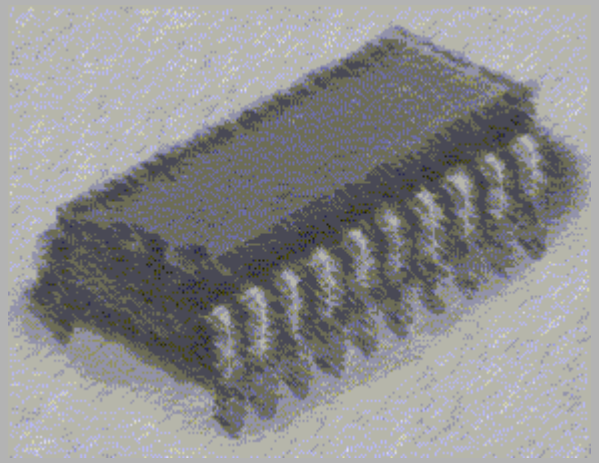
VHDL



Laboratorio de Arquitectura de Computadores

I. T. Informática de Gestión
I. T. Informática de Sistemas

Curso 2006-2007

Índice

- 5. Paquetes
- 6. Bibliotecas
- 7 Sentencias secuenciales
- 8. Sentencias concurrentes
- 9. Sentencias estructurales
- 10. Elementos léxicos del lenguaje
- 11. Constantes, variables y señales
- 12. Datos
- 13. Atributos
- 14. Asignación de señales
- 15. Señales resueltas

1. Conceptos básicos

- Lenguaje orientado a la descripción y modelado de hardware.
- Gran capacidad para definir datos.
- Hereda utilidades (if, case, for, while,...)
- Estructuración del programa mediante funciones y procedimientos.
- Permite usos de bibliotecas.

2. Modelos de Hardware

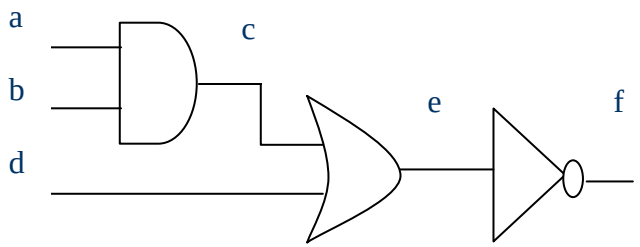
- 2.1 Modelo Estructura
  - Descripción:
    Interfaz con el exterior (**ENTITY**)
    Utiliza elementos ya definidos (**COMPONENT**)
  - Hace referencia a los distintos elementos, especificando las conexiones entre ellos.

2. Modelos de Hardware

- 2.2 Modelo Concurrencia (Procesos, señales y eventos).
  - "Pseudoconcurrencia"
  - **Process:** Describe comportamiento
    - El código que contiene es secuencial.
    - Los procesos se ejecutan concurrentemente.
    - Los procesos se comunican a través de **señales.**
    - Los procesos son sensibles a los **eventos**.

Ejemplo

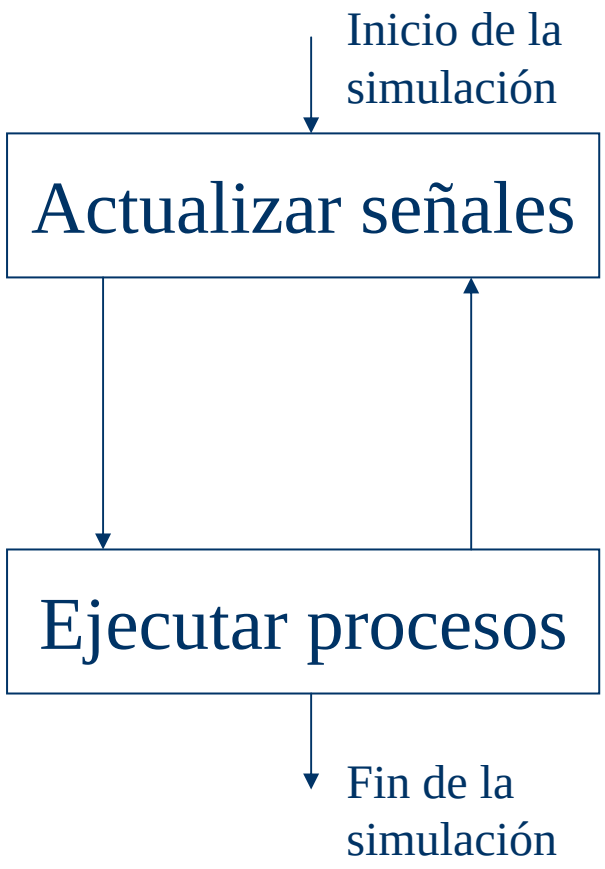
```
and2: process
begin
c<= a and b;
wait on a, b;
end process and2;

or2: process
begin
e<= c or d;
wait on c, d;
end process or2;

f <= not e;
```



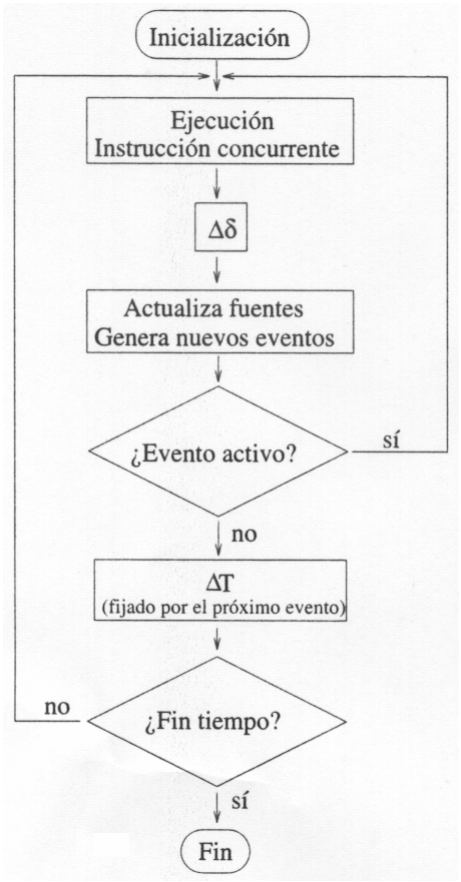
- Existe un **retardo** desde que el valor está en la cola de eventos hasta que la señal coge el valor.
- Cuando todos los eventos se hayan ejecutado y estén suspendidos se actualiza.

2. Modelos de Hardware

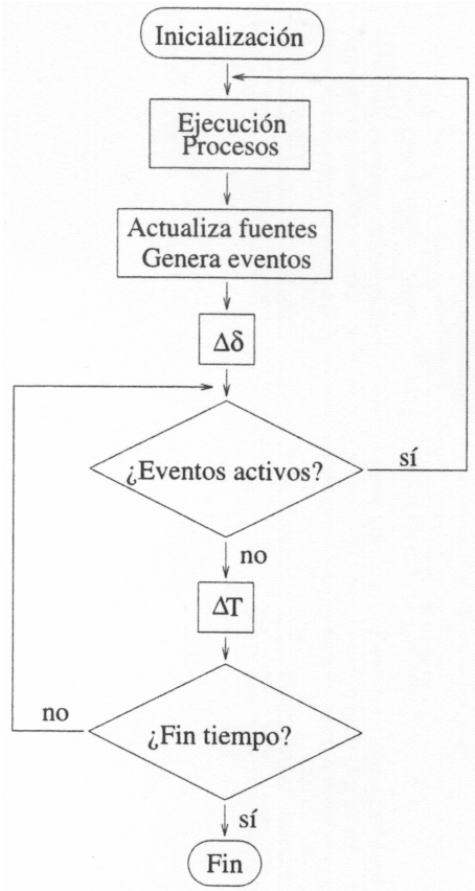
- 2.3 Modelo Temporal
  - Ciclo de simulación
  - La simulación de un modelo VHDL es una simulación dirigida por eventos.



Simulación por eventos



Simulación por incremento fijo



3. 1 La entidad (entity)

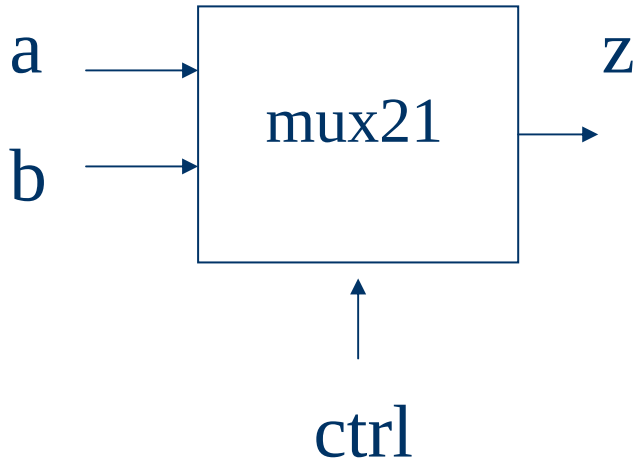
- Interfaz con el exterior.
- Define señales de entrada / salida.

- Sintaxis:

```
ENTITY nombre is
    PORT (señal-in: IN tipo; señal-out: OUT tipo ...);
END nombre;
```

Ejemplo (entidad Multiplexor)

```
ENTITY mux21 IS
    PORT (a:in bit; b,
    ctrl:in bit;  z: out bit);
END mux21;
```



3. 2 La Arquitectura (architecture)

- Definen la funcionalidad de la entidad que representa.
- Sintaxis:

```
ARCHITECTURE nombre-arq OF nombre-ent IS
(declaraciones)
BEGIN (sentencias concurrentes)
    PROCESS
            (declaraciones de funciones, constantes, variables, etc.)
            BEGIN (sentencias secuenciales)
    END  PROCESS;
END nombre-arq;
```

Ejemplo modelo de una and

```
ENTITY and2 IS
    PORT (a:in bit; b:in bit; z:out bit);
END and2;

ARCHITECTURE funcional OF and2 is
    BEGIN
        PROCESS (a, b)
            BEGIN
            IF a = '1' and b = '1' THEN z<='1';
            ELSE  z<='0';
        END PROCESS;
END funcional;
```

3.2.1 Modelos de arquitecturas

- 3.2.1.1 Estilo algorítmico (funcionalidad)
- 3.2.1.2 Estilo Flujo de datos (correspondencia hardware)
- 3.2.1.3 Estilo Estructural (componentes interconectados)

3.2.1.1 Estilo algorítmico (Ejemplo multiplexor)

```
ARCHITECTURE comport OF mux21 IS
BEGIN
    PROCESS (a, b, ctrl)
    BEGIN
        IF (ctrl='0') THEN
            z<=a;
        ELSE
            z<=b;
        END IF;
    END PROCESS;
END comport;
```

3.2.1.2 Estilo Flujo de datos (Ejemplo multiplexor)

```
ARCHITECTURE flujodatos OF mux21 IS

SIGNAL ctrl-n, n1, n2: bit;

BEGIN
    ctrl-n <= not (ctrl) after 1 ns;
    n1 <= ctrl-n and a after 2 ns;
    n2 <= ctrl and b after 2 ns;
    z <= n1 or n2 after 2 ns;

END flujodatos;
```

$$Z = \overline{ctrl} \cdot a + ctrl \cdot b$$

3.2.1.2 Estilo Estructural (Ejemplo multiplexor)

```
ARCHITECTURE estructural OF
    mux21 IS

COMPONENT inv
    PORT(y:in bit; z:out bit);
END COMPONENT;

COMPONENT and2
    PORT(x:in bit; y:in bit; z:out bit);
END COMPONENT;

COMPONENT or2
    PORT(x:in bit; y:in bit; z:out bit);
END COMPONENT;

-- declaracion de las señales
SIGNAL ctrl-n, n1, n2: bit;
```

```
--indicamos ubicación de la arquitectura de
    los componentes
FOR P1:inv USE ENTITY WORK.inv(func);
FOR P2:and2 USE ENTITY
    WORK.and2(func);
FOR P3:and2 USE ENTITY
    WORK.and2(func);
FOR P4:or2 USE ENTITY WORK.or2(func);

--conexiones
P1:inv PORT MAP (ctrl, ctrl-n);
P2:and2 PORT MAP (ctr-n, a, n1);
P3:and2 PORT MAP (ctrl, b, n2);
P4:or2 PORT MAP (n1, n2, z);

END estructural;
```

4.TEST-BENCH

- Es un programa en VHDL que tiene entidad vacía.
- Finalidad: Diagnosis de fallos y verificación.
- Son como los prog. Estructurales.
- Al inyectar señales mediante procesos, son programas comportamentales.

4.TEST-BENCH (Ejemplo)

```
ENTITY run-mux IS
END run-mux21;

ARCHITECTURE test-bench OF run-
    mux IS
COMPONENT mux21 IS
    PORT (a:in bit; b, ctrl:in bit;  z: out
    bit);
END mux21;

FOR multiplex: mux21 USE ENTITY
    WORK.mux21 (estructural);

BEGIN
-- proceso de instanciación
    multiplex: mux21 PORT MAP (a,
    b, ctrl, z);
```

```
--proceso de asignación de valores
    PROCESS
    BEGIN
        a<=0;
        b<=0;
        ctrl<=1;
        WAIT FOR 10 ns;
        a<=1;
        b<=1;
        ctrl<=1;
        WAIT FOR 10 ns;
        a<=0;
        b<=1;
        ctrl<=0;
        .....
    END PROCESS;
END test-bench;
```

5. Paquetes

- Agrupan un conjunto de declaraciones para permitir un uso múltiple:
  - Declaraciones de tipos, constantes, variables, subprogramas, procedimientos, etc.
- Constan de:
  - declaración, o parte visible;
  - cuerpo, o parte oculta en donde se realiza la implementación.

5. Paquetes

- Ejemplo de declaración del paquete:

```
PACKAGE retardos IS
 constant RetNot: TIME;
 constant RetAnd: TIME;
 constant RetOr:  TIME;
END retardos;
```

5. Paquetes

- Ejemplo de cuerpo del paquete:

```
PACKAGE BODY retardos IS
 constant RetNot: TIME := 2 ns;
 constant RetAnd: TIME := 3 ns;
 constant RetOr:  TIME := 1 ns;
END retardos;
```

6. Bibliotecas

- Almacenan el resultado del análisis y la compilación de las unidades de diseño.
- Al compilar el paquete y su cuerpo, van a parar a alguna biblioteca
- Para poder usarlos, hay que hacer visible la biblioteca y después llamar al paquete.
- Ejemplo:

```
 LIBRARY biblioteca;
USE biblioteca.retardos.RetAnd;
USE biblioteca.retardos.ALL;
```

Ejemplo de uso

- Ejemplo de uso de los valores archivados en la biblioteca:

```
LIBRARY biblioteca;
ARCHITECTURE rtl OF mux IS
 SIGNAL crtl_n, n1, n2: bit;
 BEGIN
  ctrl_n <= not (ctrl) AFTER
    biblioteca.retardos.RetNot;

  ....
END rtl;
```

Bibliotecas por defecto

- Las `work` y `std` son por defecto.
- Es como si siempre existieran estos dos comandos:

```
LIBRARY std;
LIBRARY work;
```

Otras bibliotecas

- Es importante la librería `ieee` que suelen traer todos los simuladores.
- Su paquete clave es el `std_logic_1164`, que contiene un sistema de nueve niveles lógicos de señal, definiciones de tipos junto con sus operaciones aritméticas y relacionales, y funciones de resolución.
- Invocación:

```
LIBRARY  ieee;
USE      ieee.std_logic_1164.ALL
```

7. Sentencias secuenciales

- [etiqueta:]
  wait [**on** señal {,...}]
  [**until** expresión_booleana]
  [**for** expresión_tiempo]

- [etiqueta:]
  nombre_señal **<=** [**transport**] valor
  {**after**
  expresión_tiempo}

7. Sentencias secuenciales

- [etiqueta:]
  nombre_variable := expresión

- **if** condición **then** sent_secuenciales
  {**elsif** condición **then**
  sent_secuenciales}
  [**else** sent_secuenciales]
  end if;

7. Sentencias secuenciales

- [etiqueta:]
  case expresión **is**
  when valor **=>**
  sent_secuenciales;
  {**when** valor **=>**
  sent_secuenciales;}
  end case;

7. Sentencias secuenciales

- [etiqueta:]
  [**while** condición | **for** repetición]
  loop
  sentencias secuenciales...
  end loop [etiqueta];

- [etiqueta:]
  exit [etiqueta_loop] [**when** condición]

7. Sentencias secuenciales

- [etiqueta:]
  next [etiqueta_loop] [**when** condición]

- [etiqueta:]
  assert expresión_booleana
  [**report** string]
  [expresión_severidad]
- expresión_severidad::=
  {note | warning | error | failure}

8. Sentencias concurrentes

- ```
  [etiqueta:]
  process [(nombre_señal {,...}] [is]
  [declaraciones]
  begin
   sentencias secuenciales
  end process [etiqueta];
  ```

- ```
  [etiqueta:]
  nombre_señal <= [transport] forma_onda
  ```

8. Sentencias concurrentes

- [etiqueta:]
  nombre_señal **<=** [**transport**]
  {forma_onda **when** expr_booleana
  else}
  forma_onda [**when** expr_booleana]

8. Sentencias concurrentes

- [etiqueta:] **with** expresión **select**
  nombre_señal **<=** [**transport**]
  {forma_onda **when** valor,}
  forma_onda **when** valor;

- [etiqueta:]
  assert expresión_booleana
  [**report** string]
  [expresión_severidad

8. Sentencias concurrentes

- etiqueta: **block** [expresión_guarda]
  [**is**]
  [**generic** (lista_genéricos)]
  [**generic map** (lista_asociación)]
  [**port** (lista_puertos)]
  [**port map** (lista_asociación)]
  {parte declarativa}
  begin
  {sentencias concurrentes}
  end block [etiqueta];

9. Sentencias estructurales

- **component** nombre [**is**]
  [**generic** (lista_genéricos)]
  [**port** (lista_puertos)]
  end component [nombre];

- etiqueta_referencia: nombre
  [**generic map** (lista_asociación)]
  [**port map** (lista_asociación)]
- etiqueta_generate:

{[**for** espec_for | **if** cond]} **generate**
  {sentencias concurrentes}
 end generate;

10. Elementos léxicos

- Identificadores
- Operadores y símbolos especiales
- Literales

Identificadores

- No hay longitud máxima
- No hay diferencia entre mayúsculas y minúsculas.
- Deben empezar por carácter alfabético
- No pueden terminar con subrayado ni tener dos subrayados seguidos.
- No se permiten palabras reservadas.

Operadores y símbolos especiales

- Aritméticos: +, -, /, *
- Lógicos: not, and, or, nand, nor, xor
- Relación: =, /=, <=, >=, <, >,
- Concatenación: &
- Otros: (,), :, ;
- Los comentarios comienzan por - -
- Las sentencias terminan con ;

Literales

- Enteros
- Pto. Flotante: ej. 0.423....
- Literales físicos: ej ns
- Bases 2#1011#, 8#17#, 16#9FC#
- Caracteres ascii: ‘M’, ‘5’
- Cadenas de caracteres: “esto es una cadena”

11. Constantes, variables y señales

- Constantes:

  Ej. Constant RetardoAnd2: time:=2ns;

- Variables: Locales en un proceso

  Ej. variable resultado:integer:= 0;

- Señales: Se modifican mediante sentencias de asignación pero no se hace efectiva hasta que todos los procesos terminan.

  Ej. Signal reloj: bit := 0;

12. Datos

- VHDL tiene tipos predefinidos y permite declarar nuevos tipos.
  - Ej. Type semana is (lunes, martes, ...);

12. Datos

- En el paquete estandar:
  - Type bit is (‘0’, ‘1’);
  - Type boolean is (false, true);
  - Type time is range 0 to 1E20
    Units
    Fs;
    Ps = 100 fs;
    .
    End units;
- En el paquete IEEE std_logic_1164:
  - Type std_ulogic is (‘u’, ‘x’, ‘0’,’1’,’z’,’w’,’l’,’h’,’-’)

12. Datos

- Datos compuestos:
  - **Vectores**: Objetos del mismo tipo ordenados por indices:

    Ej. Type byte2 is array (7 down to 0) of bit;
  - **Registros**
    - Ej. Type fecha is record

      Dia: integer range 1 to 31;

      Mes: integer range 1 to 12;

      Año: integer range 1900 to 2025;

      End record;

      Variable hoy, ayer: fecha

13. Atributos

- Son características que se pueden asociar a cualquier elemento vhdl.
  - Algunos ejemplos

A'range(n)	Rango del índice n de A
A'low(n)	Valor mínimo de A
A'high(n)	Valor máximo de A
A'left(n)	Valor izq. de A
A'right(n)	Valor dcho. de A

12. Atributos (Ejemplos)

Ej.

- Señales:

  If(Reloj='1' and Reloj'event);

- Variables:

  Type matriz is array(3 downto 0)

  Variable M:matriz;

  M'LEFT=3;

  M'LEFT(2)=5;

  ...

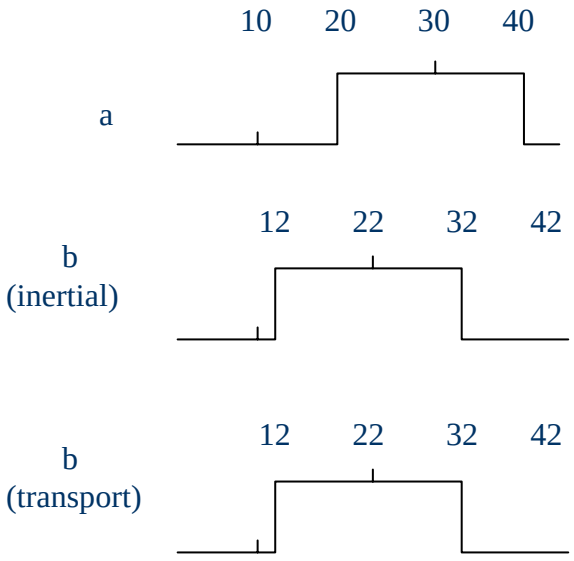
14. Asignación de señales

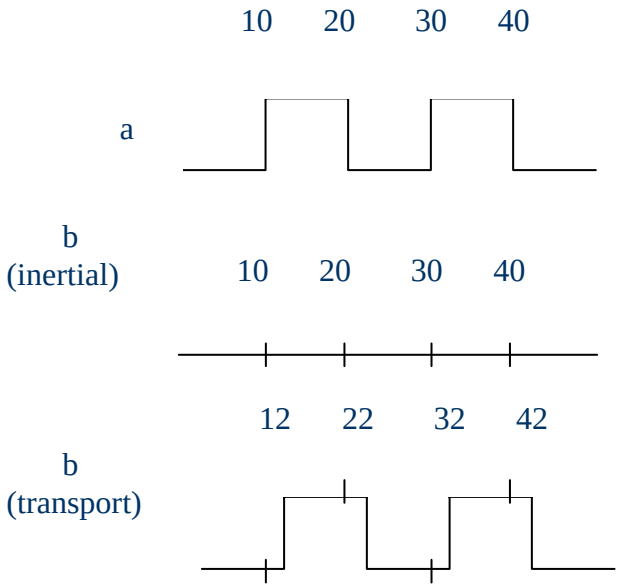
- El valor pasa a una cola de eventos y se actualiza al terminar el proceso.
- Existen dos tipos de retardos:
  - **Transport** (filtra los cambios inferiores a un mínimo)
  - **Inertial** (Propaga los cambios)
- Por defecto las asignaciones son inertial
- Si queremos que sea transport se debe especificar.

14. Asignación de señales (ejemplos)

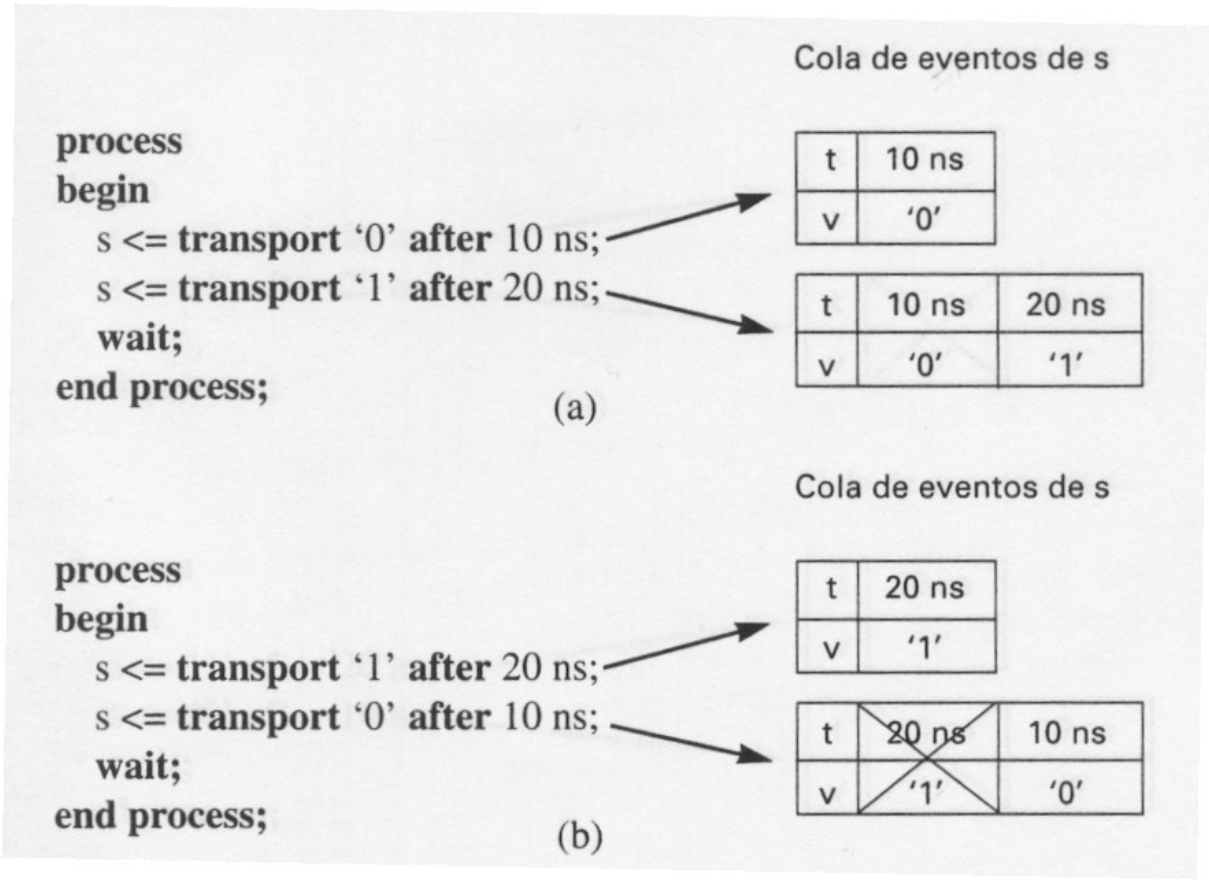
```
Entity inversor is
   port(a:in bit; b:out bit);
Architecture rtl of inversor is
begin
  b<= transport not a after 12 ns;
end rtl;
```

14. Asignación de señales (ejemplos)

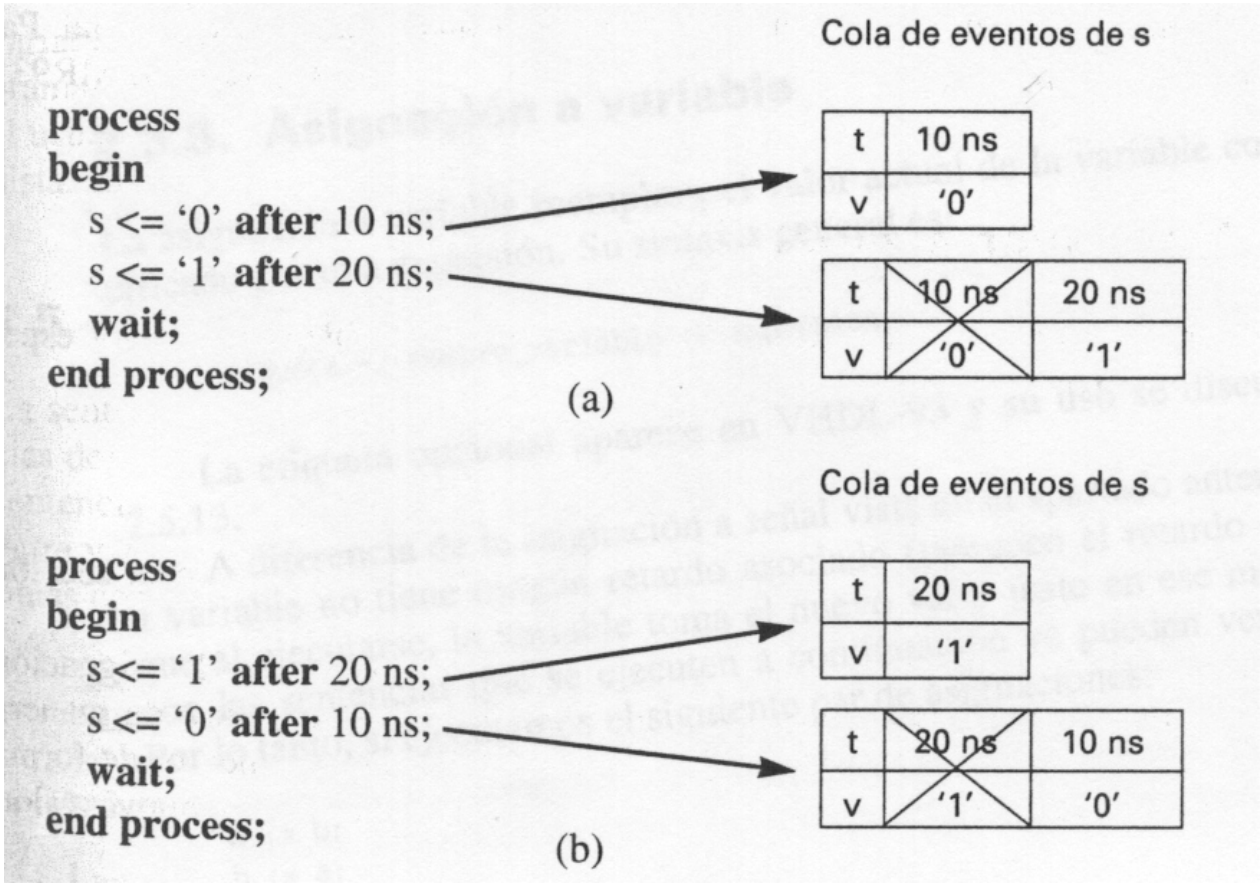




Asignación de la señal (TRANSPORT)



Asignación de la señal (INERTIAL)



15. Señales resueltas

- Un driver de la señal es cada proceso que asigna un valor a una señal.
- Si una señal tiene asociado mas de un driver hay que determinar cual de los diferentes valores se toma como valor de la señal.
- La señal estará resuelta si tiene una función asociada que determina ese valor.

15. Señales resueltas

- El programador puede crear una función de resolución.
- En el paquete *arq-pack* se define un tipo de datos tri-estado (‘0’,’1’,’z’) y un array del tipo tri-estado llamado vector-bus
- En el paquete *arq-pack* existe una función de resolución y un subtipo bus-resuelto.

16. Secuencias de generación

- Cuando tenemos varias conexiones de un mismo componente conectado según una regla es útil emplear la sentencia GENERATE .
- GENERATE permite crear una o más copias de un conjunto de interconexiones facilitando el diseño de circuitos.

16. Secuencias de generación

- 16.1 Sintaxis

```
for indice in rango generate
   --instrucciones concurrentes
end generate;

componente comp
   port(x:in bit; y: out bit);
end component comp
...
signal a, b: bit_vector (o to 7)
...
gen: for i in 0 to 7 generate
u: comp port map(a(i), b(i));
end generate gen;
```

16. Secuencias de generación

16.2 Ejemplo

Registro de 8 bits a partir de biestables (I):

```
ENTITY biestable IS
    PORT (clk, reset, c: in bit;
    d:out bit);
end biestable
```

```
ARCHITECTURE comport OF biestable
    IS
BEGIN
    PROCESS (clk, reset)
    BEGIN
        IF (reset='1') THEN d<='0';
        ELSEIF (clk'event and clk='1)
          THEN   d<=c;
        END IF;
    END PROCESS;
END comport;
```

16. Secuencias de generación

16.2 Ejemplo

Registro de 8 bits a partir de biestables (II):

```
ENTITY registro_8 IS
    PORT (clk, reset: in bit;  A: in bit_vector(7 downto 0);
    B: out bit_vector (7 downto 0);
end registro_8;
```

```
ARCHITECTURE estructural OF registro_8 is
COMPONENT biestable
    PORT(clk, reset, c: in bit;  d:out bit);
END COMPONENT biestable;
SIGNAL R: bit_vector (7 downto 0);
BEGIN
    gen: FOR i in 0 TO 7 GENERATE
            u: biestable PORT MAP(clk, reset, A(i), R(i));
    END GENERATE gen;
    B<=R;
END estructural;
```