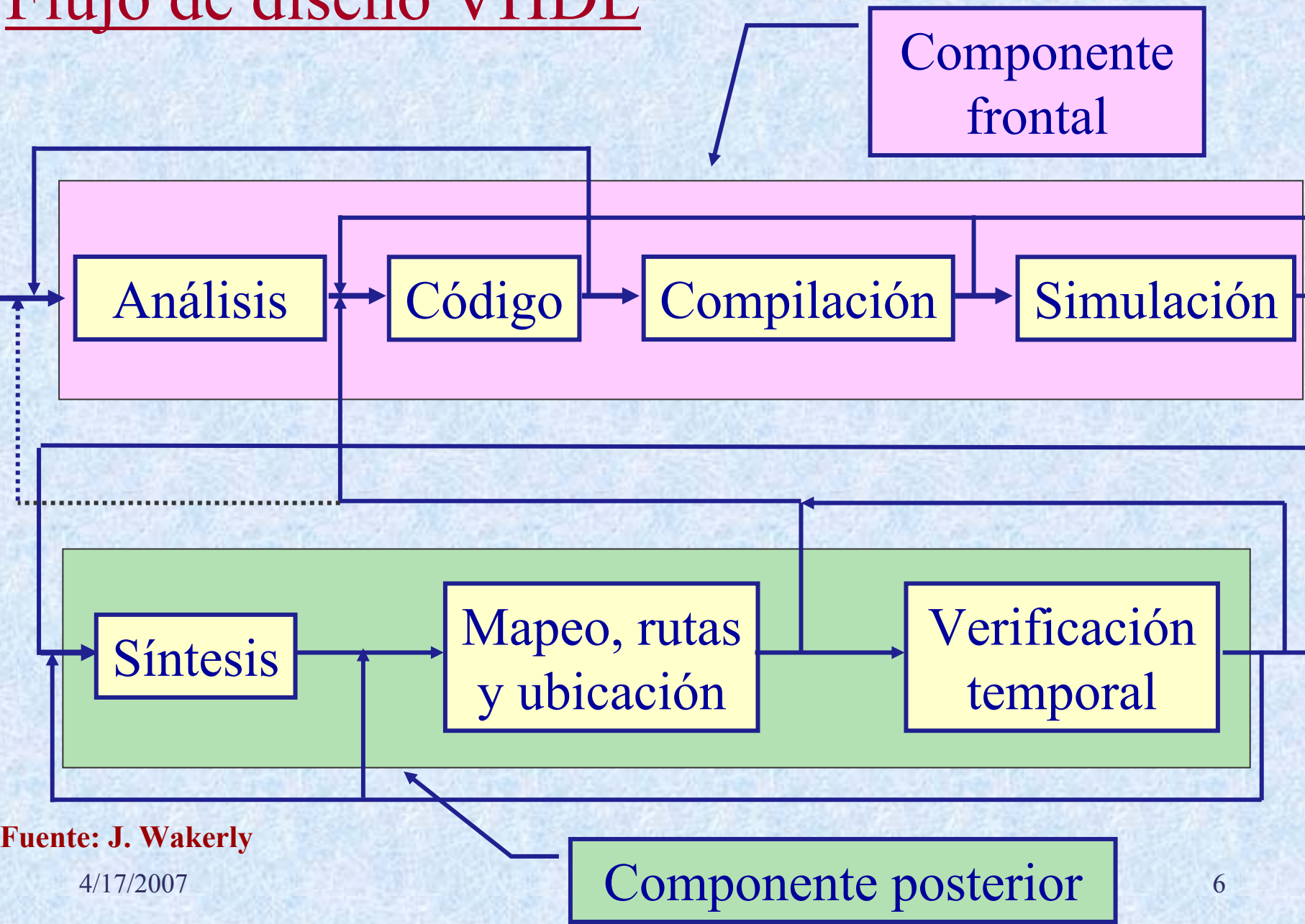
LIMITE → A partir de 6.000 puertas se tornan poco comprensibles.

4. Lenguajes de Descripción de Hardware

- ***Permiten descripciones de alto nivel***
- ***La edición es rápida y sencilla***
- ***Soportados por herramientas EDA***

Las herramientas EDA permiten integrar distintos estilos de diseño

Flujo de diseño VHDL



Fuente: J. Wakerly

Programación de un dispositivo

Se cambian las instrucciones

Cambia el Software

Se cambian las conexiones y las funciones lógicas

Cambia el Hardware

- ***Lenguaje VHDL.***
  - ***Antecedentes***
  - ***Características***

VHDL: Antecedentes históricos

Herramientas EDA → ***Necesidad de intercambio de información***

Primera descripción de un diseño mediante un lenguaje ← NETLIST

↓

Describe los componentes de un circuito y su interconexión

↓

El formato EDIF (Electronic Design Interchange Format) es un estándar de los lenguajes de tipo Netlist

VHDL: Antecedentes históricos

Uso de los lenguajes de tipo Netlist

Sugiere la idea de una descripción de mayor nivel de abstracción (descripción funcional)

Un lenguaje permite la edición más rápida y sencilla

Una descripción con esquemas resulta más fácil de entender

VHDL: Antecedentes históricos

Necesidad de estandarizar los diseños

DoD + IEEE → Patrocinan el desarrollo en los 80´s

Es estándar de IEEE en 1987 → **VHDL-87**

El estándar se amplía en 1993 → **VHDL-93**

Existen otros HDL populares como ABEL y VERILOG

VHDL: Evolución

Se inicia como un leguaje de modelado y especificación.

Ligado a actividades de

- ★ Documentación
- ★ Simulación

Luego se incorporan herramientas de síntesis capaces de manejar este tipo de descripciones

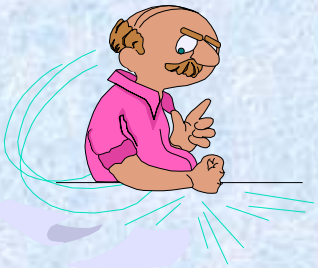
Ambientes integrados de diseño de ASIC´s

VHDL: Evolución

Es un lenguaje para el modelado, simulación lógica dirigida por eventos y síntesis de sistemas de Hardware

Síntesis

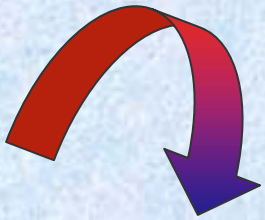
Atraviesa verticalmente los niveles de abstracción de la descripción desde el más alto al más bajo



Subsisten problemas y algunas construcciones de alto nivel no resultan sintetizables

Depende de la herramienta

VHDL: Ventajas:



Es un estándar

- Facilita la documentación, minimiza los errores de comunicación en el equipo de desarrollo e incrementa la portabilidad de los diseños.

Flexibilidad

- Distintas metodologías de diseño: Top-down, Bottom-up, diseño jerárquico.
- Diversos niveles de descripción en los componentes de un mismo circuito.

VHDL:Ventajas

Independencia tecnológica.

- Los diseños pueden implementarse en dispositivos de diferentes tecnologías.

Bibliotecas y paquetes.

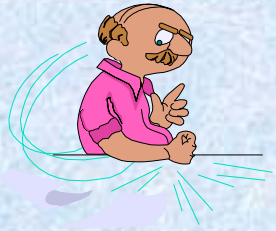
- Gestión de diseño entre miembros de equipos de trabajo grandes.
- Diseño enfocado a la reutilización.

VHDL como lenguaje de alto nivel

Lenguaje fuertemente tipado

El compilador no permite asignar un valor a un objeto a menos que coincida con el tipo declarado previamente para ese objeto.

Programas más confiables y fáciles de depurar



Necesidad de conversiones de tipo

VHDL como lenguaje de alto nivel

Potente control de flujo

Control de condiciones: if-then, case

Control de iteraciones: for, while

Desarrollo y utilización de bibliotecas de diseño

Estructuración de código con funciones y procedimientos

- *Objetos VHDL*
- *Modelo de Estructura*
  - *Entidad*
  - *Arquitectura*

Objetos VHDL: Constantes, variables y señales

Un objeto VHDL es un elemento que guarda un valor de un tipo de datos específico. Sobre él pueden invocarse operaciones coherentes con su tipo.

Compilador

Los objetos VHDL deben declararse en forma previa a su utilización en cualquier unidad de diseño.

Objetos VHDL: Constantes, variables y señales

Constantes { Contribuyen a la legibilidad, mantenimiento y portabilidad del programa. No cambian su valor una vez inicializadas.

constant nombre_constante: tipo := expresión ;

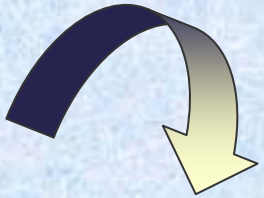
Variables { Cambian su valor con sentencias de asignación
Son locales a los procesos.
No tienen significado físico.

variable nombre_variable: tipo ;
nombre_variable := expresión ;

Objetos VHDL : Constantes, variables y señales

Las señales son abstracciones de conexiones físicas.

Señales



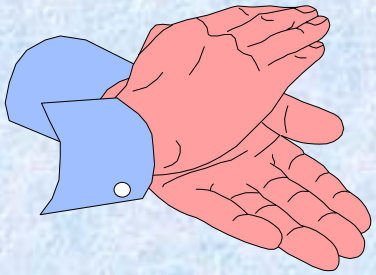
- Cambian su valor con sentencias de asignación.
- Tienen una analogía física directa.
- Interconectan componentes de un circuito.
- Sincronizan la ejecución y suspención de procesos.
- Son visibles para todos elementos de una arquitectura.
- Los puertos de una entidad son señales.

```
signal  nombre_señal: tipo ;
nombre_señal  <= expresión ;
```

VHDL como lenguaje de alto nivel

1º OBJETIVO

Modelar Hardware



VHDL incorpora tres características.

- Modelo de estructura
- Modelo de concurrencia
- Modelo de tiempo

VHDL: Modelo de estructura

Componentes

Dispositivo VHDL

Entidad

Arquitectura

Declaración de I/O

Descripción comportamental

Interfaz

Distintos estilos

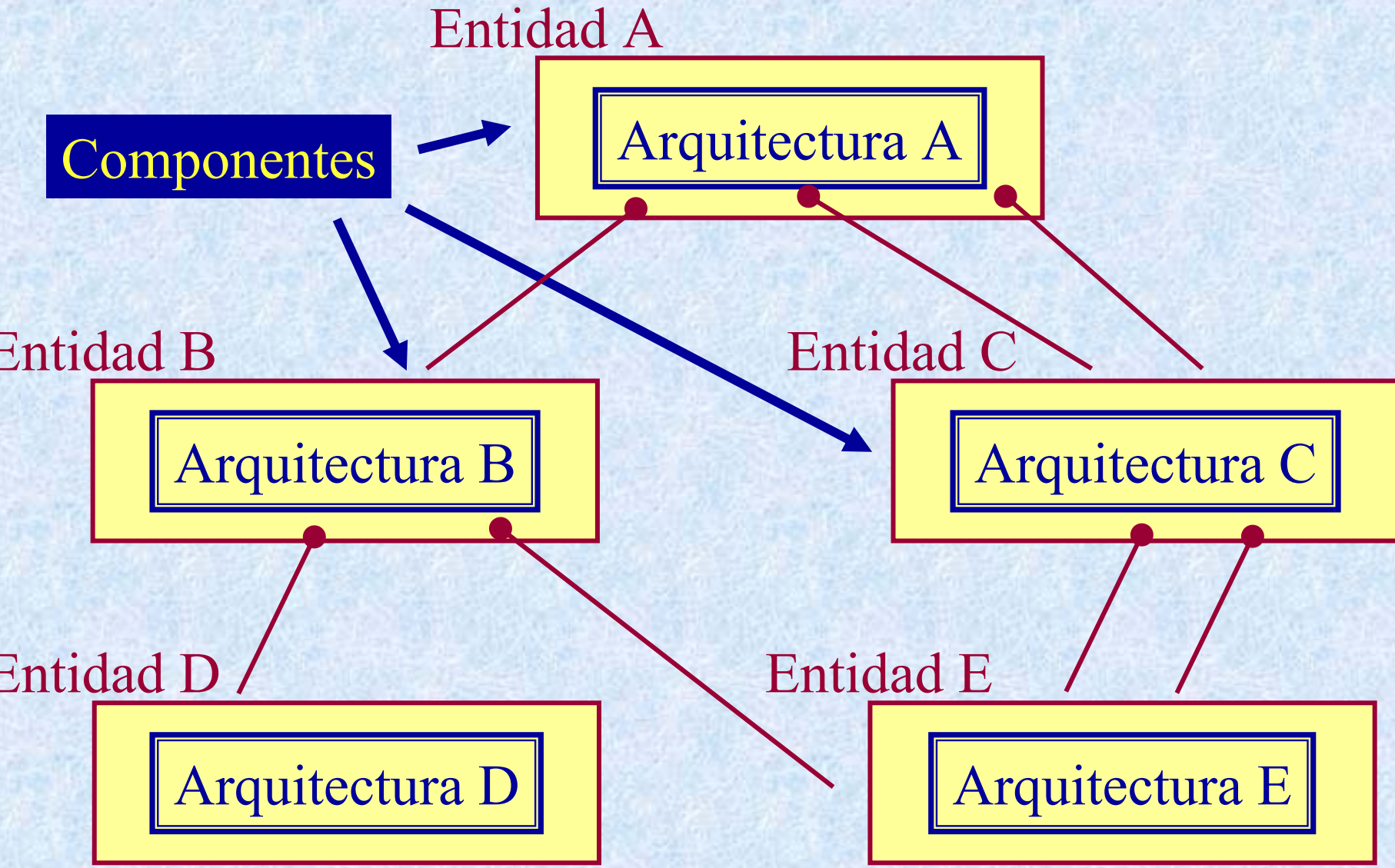
VHDL: Modelo de estructura

Programa VHDL

Se pueden definir múltiples arquitecturas para una entidad.

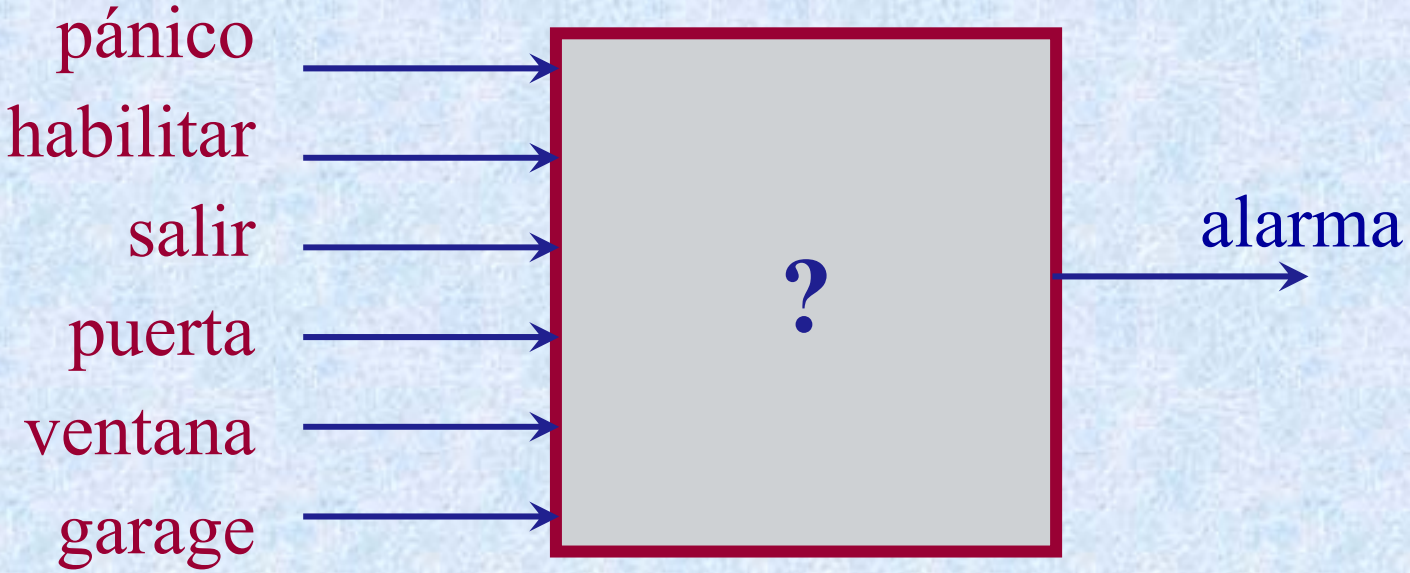
Una entidad de nivel superior puede instanciar múltiples entidades de niveles inferiores.

VHDL: Modelo de estructura, uso jerárquico



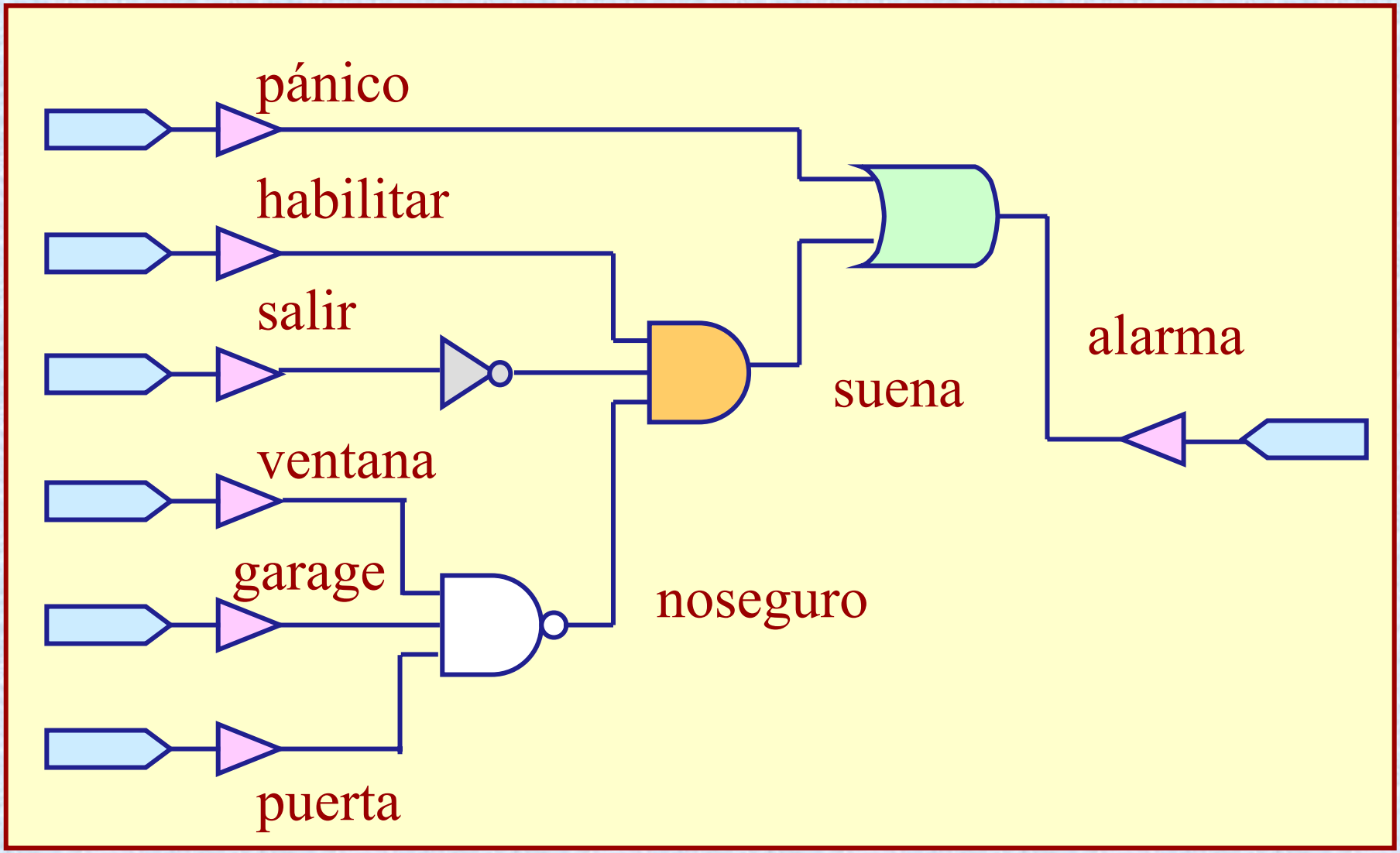
- ***Modelado VHDL de circuitos combinacionales.***
- ***Estilos de descripción VHDL***

Problema Alarma:



La salida alarma es ‘1’ si pánico es ‘1’, o si habilitar es ‘1’,salir es ‘0’ y la casa no es segura. La casa es segura si puerta, ventana y garage son ‘1’.

Modelado del circuito con esquemas:



Declaración de Entidad para Alarma.

```
--The IEEE standard 1164 package
library IEEE;
use IEEE.std_logic_1164.all;
entity alarma1 is
   port (panico, puerta,ventana,garage:in std_logic;
         salir,habilitar:in std_logic;
         alarma: out std_logic
        );
end alarma1;
```

Componentes básicos: Declaración de entidad

Define el módulo

```
entity   nombre_entidad  is
    [ generic  (lista propiedades);]
    [ port  (nombre_señal : modo  tipo_señal );
      port  (nombre_señal : modo  tipo_señal) ;
       .............
      port  (nombre_señal : modo  tipo_señal ) ;]
     [ declaraciones ]
[ begin
         sentencias]
end  [ nombre_entidad] ;
```

Componentes básicos: Declaración de entidad

Entidad mínima

entity nombre_entidad ***is***
end ;

generic

Permite utilizar parámetros en la definición de la entidad

Al instanciar el componente se establece el valor

Ejemplo: cantidad de bits de un sumador

Componentes básicos: Declaración de entidad

Define las entradas y salidas

port (nombre_señal : modo tipo_señal) ;

Modo → Define cómo se lo utiliza

- **in:** Entrada, se lee. No se le puede asignar valor.
- **out:** Salida, se le asigna valor. No se puede leer.
- **inout:** Entrada / Salida. CUIDADO !!
- **buffer:** Salida / Puede ser usado en el sistema (leído)

Tipo → Define los valores y operaciones legales

Componentes básicos:Declaración de arquitectura

```
architecture nombre_arquitectura of entidad is
    [declaraciones]
    begin
        [sentencias estructurales
              o concurrentes]
end [nombre_arquitectura];
```

Define el comportamiento del módulo

Se modela HARDWARE

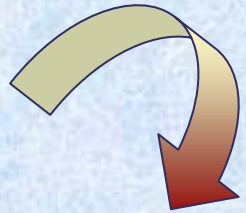
Las sentencias concurrentes se ejecutan en forma "simultánea"

La arquitectura puede modelarse en tres estilos

- Estilo estructural
- Estilo flujo de datos
- Estilo algorítmico

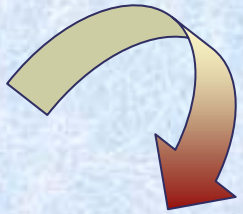
Estilos de descripción VHDL:

- **Descripción Estructural**



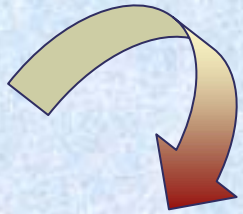
Lista de componentes interconectados

- **Descripción de Flujo de datos**



interconexión entre objetos del lenguaje

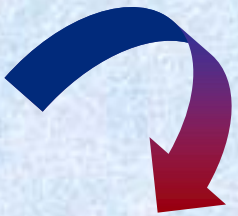
- **Descripción Algorítmica**



Descripción comportamental p. p. dicha

Diseño Estructural:

Describe la estructura del circuito como interconexión entre componentes



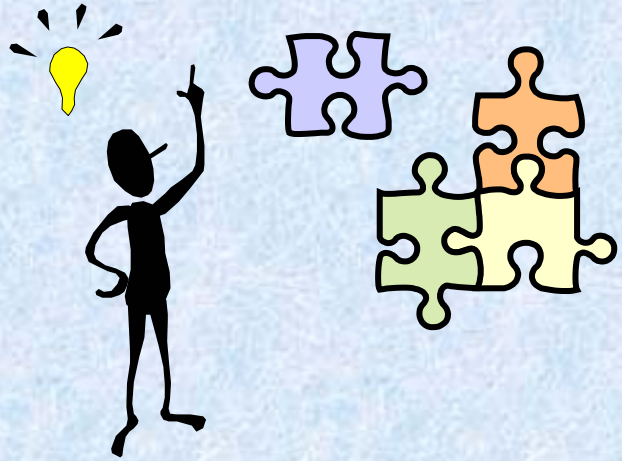
Descripción similar a un NETLIST
(lenguaje de descripción de estructura)

La descripción de tipo NETLIST se corresponde en forma directa con su implementación en hardware ***cuando se utilizan primitivas***.

Síntesis directa de la descripción

Diseño Estructural:

Diseño jerárquico



El estilo estructural, también se utiliza para interconectar módulos, subsistemas o IP cores

Divide y vencerás

En este caso representa una vista de alto nivel del sistema

Estilo Estructural

Entidad

Declaraciones

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
entity alarma1 is
  port (   panico, puerta,ventana,garage,salir,habilitar:in std_logic;
           alarma: out std_logic
           );
end alarma1;
architecture alarm_est of alarma1 is
signal seguro: std_logic;
signal noseguro: std_logic;
signal nosalir: std_logic;
signal suena: std_logic;
component INV
  port (I:in std_logic; O:out std_logic) ;
end component ;
component AND3
  port (I0,I1,I2:in std_logic; O:out std_logic) ;
end component ;
component OR2
  port (I0,I1:in std_logic; O:out std_logic) ;
end component ;
```

Las conexiones internas se deben modelar con señales

Modelado VHDL

```
begin
U1: INV
  port map (
            salir,nosalir
              );
U2: AND3
  port map (
            garage,ventana,puerta,seguro
              );
U3: INV
  port map (
            seguro,noseguro
              );
U4: AND3
  port map (
            habilitar,nosalir,noseguro,suena
              );
U5: OR2
  port map (
            panico,suena,alarma
              );
end alarm_est;
```

Instanciación del componente

Modelado VHDL

Las sentencias expresan la interconexión de los componentes declarados

Estilo Netlist

Estilo Estructural (continuación)

Biblioteca UNISIM:

- Contiene modelos funcionales para simulación.
- Incluye todos las primitivas descriptas en las Bibliotecas Unificadas de Xilinx.

En el modelo se debe respetar la interfaz definida

Nombre del componente → **CONVENCION**

Cantidad de puertos correctamente nombrados

C:\Xilinx\doc\usenglish\books\docs\lib\lib.pdf

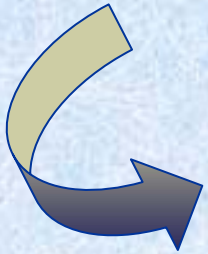
Elementos sintácticos para descripción estructural.

Elementos de diseño Estructural:

Declaración de componente:

```
component INV
    port (I:in std_logic; O:out std_logic) ;
end component ;
```

Los componentes utilizados deben estar definidos



- *Previamente en el diseño*
- *Bibliotecas del entorno*
- *Paquetes o bibliotecas adicionales*

Elementos de diseño Estructural:

Sentencia component:

Nombre de una entidad ya definida que se usará en la arquitectura.

```
U1: INV
    port map ( salir,nosalir );
```

Palabras clave que asocian una lista de los ports de la entidad con señales en la arquitectura.

Diseño de Flujo de Datos:

Describe el flujo de datos entre módulos usando ecuaciones de transferencia concurrentes.

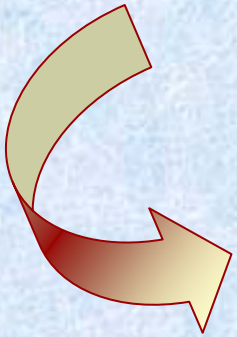
Descripción a nivel transferencia de registros (RTL)

Las estructuras a nivel RTL tienen una correspondencia muy estrecha con su implementación en hardware

Síntesis bastante directa de la descripción

Diseño de Flujo de Datos:

Es un estilo de descripción a mitad de camino entre una descripción estructural y una completamente abstracta



- Describe interconexiones entre objetos del lenguaje
- Permite estructuras de tipo condicional típicas de una descripción más abstracta

Todas las sentencias de la descripción son concurrentes.

Estilo
Flujo de Datos

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
entity alarma1 is
  port (
        panico, puerta,ventana,garage,salir,habilitar:in std_logic;
        alarma: out std_logic
        );
end alarma1;

architecture arch_flu of alarma1 is
signal seguro: std_logic;
signal noseguro: std_logic;
signal nosalir: std_logic;
signal suena: std_logic;
begin
nosalir <= not salir;
seguro <= puerta and garage and ventana;
noseguro <= not seguro;
suena <= habilitar and nosalir and noseguro;
alarma <= suena or panico;
end arch_flu;
```

Declaraciones

Sentencias concurrentes

Modelado VHDL

Elementos sintácticos para descripción de Flujo de Datos.

Elementos de diseño de Flujo de Datos:

Sentencias concurrentes que se disparan cuando cambia el valor de alguno de los argumentos de las asignaciones.

Asignación concurrente de señal:

```
nosalir <= not salir;
```

Asignación condicional: WHEN ... ELSE

```
architecture mux_flu of mux is
begin
        salida <= a when selec = '0' else b;
end mux_flu;
```

Elementos de diseño de Flujo de Datos:

Asignación con selección: WITH...SELECT...WHEN

```
with estado select
        semaforo <= rojo        when "01",
                    verde       when "10",
                    amarillo    when "11",
                    nofunciona  when others;
```

Expresiones con valores indefinidos → ✓ *Sintaxis incorrecta*

Las asignaciones con selección deben cubrir todos los posibles valores de la expresión

Diseño algorítmico:

Es una descripción de funcionalidad con un alto nivel de abstracción.

No es suficiente asignar valores a señales.

Se necesitan instrucciones más complejas para:

- Realizar cálculos.
- Acumular resultados intermedios.
- Repetir una secuencia de operaciones.

Diseño algorítmico:

Se necesitan instrucciones de ejecución secuencial similares a los lenguajes software de alto nivel

Sentencias concurrentes bastante particulares.

Describen comportamiento en forma secuencial pero su ejecución conjunta es concurrente

Introducen variables locales para soportar los cálculos.

Diseño algorítmico:

PROCESOS → **Instrucciones secuenciales.**

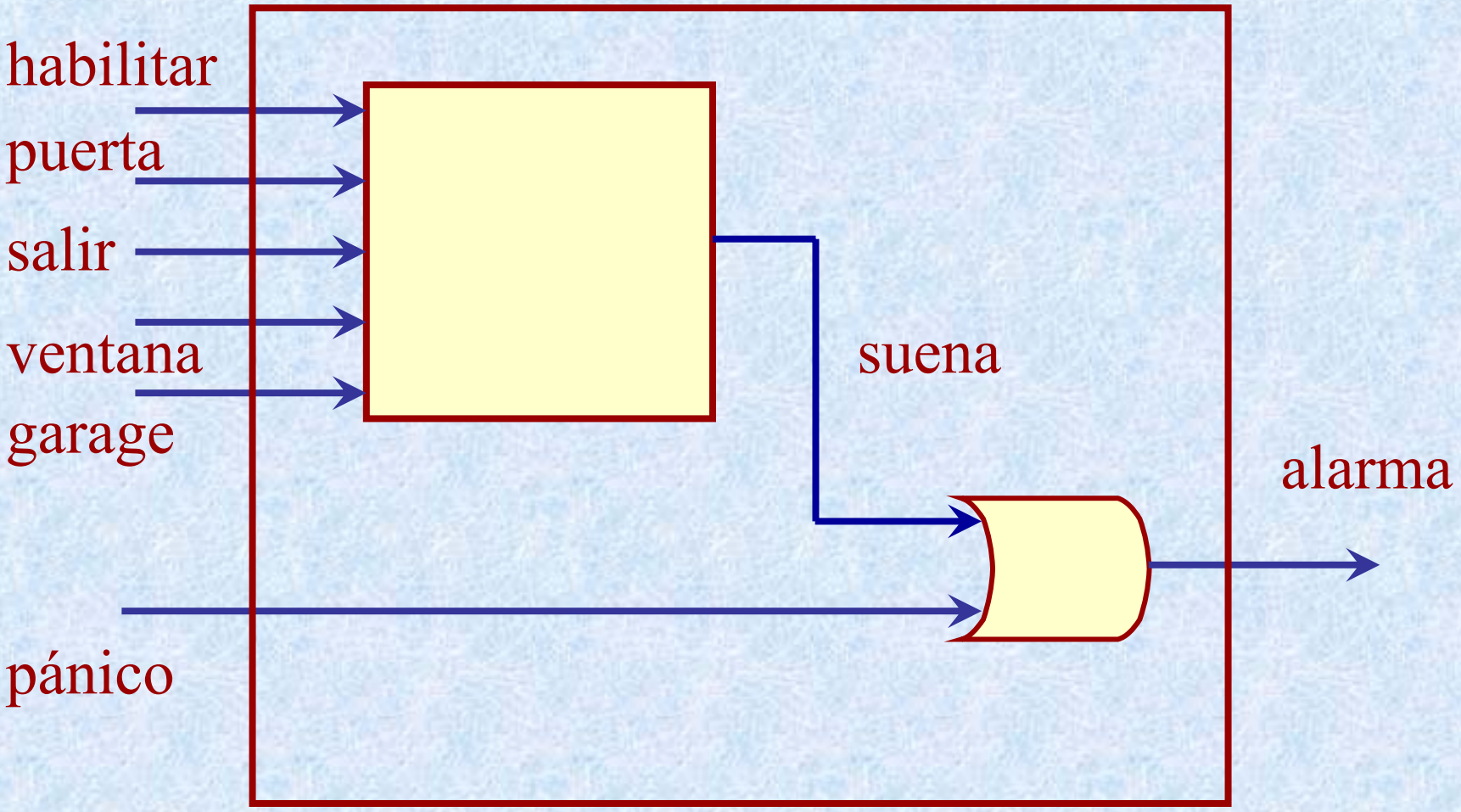
Sensibles a

SEÑALES → **Sincronizan procesos entre sí.**

El significado en hardware es más difuso, las instrucciones se ejecutan cuando cambian ciertos argumentos.

Síntesis mucho más complicada

Partición propuesta para modelado:

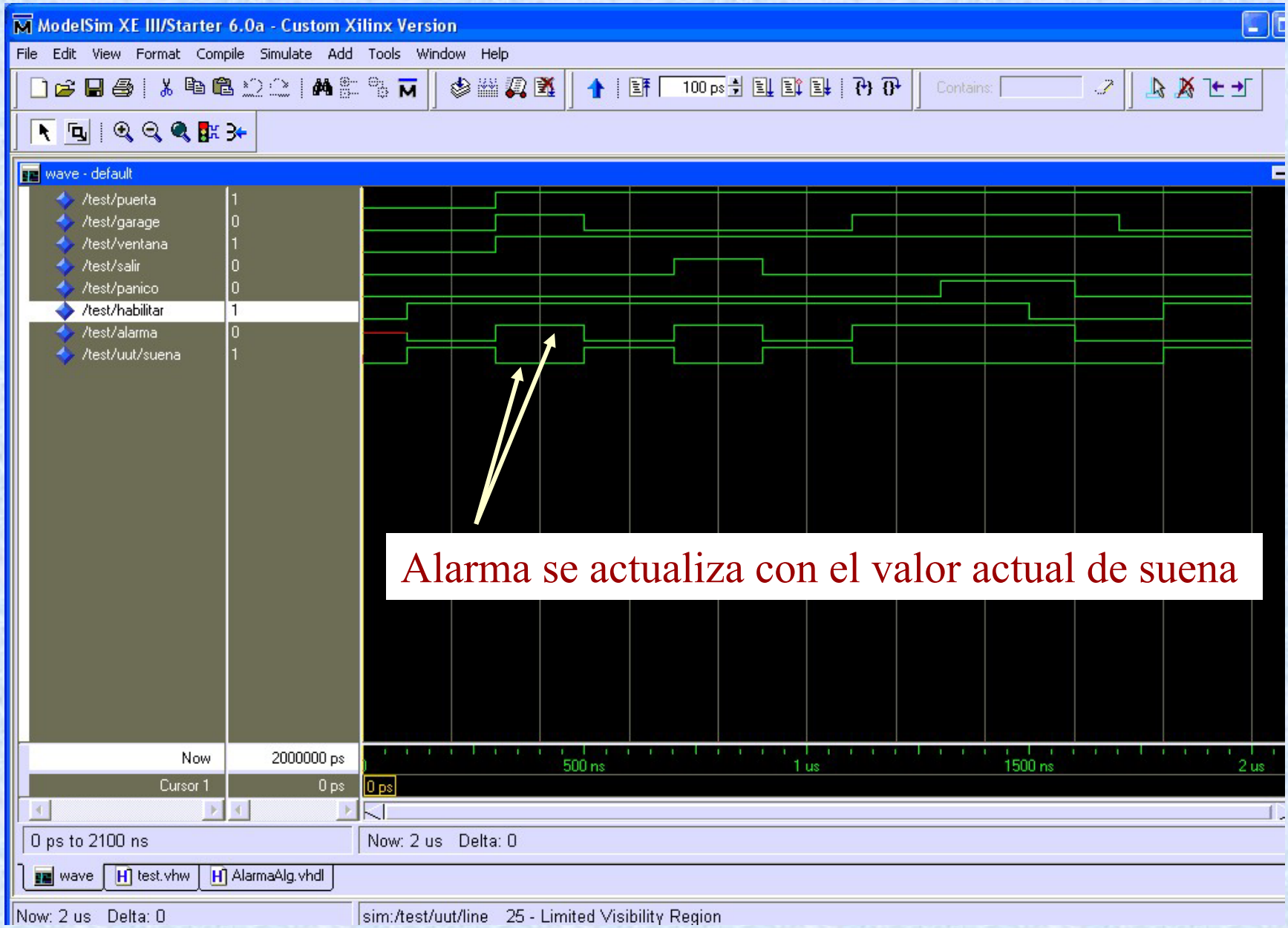


Procesos: El problema de la concurrencia:

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
entity alarma1 is
   port (  panico,puerta,ventana,garage,salir,habilitar:in std_logic;
           alarma: out std_logic );
end alarma1;
architecture arch_alarAlg of alarma1 is
signal suena: std_logic;
begin
process (panico, puerta,ventana,garage,salir,habilitar)
begin
suena <= ( not (ventana and puerta and garage)) and (not salir) and
habilitar;
alarma <= panico or suena;
end process;
end arch_alarAlg;
```

Lista de sensibilidad

Señales relacionadas


ModelSim XE III/Starter 6.0a - Custom Xilinx Version
File Edit View Format Compile Simulate Add Tools Window Help
wave - default
/test/puerta 1
/test/garage 0
/test/ventana 1
/test/salir 0
/test/panico 0
/test/habilitar 1
/test/alarma 0
/test/uut/suena 1
Now 2000000 ps
Cursor 1 0 ps
500 ns
1 us
1500 ns
2 us
0 ps
0 ps to 2100 ns
Now: 2 us Delta: 0
wave
test.vhw
AlarmaAlg.vhdl
Now: 2 us Delta: 0
sim:/test/uut/line 25 - Limited Visibility Region
Alarma se actualiza con el valor actual de suena

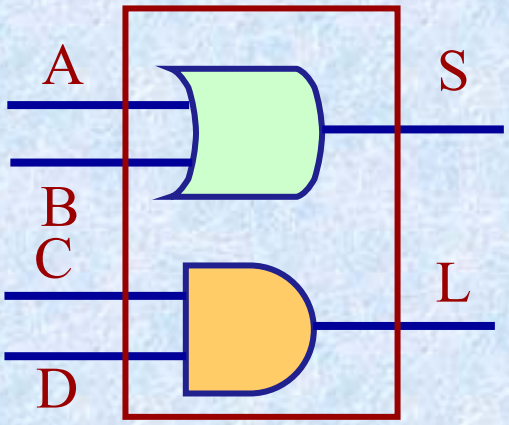
↳ *VHDL: Modelo de concurrencia / tiempo*

VHDL: Modelado y simulación de la concurrencia

Se modela ***HARDWARE*** usando ***SOFTWARE***

Concurrente

Secuencial



Las puertas deben funcionar al mismo tiempo !!

No se puede ***modelar*** y ***simular*** en un lenguaje convencional

VHDL: El problema de la concurrencia

Se utilizan sentencias "concurrentes" para describir el comportamiento del circuito

Estas sentencias son sensibles a señales (A, B, C y D)

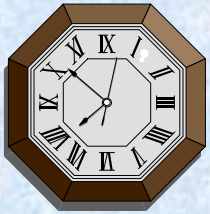
S <= A or B;
Q <= C and D;

Los cambios en las señales "fuente" (eventos), disparan la ejecución de las sentencias concurrentes.

Hay un soporte explícito de la concurrencia en simulación.

SIMULACIÓN

Dirigida por eventos



Interactúan para soportar concurrencia en la simulación

Dos escalas de tiempo

Macroescala
ΔT

Mide el tiempo real, transcurre al finalizar un ciclo de simulación.

Microescala
δ

Pasos de simulación que se ejecutan sin incrementos reales de tiempo

Los eventos que dirigen la simulación son cambios en las señales fuente de las sentencias concurrentes.

Las señales "fuente" son:

- Señales en la parte derecha de una asignación concurrente a señal
- Señales en la lista de sensibilidad de un proceso.

Asignación a señal → ***Se crea un "driver" que se asocia a una cola de eventos futuros.***

CUIDADO → El driver se actualiza sólo si el simulador detecta un evento.

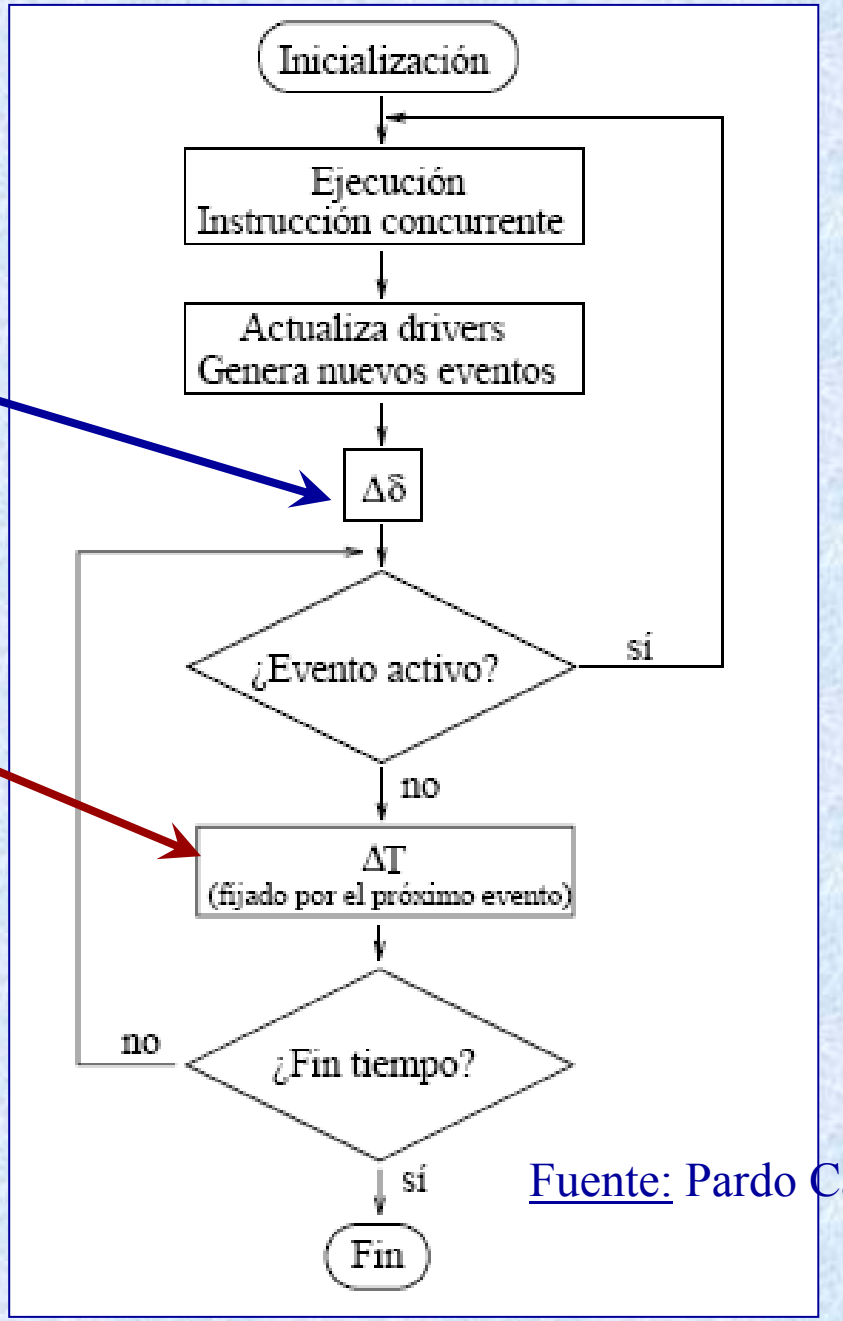
Se incrementa el tiempo real sólo cuando no hay más eventos activos.

VHDL: Simulación dirigida por eventos.

Paso de simulación

Ciclo de simulación

ΔT será el siguiente cambio en las señales fuente



Fuente: Pardo Carpio

Procesos: El problema de la concurrencia:

```
--The IEEE standard 1164 package, declares std_logic, rising_edge(), etc.
library IEEE;
use IEEE.std_logic_1164.all;
entity alarma1 is
   port (  panico, puerta, ventana, garage, salir, habilitar: in std_logic;
           alarma: out std_logic  );
end alarma1;
architecture arch_alarAlg of alarma1 is
begin
process (panico, puerta,ventana,garage,salir,habilitar)
variable suena: std_logic;
begin
suena := ( not (ventana and puerta and garage)) and (not salir) and
habilitar;
alarma <= panico or suena;
end process;
end arch_alarAlg;
```

Uso de variables !!

Variables:

No tienen significado físico

Deben ser declaradas dentro del proceso y son locales a él.

Su asignación es inmediata pero secuencial, al estar en distintas sentencias en el proceso.

Deben tener un valor asignado antes de utilizarlas.

El orden de las sentencias es importante

```
a := b or c;
d := e and a;
```

Variables vs. Señales

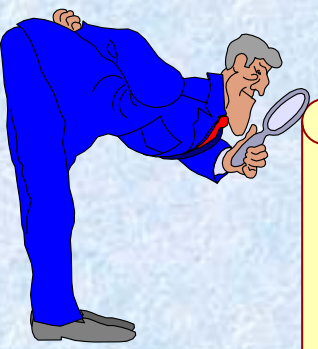
	Señales	***Variables***
Sintaxis	destino<= fuente	destino:= fuente
Utilidad	Modelan nodos físicos del circuito	Representan almacenamiento local
Visibilidad	Global (comunicación entre procesos)	Local (dentro del proceso)
Comportamiento	Se actualizan al avanzar el tiempo (suspensión proceso)	Se actualizan inmediatamente

Concurrencia y procesos: Conclusiones

- Las señales modelan hilos del circuito. Tienen un valor actual y uno/s futuro/s, y su actualización está ligada a drivers de eventos.

- Las sentencias se ejecutan en δ delay mientras hay drivers activos

- El tiempo real avanza cuando se suspenden todas las sentencias concurrentes.



- Usar variables para valores intermedios.
- Asignar señales relacionadas en procesos separados (u otra sentencia concurrente).

Elementos sintácticos para descripción algorítmica.

Elementos de diseño algorítmico:

El bloque PROCESS

Compuesto por un conjunto de instrucciones secuenciale

Permite estructuras más abstractas como la ejecución de bucles

En un programa puede haber varios bloques PROCESS

Cada bloque PROCESS es una instrucción concurrente

Elementos de diseño algorítmico:

¿Cómo se activa el bloque PROCESS?

1. ***La forma habitual es con una lista de sensibilidad***

2. ***Existe la posibilidad de usar un WAIT explícito en algún lugar del bloque***

Elementos de diseño algorítmico:

Activación habitual del bloque PROCESS

process (panico, puerta,ventana,garage,salir,habilitar)

Lista de sensibilidad

Lista de señales

Cada vez que cambia alguna de las señales se ejecuta el bloque

Elementos de diseño algorítmico:

Activación del bloque PROCESS con wait

wait on lista-sensible until condición for timeout

Los tres parámetros son optativos.

El que se verifique antes continuará la ejecución del bloque

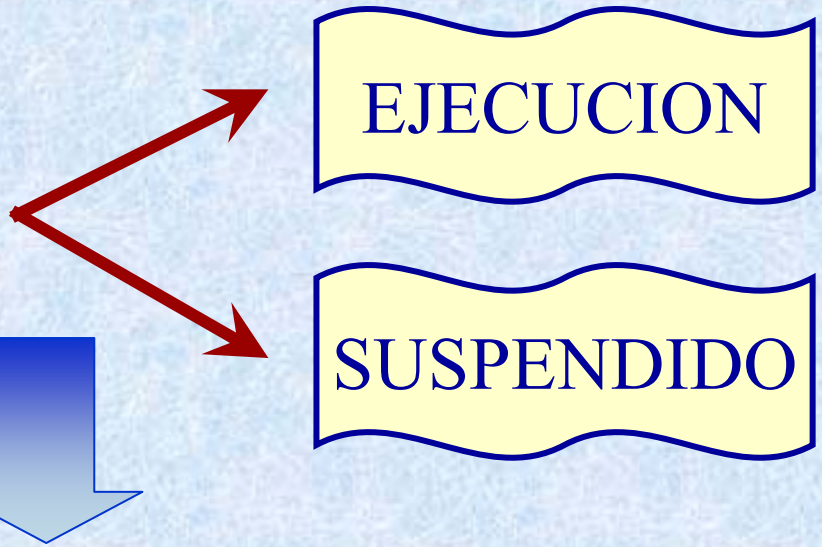
Si el bloque tiene una lista sensible no puede tener wait.

wait on pulso;
wait until contador > 6;
wait on interrupción for 25 ns;
wait on reloj, sensor until contador > 5 for 50 ns;

Ejemplos

Elementos de diseño algorítmico:

El bloque PROCESS tiene dos estados posibles



El proceso debe tener una lista sensible o un wait, en caso contrario se ejecuta en forma indefinida.

En simulación no se podría salir nunca del lazo del process.

Elementos de diseño algorítmico:

Sentencia condicional IF...THEN...ELSE

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

Esta estructura puede anidarse.

elsif y else son optativas. Conviene contemplar todos los posibles casos

Problema Memoria implícita

Elementos de diseño algorítmico:

Sentencia condicional IF...THEN...ELSE

```
-- Ejecución secuencial
process ( a, b, c )
begin
if a > b then
p <= 2;
elsif a > c then
p <= 3;
elsif (a = c and c = b) then
p <= 4
else p <= 5;
end if ;
end process ;
```

```
-- Ejecución concurrente
p < = 2 when a > b else
      3 when a > c else
      4 when (a = c and c = b) else
      5 ;
```

En asignaciones tiene su equivalente concurrente.

La lista sensible debe incluir todas las señales cuyo cambio es importante.

Elementos de diseño algorítmico:

Sentencia de selección CASE

```
case expresión is
    when caso1 = >
        instrucciones
    when caso2 = >
        instrucciones
          ● ● ●
    when others = >
        instrucciones
end case ;
```

```
-- Ejemplo
case nota is
    when 10 = > acta <= “SOB”;
    when 8 to 9 = > acta <= “DIS”;
    when 6 to 7 => acta <= “BUE”;
    when 4 | 5 => acta <= “APR”;
when others = > acta <= “INS”;
end case ;
```

- *La expresión de selección debe ser discreta.*
- *No puede haber casos duplicados.*
- *Se deben cubrir todas las opciones de selección.*

Elementos de diseño algorítmico:

Sentencia de bucles WHILE Y FOR

```
Bucle_id:
while condición loop
        instrucciones
end loop;
```

```
-- Lazo while
cuenta := 5;
while cuenta >= 0  loop
   tabla (cuenta) <= cuenta * 2;
   cuenta := cuenta - 1;
end loop;
```

Entero o enumerado

```
bucle_id:
for identificador in rango loop
        instrucciones
end loop;
```

```
-- Lazo for
for cuenta in 5 downto 0 loop
   tabla (cuenta) <= cuenta * 2;
end loop;
```

Elementos de diseño algorítmico:

Interrupción de los bucles

next bucle_id when condición ; → ***Detiene una iteración de bucle y pasa a la siguient***

exit bucle_id when condición ; → ***Detiene la ejecución y sale del bucle***

Interrumpe el for y sigue en el while

```
fuera:
while a < 10 loop
   -- varias sentencias
   dentro:
   for i in 0 to 10 loop;
      -- varias sentencias
      next fuera when i = a;
   end loop dentro;
end loop fuera;
```

Sentencia configure:

Cuando un diseño incluye varias arquitecturas posibles para la misma entidad

Se debe indicar al compilador con cuál de ellas se desea trabajar.

Si no se indica la arquitectura objetivo el compilador toma alguna decisión por defecto

Sentencia configure: Sintaxis

```
entity Mux2 is
    Port ( a : in bit;
           b : in bit;
           ctrl : in bit;
           z : out bit );
end Mux2;
-- Definición arquitectura Algoritmica para la entidad mux2
-- Definición arquitectura Flujo para la misma entidad

-- Uso de sentencia configure
configuration confmux2 of mux2 is
for Algoritmica
end for;
end;
```

Se liga la arquitectura Algorítmica con la entidad mux2

↳ *Modelado VHDL de circuitos secuenciales.*

Para la descripción se utilizan los recursos ya analizados.

Se describen circuitos temporizados

Es necesario reconocer un flanco activo de reloj.

Atributo event

clk'event and clk = '1'

Expresión reconocida por XST para describir un flanco ascendente de reloj

Se depende de una herramienta de síntesis

Ejemplos de código en ISE para FF:

```
process (<clock>)
begin
if <clock>'event and <clock>='1'
then   <output> <= <input> ;
end if;
end process ;
```

La señal output, no siempre tiene valor definido. Por lo tanto se almacena.

Memoria implícita

```
process (<clock>, <reset>)
begin
if <reset>='1'  then
 <output> <= '0' ;
elsif (<clock>'event and <clock>='1') then
<output> <= <input>;
 end if;
end process ;
```

Memoria implícita:

Causa: Las señales tienen valor actual y futuro

Consecuencia: Si en una sentencia concurrente no se conoce el valor futuro, se mantiene el actual.

Se sintetiza un elemento de memoria

VENTAJAS → Simplifica la creación de memoria.

CONTRAS → Puede crearse memoria no deseada

Memoria implícita: Ejemplos

El latcheado puede ser correcto o no !!

```
process (tempObj, tempActual)
begin
if tempActual < tempObj – 2 then
encendido <= true;
elsif  tempActual > tempObj + 2 then
encendido <= false;
end if;
end process;
```

No incluye ELSE

```
process (a)
begin
case  a  is
when "00" =>  o <= '1';
when "01" =>  o <= '1';
when "10" =>  o <= '0';
when others => null ;
end case;
end process ;
```

Valor indefinido

Memoria implícita: Formas de evitarla

Las instrucciones IF.. THEN.. ELSE, deben incluir ELSE

Se deben especificar todas las alternativas de un CASE.

Cláusula WHEN OTHERS …

Ejemplo de código para un registro de desplazamiento

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
--Registro con reset sincrónico load paralelo y corrimiento
--lógico a la izquierda, con prioridades en ese orden.
-- Se dejo la salida paralela (bout) para claridad en
--simulación, corresponde mantener sólo la salida s.
entity regis is
   port (
      din: in STD_LOGIC_VECTOR (3 downto 0);
      clk: in STD_LOGIC;
      l: in STD_LOGIC;
      r: in STD_LOGIC;
      shl: in STD_LOGIC;
      s: out STD_LOGIC;
      bout: out std_logic_vector (3 downto 0)
   );
end regis;
```

Registro de desplazamiento: Continuación

```
architecture register_arch of regis is
begin
process (clk)
variable dout : std_logic_vector (3 downto 0);
variable aux: std_logic_vector (2 downto 0);
begin
aux := (r,l,shl);
if ( clk'event and clk = '1') then
case conv_integer (aux) is
when 4 | 5 | 6 | 7 => dout := ('0','0','0','0');
when 2 | 3 => dout := din;
when 1 => dout :=  dout (2 downto 0) & '0';
when others => null;
end case;
end if;
s <= dout(3);
bout <= dout;
end process;
end register_arch;
```

Todas las señales de control son sincrónicas

Flanco ascendente del reloj.

Operador de concatenación

Se almacena !!

No están interrelacionadas !!

Consideraciones para diseños sincrónicos:

- Es aconsejable registrar las señales asincrónicas cuando ingresan al sistema.
- No colocar puertas en el reloj, utilizar señal de enable.
- Utilizar un único reloj por proceso.
- Si todas las señales de control son sincrónicas, basta el clk en la lista de sensibilidad. Agregar el reset asincrónico.