Laboratorio de
Comunicaciones y
Teledetección
Dpto. Física Fundamental y Experimental,
Electrónica y Sistemas

VHDL
Presentación del lenguaje

Contenidos

- Introducción al lenguaje. Modelos de hardware
  - Introducción
    - Aplicación de VHDL
    - Rango de uso
    - Historia del VHDL
    - Campos de aplicación
  - Conceptos básicos
    - Niveles de abstracción
    - Estilos descriptivos
  - Modelos de hardware
- Unidades básicas de diseño
  - Declaración de la Entidad y arquitectura, un ejemplo de diseño en VHDL

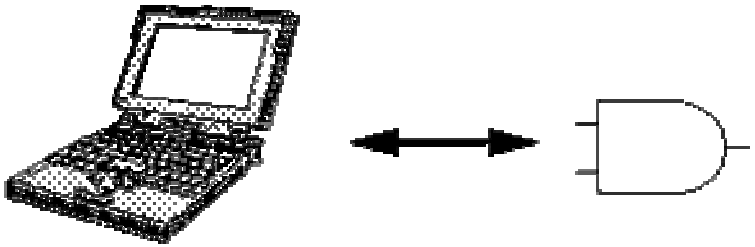
Contenidos

- Introducción
- Ejemplo de diseño en VHDL
- Modelo de componentes en VHDL
  - Declaración de la Entidad
  - Descripción de la arquitectura
- Construcciones Básicas en VHDL
  - Tipos de datos, Objetos
  - Sentencias Secuénciales y Concurrentes
  - Paquetes y Librerías
  - Atributos, Operadores
- Resumen

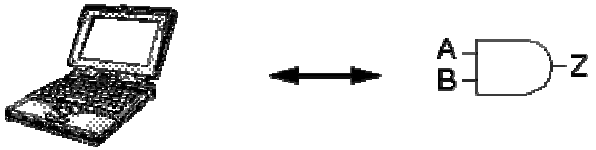
Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

VHDL - Una vista y campos de aplicación

Que es hardware?



Que clase de descripción?




```
if PRINTREQUEST then
   -- print protokoll
end if;
```

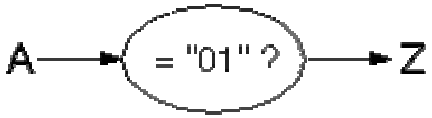
```
Z <= A and B;
```

Hardware Description Language (HDL) =
lenguaje de “programación” para modelado de hardware digital

VHDL - Una vista y campos de aplicación

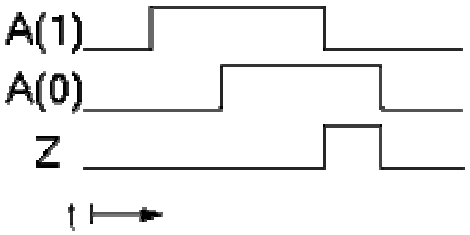
Aplicaciones de los HDLs (1)

Modelling

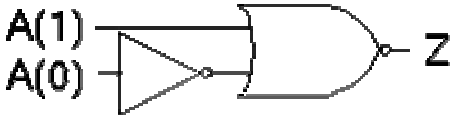



```
Z <= '1' when A="01"
     else '0' ;
```

Simulation



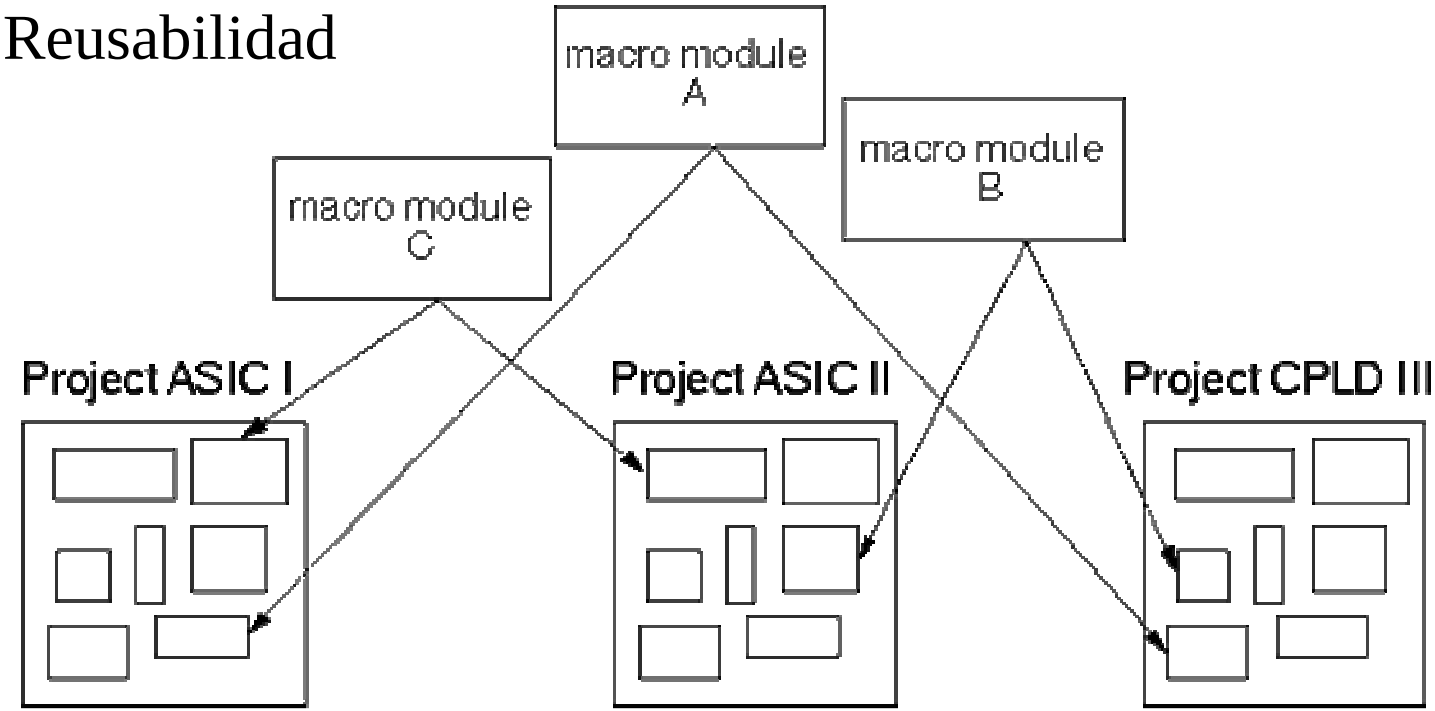
Synthesis



VHDL - Una vista y campos de aplicación

Aplicaciones de los HDLs (2)

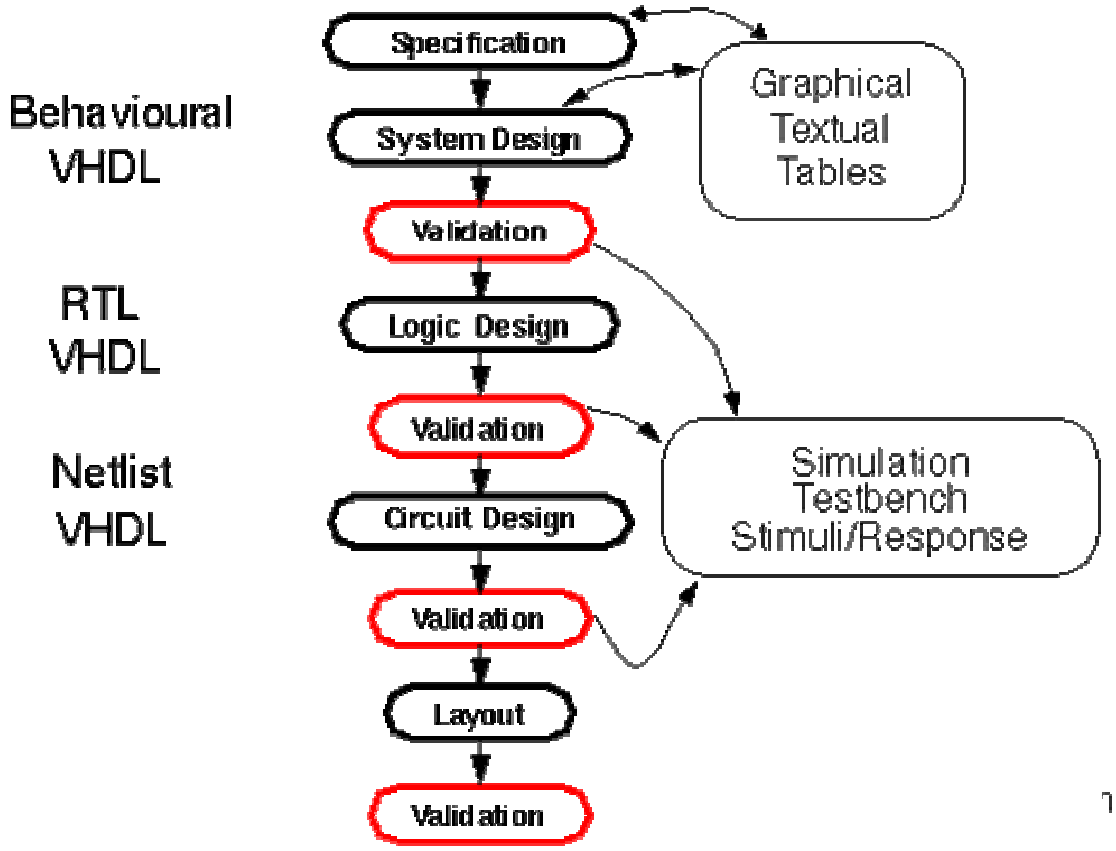
Reusabilidad



Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

VHDL - Una vista y campos de aplicación

Rango de uso en el proceso de diseño



Especificaciones

Descripción de los requerimientos del sistema

Diseño del Sistema

Modelado del comportamiento

Diseño lógico

Modelado de la estructura

Diseño del circuito

Modelado automático de la descripción estructural del circuito

Validación

Chequear la función a través de la simulación suministrando unos estímulos de entrada y chequeando las respuesta esperada

VHDL - Una vista y campos de aplicación

Diseño de Hardware.

- ASIC: mapeo sobre alguna tecnología
- FPGA: mapeo sobre los CLB
- PLD: Estructuras pequeñas, poco uso del VHDL
- Soluciones Estándares, modelos, descripción comporamental, ...

Diseño de software

- VHDL - C interface (herramientas específicas)
- Principal punto de interés (hardware/software co-diseño)

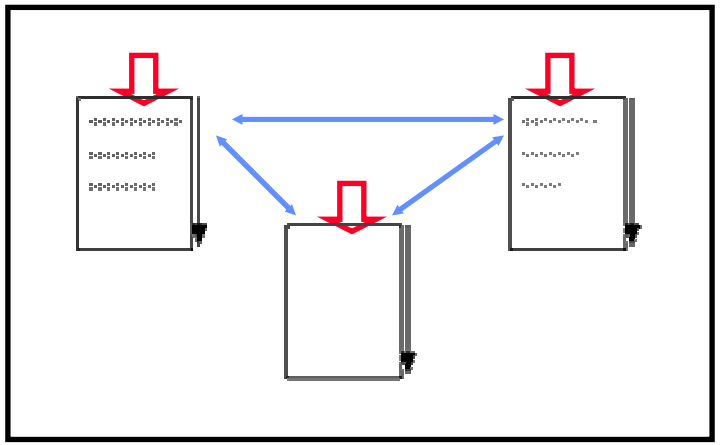
Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

VHDL - Una vista y campos de aplicación

Conceptos de VHDL

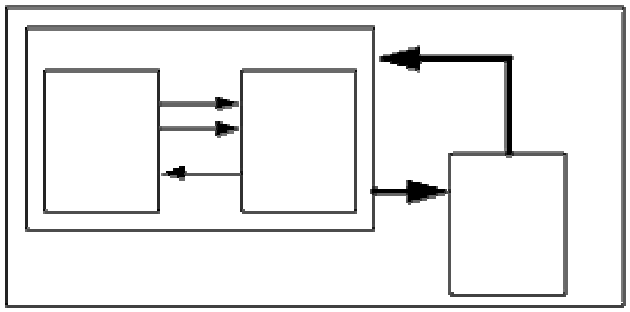
Ejecución de las asignaciones:

- Secuenciales
- Concurrentes



Metodología:

- Abstracción
- Modularidad
- jerarquía



Historia de VHDL

- El Programa Very High Speed Integrated Circuit (VHSIC)
  - Lanzado en 1980 Gobierno Americano
  - El objetivo fue la de realizar un avance significativo en el diseño (Tecnología) VLSI
  - Había necesidad de la creación de un lenguaje común para describir hardware
- En Diciembre 1987, VHDL es acogido como Estándar del IEEE 1076-1987 y en 1988 un ANSI
- En Septiembre 1993, VHDL fue restandarizado Para clarificar y mejorar el lenguaje

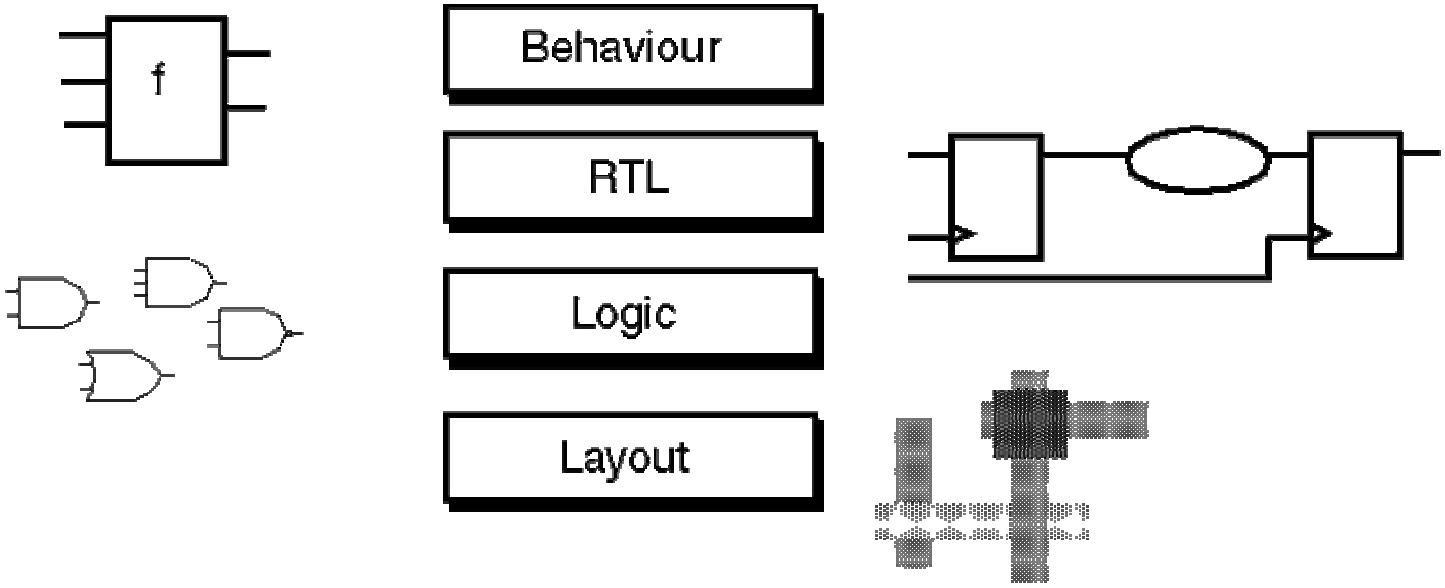
Abstracción

Abstracción es la ocultación de detalles:
Se diferencia entre información esencial y no esencial

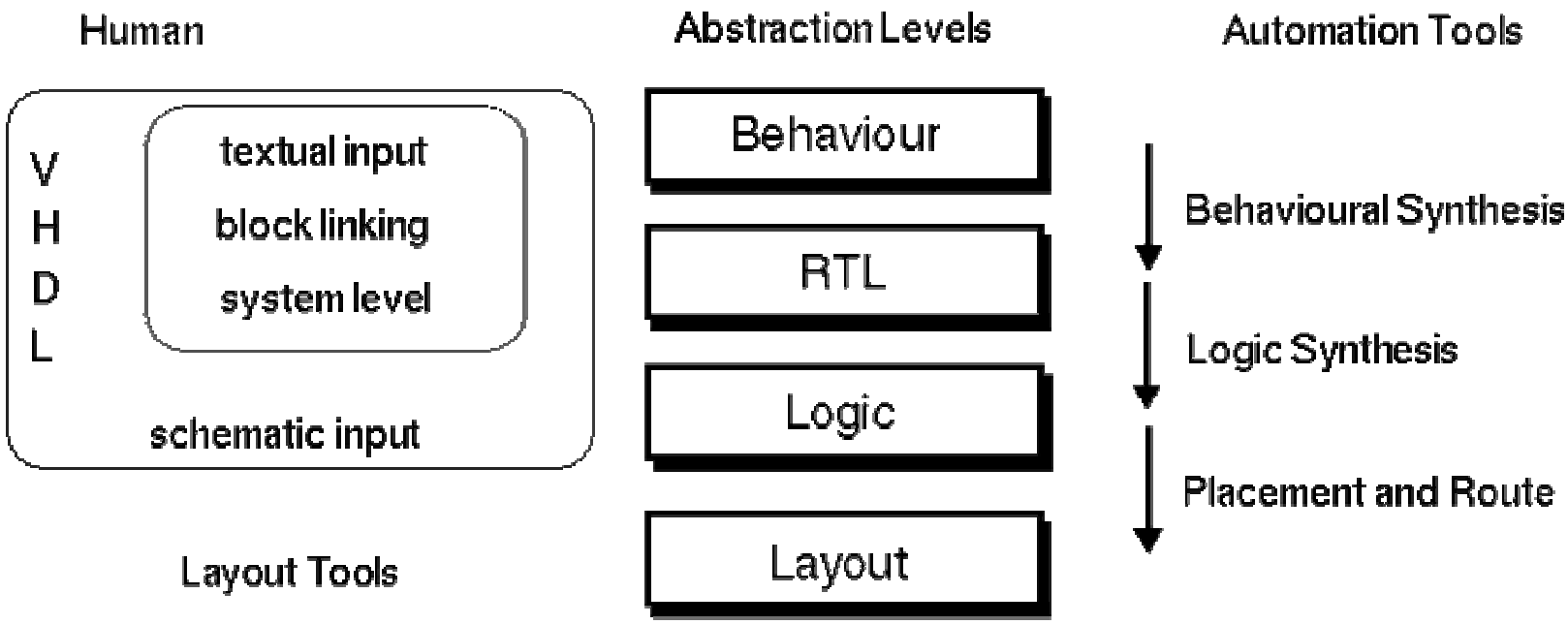
Creación de niveles de abstracción :
Sobre cada nivel de abstracción solamente la información esencial es considerada el resto es dejada fuera

Igualdad de abstracción:
Toda la información de un modelo sobre un nivel de abstracción contiene el mismo grado de abstracción

Niveles de Abstracción en el diseño IC

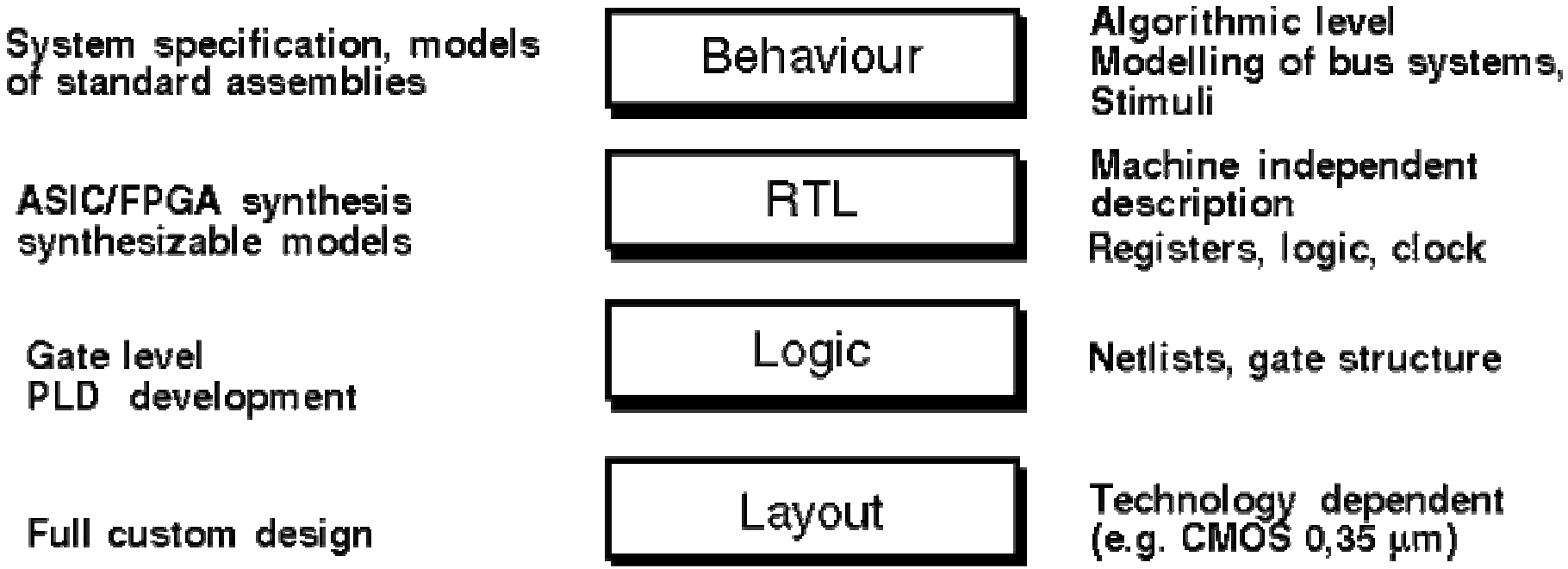


Niveles de Abstracción y VHDL

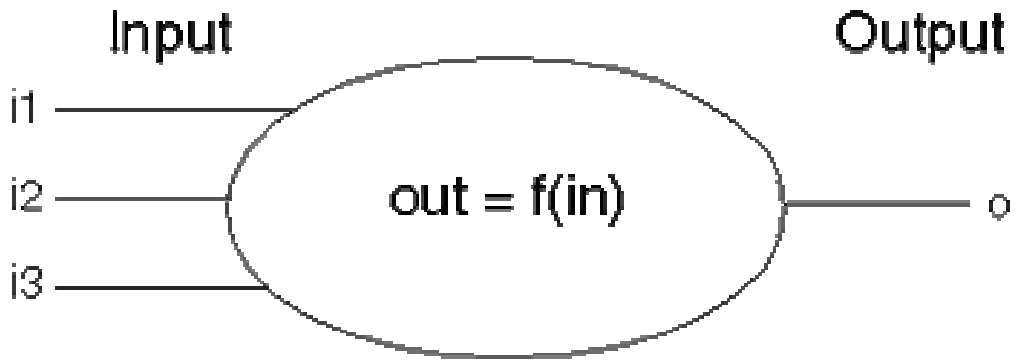


Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

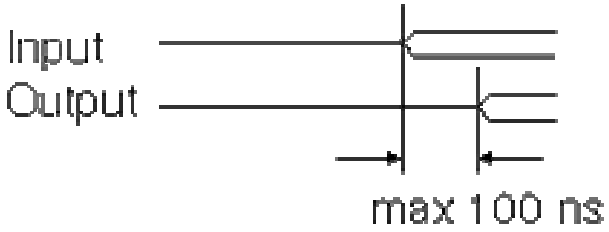
Descripción de los niveles de abstracción



Descripción en el nivel comportamental en VHDL

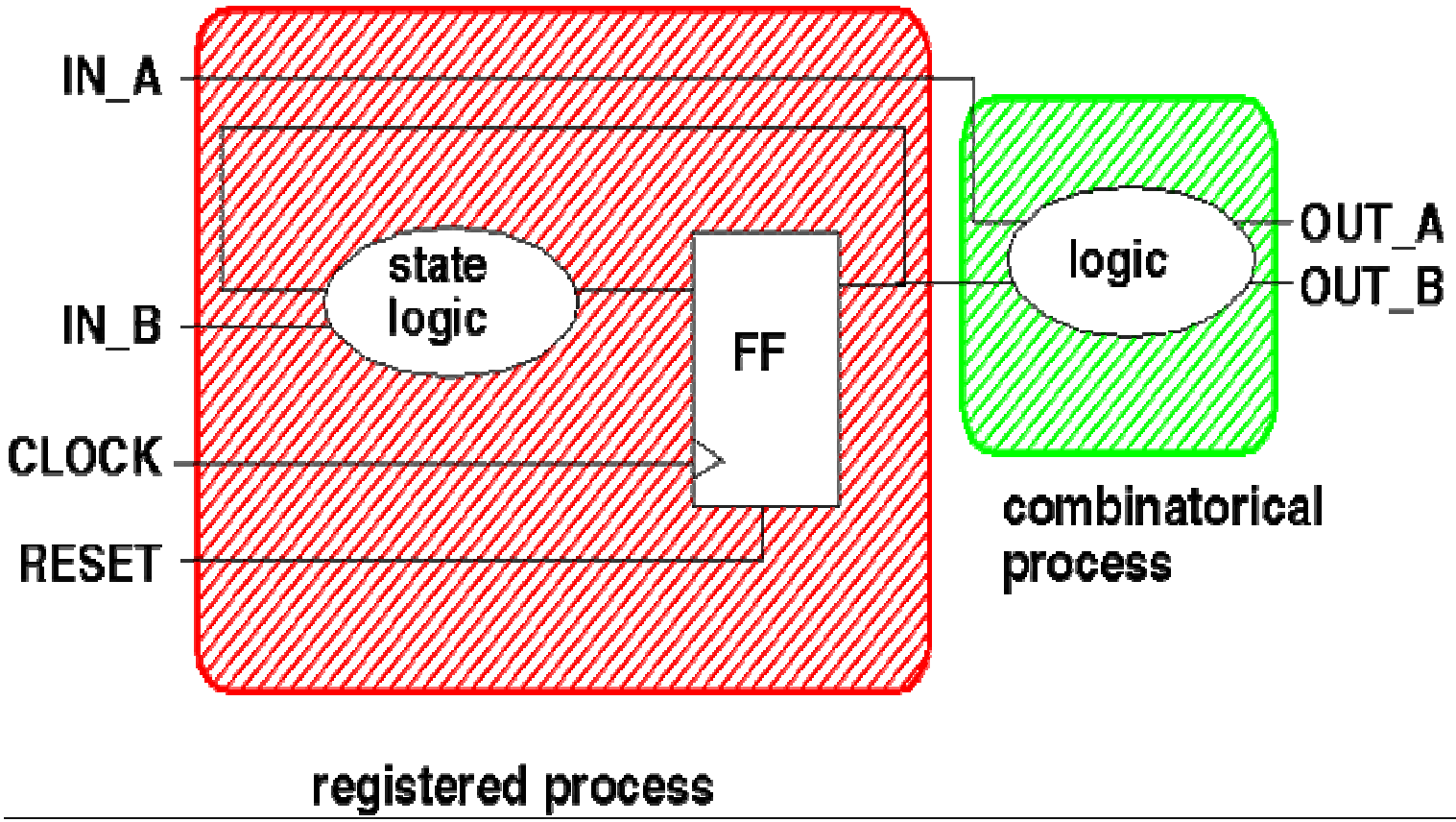


Specification.

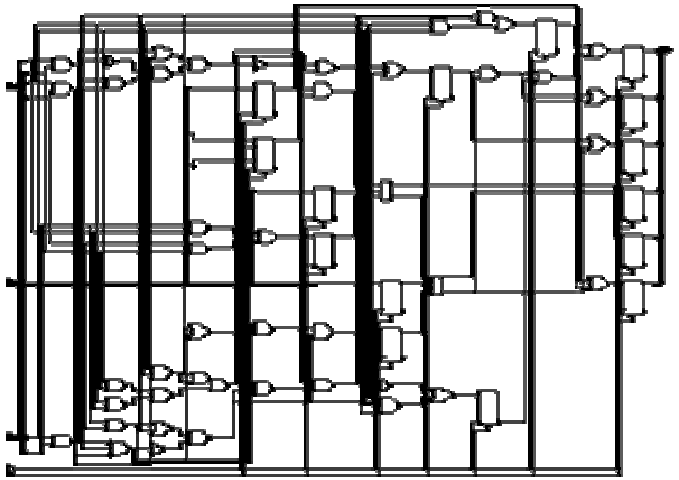


*o <= transport i1 + i2 * i3 after 100 ns;*

Descripción en el Nivel RT

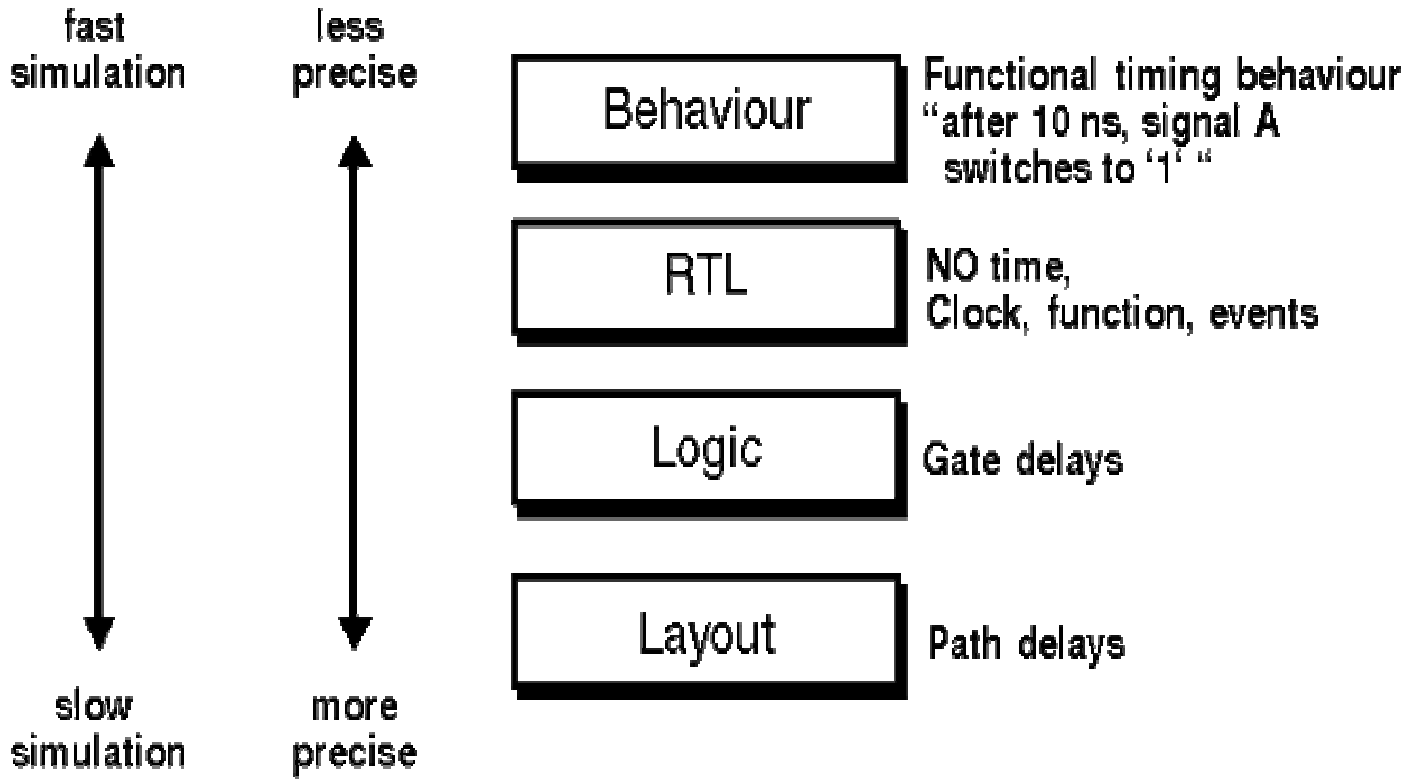


Descripción en el Nivel Lógico



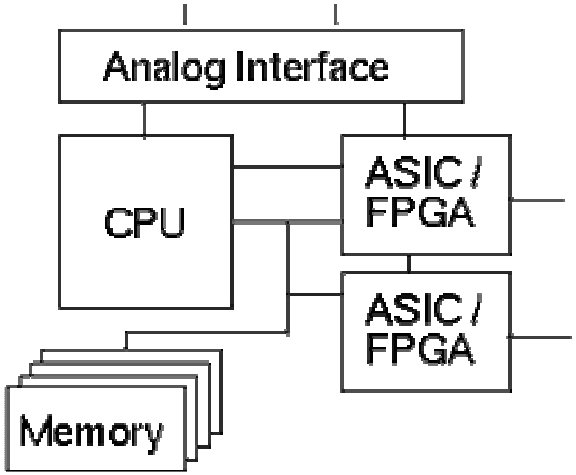
```
U86  ND2 port map( A => n192, B => n191, Z => n188),
U87  ND2 port map( A => I3_2, B => I2_0, Z => n175),
U88  ND2 port map( A => I2_2, B => I3_0, Z => n173),
U89  NR2 port map( A => mul_36_PROD_not_0,
       B => n174, Z => n185),
U90  EN port map( A => n181, B => n182, Z => n180);
U91  ND2 port map( A => I3_2, B => I2_1, Z => n181);
U92  ND2 port map( A => I2_2, B => I3_1, Z => n182);
U93  IVP port map( A => n180, Z => n192);
U94  AO6 port map( A => n173, B => n174, C => n175,
       Z => n172),
U95 : NR2 port map( A => n174, B => n173, Z => n176);
U96 : ND2 port map( A => I3_1, B => I2_1, Z => n174);
U97 : EN port map( A => n183, B => n178,
       Z => product64_4);
U98 : ND3 port map( A => I2_2, B => I3_2, C => n174,
       Z => n183),
```

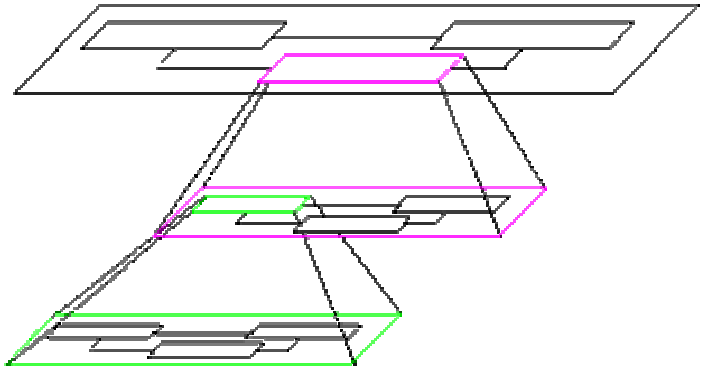
Contenido de información de los niveles de abstracción



Modularidad y Jerarquía

Permite la partición, en diferentes bloques o módulos (diseños parciales), de sistemas complejos
Restringiendo la complejidad
Permitiendo o facilitando el trabajo de el equipo de diseño
Facilita el estudio de diferentes alternativas de implementación
permite implementar modelos Soft para tener en cuenta el entono real de operación.





Razones para usar VHDL

- VHDL es una especificación de lenguaje internacional estándar del IEEE (IEEE 1076-1993) para describir hardware digital usado en la industria de todo el mundo
- VHDL es un acrónimo de VHSIC (Very High Speed Integrated Circuit) Hardware Description Lenguage
- VHDL permite modelar hardware desde nivel de sistema hasta nivel de puertas
- VHDL suministra un mecanismo para el diseño digital y re_uso de la documentación de los diseños

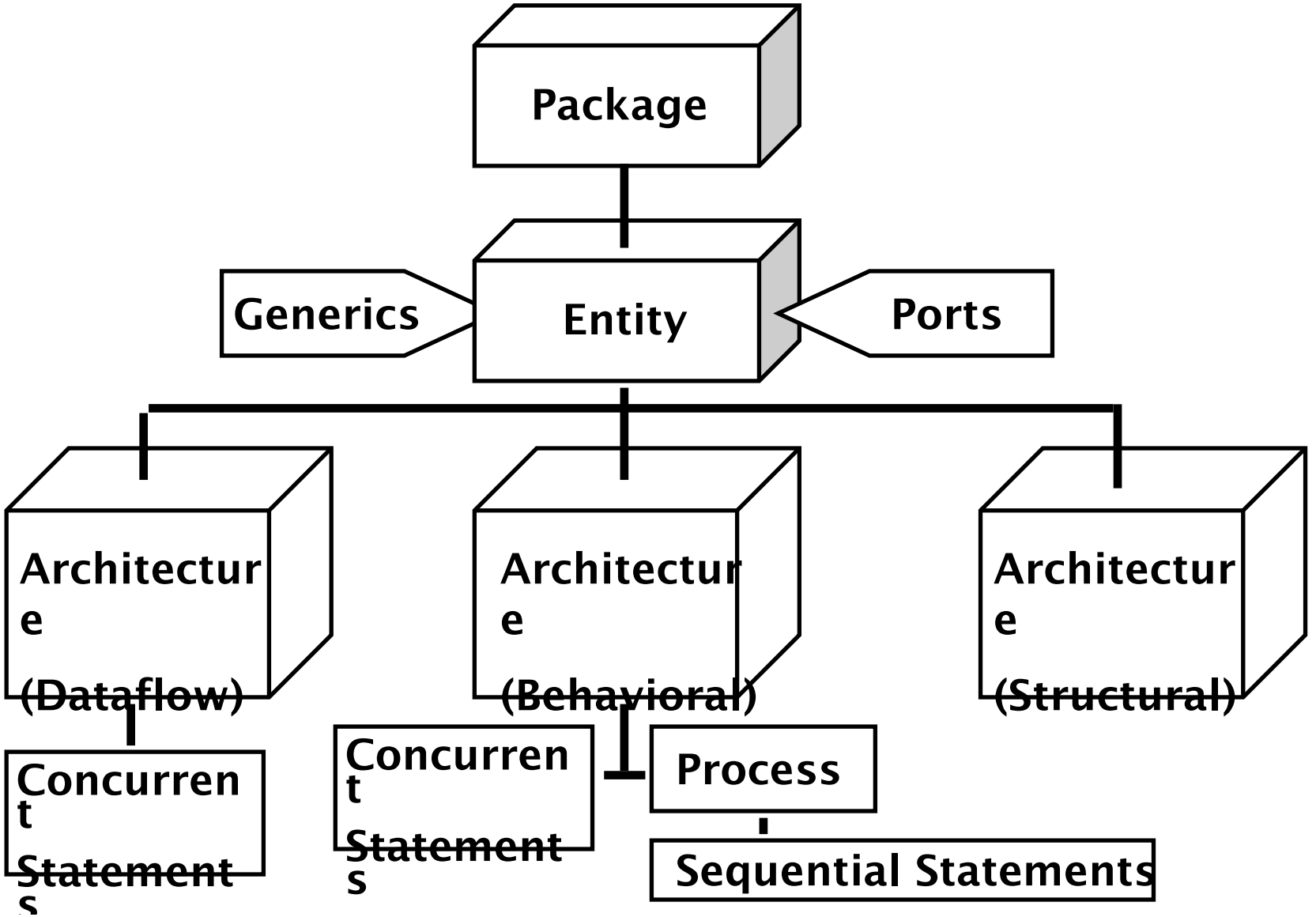
Beneficios Adicionales de VHDL(1)

- Permite varias metodología de diseño
- Es independiente de la tecnología
- Describe una ancha variedad de hardware digital
- Facilita la comunicación al ser un lenguaje estándar
- Permite un mejor manejo de los diseños
- Es un lenguaje flexible

Beneficios Adicionales de VHDL (2)

- **Cada diseño tiene:**
  - **Una interfase bien definida**
  - **Un comportamiento preciso usando :**
    - **Un algoritmo**
    - **O una descripción estructurada**
- **Modela concurrencia, timing, y clocking:**
  - **Maneja circuitos asíncronos y sincronos**
  - **Los diseños pueden ser simulados**

Poniendo todo junto

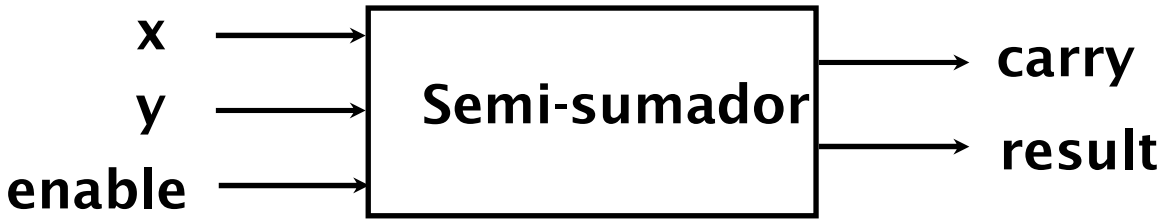


Contenidos del Modulo

- Introducción
- **VHDL Diseño Ejemplo**
- VHDL Modelos de Hardware
- Basic VHDL Construcciones
- Ejemplos
- Summary

Ejemplo de Diseño en VHDL

- **Problema: Diseñar un Simple semi-sumador de un Bit con carry and enable**
- **Especificaciones**
  - **Entradas y salidas son cada una de bit**
  - **Cuando *enable* esta a nivel alto, *result* es *x* + *y***
  - **Cuando *enable* esta a nivel alto, *carry* es el carry de *x* + *y***
  - **Las salidas son cero cuando *enable* esta a nivel bajo**



Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

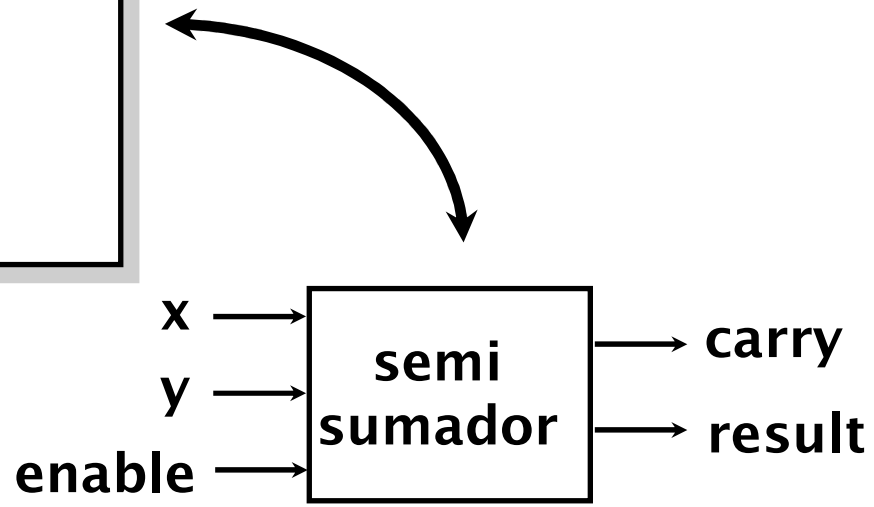
Ejemplo de Diseño en VHDL
Declaración de la Entity

- **Como primer paso, la declaración de la entidad *entity* permite definir la interfaces del componente**
  - **Puertos de entrada y Salidas son declarados**

```
ENTITY semi_sumador IS

    PORT( x, y, enable: IN BIT;
          carry, result: OUT BIT);

END semi_sumador;
```



Ejemplo de Diseño en VHDL

Especificación Comportamental

- **Una descripción a alto nivel puede ser usada para describir la función del sumador**

```
ARCHITECTURE semi_sumador_a OF semi_sumador IS
    BEGIN
      PROCESS (x, y, enable)
        BEGIN
                IF enable = '1' THEN
                        result <= x XOR y;
                        carry <= x AND y;
                ELSE
                        carry <= '0';
                        result <= '0';
                END IF;
        END PROCESS;
    END semi_sumador_a;
```

- **El modelo puede ser simulado para verificar la correcta funcionalidad del componente**

Ejemplo de Diseño en VHDL

Especificación Flujo de datos

- **Un segundo método es haciendo uso de las ecuaciones boleana, flujo de datos**

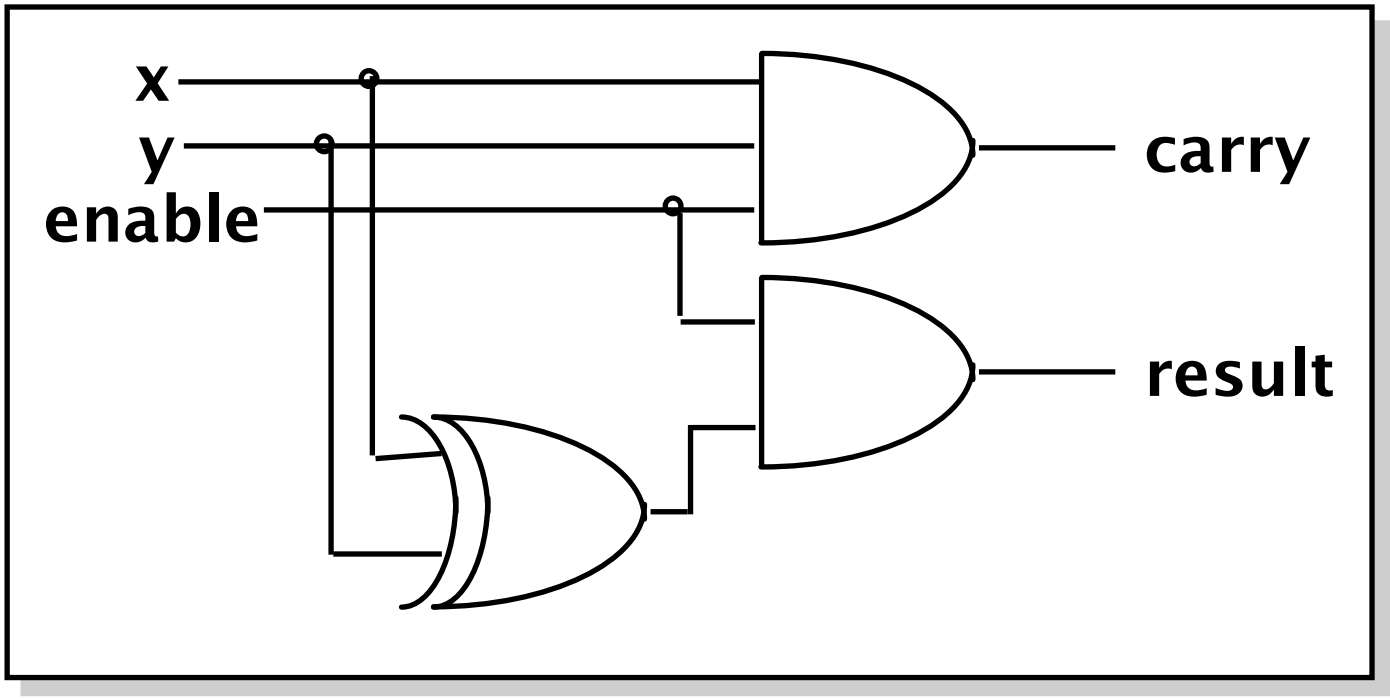
```
ARCHITECTURE semi_sumador_b OF semi_sumador IS
    BEGIN
      carry <= enable AND (x AND y);
      result <= enable AND (x XOR y);
    END semi_sumador_b;
```

- **De nuevo, el modelo puede ser simulado para confirmar las ecuaciones lógicas**

Ejemplo de Diseño en VHDL

Especificación Estructural

- Como tercer método, una descripción estructural puede ser creada a partir de componentes ya predefinidos



- Estas puertas pueden ser extraídas de una librería

Ejemplo de Diseño en VHDL

Especificación Estructural (Cont.)

```
ARCHITECTURE semi_sumador_c OF semi_sumador IS

    COMPONENT and2
      PORT (in0, in1 : IN BIT;
            out0 : OUT BIT);
    END COMPONENT;

    COMPONENT and3
      PORT (in0, in1, in2 : IN BIT;
            out0 : OUT BIT);
    END COMPONENT;

    COMPONENT xor2
      PORT (in0, in1 : IN BIT;
            out0 : OUT BIT);
    END COMPONENT;

    FOR ALL : and2 USE ENTITY gate_lib.and2_Nty(and2_a);
    FOR ALL : and3 USE ENTITY gate_lib.and3_Nty(and3_a);
    FOR ALL : xor2 USE ENTITY gate_lib.xor2_Nty(xor2_a);

-- La descripción sigue en la siguiente transparencia
```

Ejemplo de Diseño en VHDL

Especificación Estructural (cont.)

```
--
    SIGNAL xor_res : BIT; -- Señal interna
    -- Notar que las otras señales ya están declaradas en la --
    -- entidad

    BEGIN

      A0 : and2 PORT MAP (enable, xor_res, result);
      A1 : and3 PORT MAP (x, y, enable, carry);
      X0 : xor2 PORT MAP (x, y, xor_res);

    END semi_sumador_c;
```

Contenidos del Modulo

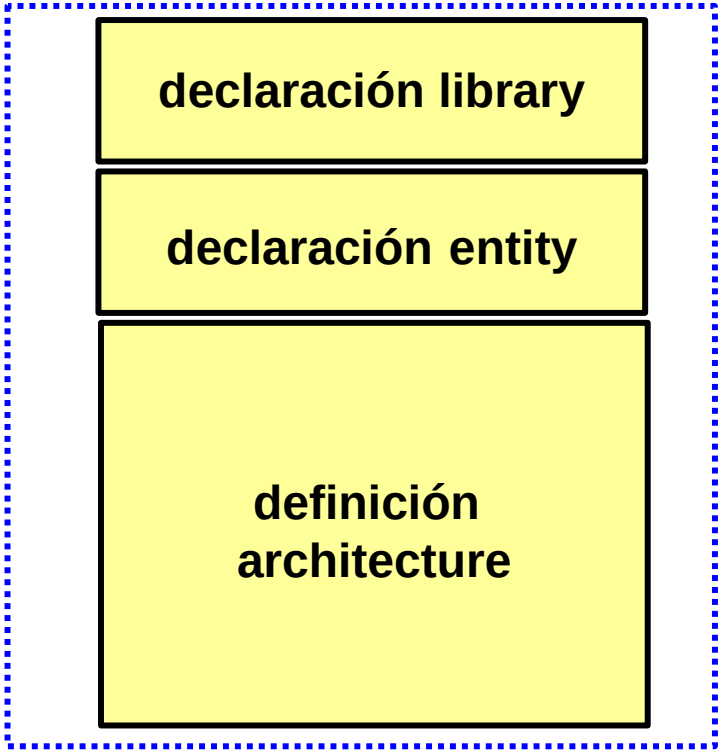
- Introducción
- Ejemplo de diseño en VHDL
- Modelo de componentes en VHDL
  - Declaraciones de la entidad
  - Descripciones de la arquitectura
- Construcciones Básicas en VHDL
- Ejemplos
- Resumen

Modelos de componentes en VHDL (1)

- Una descripción completa de un ***componente*** en VHDL requiere de una *entity* y una *architecture*
  - La ***entity*** define la interface del componente
  - La ***architecture*** define a la función del componente
- Varias arquitecturas alternativas pueden ser desarrollada para el uso con una misma entidad
- Existen dos formas básica para describir un componente en VHDL:
  - Descripción estructural
  - Descripción comportamental

Modelos de componentes en VHDL (2)

- **Código Fuente, Fichero de Texto (ej., midiseño.vhd)**



Declaración de la Entidad

Sintaxis:

```
entity entity-name is
  port (
        signal-name: modo signal-type;
        ...
        signal-name: modo signal-type);
end entity-name;
```

Hecho ya o definidos por el usuario

Ojo al final no punto y coma

Modos:

- *in*
- *out*
- *inout*
- *buffer*

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Declaración de la Entidad (ejemplo)

- El propósito principal de una entidad es la declarar las señales en la interfase del componente
  - Las señales de la interfase son listadas en la cláusula PORT
    - Según esto la entidad es el equivalente a el símbolo de un componente en un esquemático
  - Las demás cláusulas en la entidad serán introducidas mas tarde


```
ENTITY half_adder IS

    GENERIC(prop_delay : TIME := 10 ns);

    PORT( x, y, enable : IN BIT;
          carry, result : OUT BIT);

END half_adder;
```

Definición de la Arquitectura

- La unidad fundamental para describir el comportamiento de un componente es el *proceso*
  - Los procesos pueden ser definidos explícita o implícitamente dentro de una arquitectura

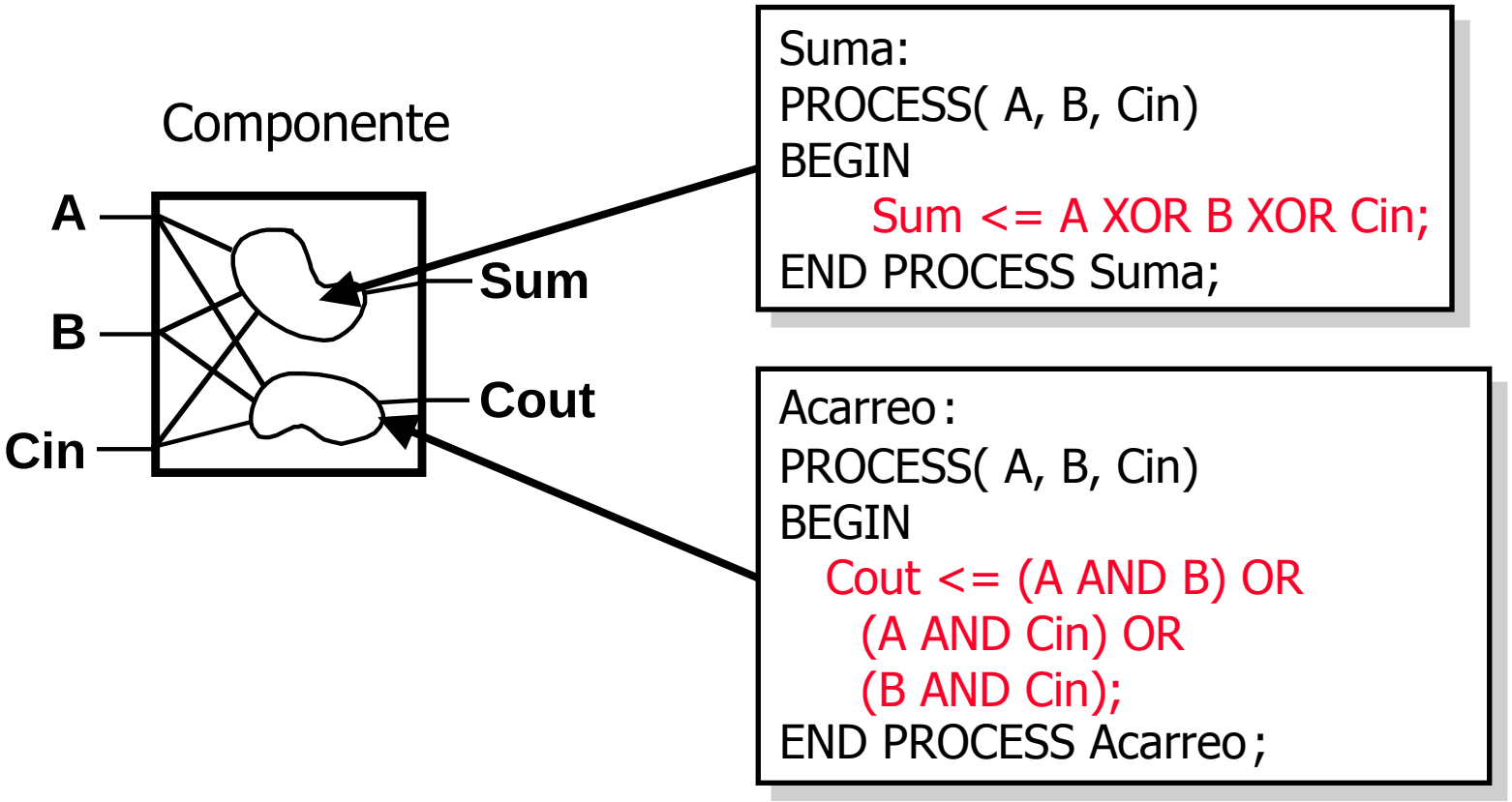
Syntax:

```
Process-Name: process (sensitivity-list)
begin
  statements
end process;
```

- El mecanismo primario de comunicación en los *procesos* son las señales
  - la ejecución de los procesos programan nuevos valores de las señales los cuales son accesibles a otros procesos
  - Similarmente, una señal puede ser accedida por otro proceso en otra arquitectura a través de los puertos de la entidad asociada

Definición de la Arquitectura

- Una descripción completa de un componente en VHDL requiere de una *entity* y una *architecture*
  - La *entity* define la interface del componente
  - La *architecture* define a la función del componente



Puertos del componente
Port cláusula

- **PORT declara las señales de interfase de los objetos con el mundo exterior**

- **Partes de la cláusula PORT**
  - **nombre_señal**
  - **Modo**
  - **Tipo_dato**

```
PORT (nombre_señal : modo tipo_dato);
```

- **Ejemplo PORT :**

```
PORT ( input : IN BIT_VECTOR(3 DOWNTO 0);
       ready, output : OUT BIT );
```

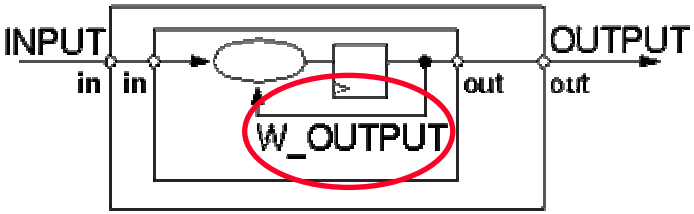
  - **Notar que puertos del mismo modo y tipo deben ser declarado en la misma línea**

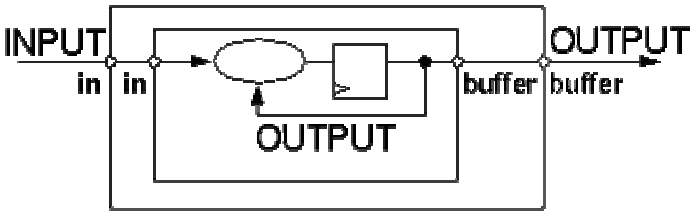
Puertos del componente
Port cláusula (cont.)

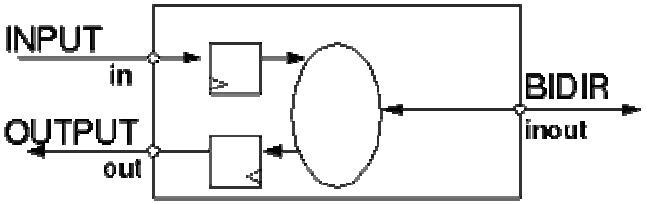
- El *modo* del puerto describe la dirección en la cual viajan los datos con respecto al *componente*

- 5 modos hay disponible:
  - In - los datos entran en el puerto, y solamente pueden ser leídos
  - Out - Los datos salen del puerto
  - Buffer – los datos salen del puerto pero estos pueden ser leídos desde la propia entidad
  - Inout - Los datos pueden viajar en cualquier dirección

Puertos del componente
Port cláusula (cont.)







in: Los valores de las señales son de lectura

out: Los valores de las señales son de escritura por múltiples drivers

buffer: comparable a out pero de escritura/lectura

inout: puerto bi-direccional

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Parámetros del componente
Cláusula Generic

- Generics puede se usado para legibilidad, mantenimiento y configuración
- Sintaxis de la declaración generic :
  - Si el valor opcional por defecto no es explicitado en la

```
GENERIC (generic_name : type [:= default_value]);
```

    componente sea usado (por ejemplo en la instanciación del mismo)
- Ejemplo de genérico:
  - El parámetro genérico *Mi_ID,con valor por defecto* 37, puede ser referenciado en alguna arquitectura de la entidad

```
GENERIC (Mi_ID : INTEGER := 37);
```

  - El valor de defecto puede ser cambiado en la instanciación

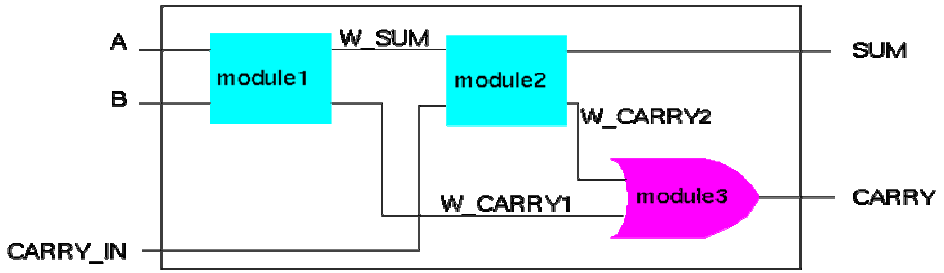
Cuerpo de la Arquitectura

- **Describe La funcionalidad del componente**
- **Posee dos partes :**
  - **Parte declarativa -- Incluye todas las declaraciones necesaria**
    - **Declaración de tipos, señales, componentes y subprogramas**
  - **Parte sentencias -- incluye sentencia que describen organización y/o funcionalidad del componente**
    - **Tales como: Asignación concurrente de señales, procesos componentes instanciación**

```
ARCHITECTURE half_adder_d OF half_adder IS
    SIGNAL xor_res : BIT;    --Parte declarativa de la arquitectura
    BEGIN                  -- Parte de sentencias de la arquitectura
      carry <= enable AND (x AND y);
      result <= enable AND xor_res;
      xor_res <= x XOR y;
    END half_adder_d;
```

Estilos. Descripción Estructural

- Componentes pre-definidos de VHDL son *instanciados* y conectados juntos
- Descripción estructural puede conectar puertas simple y/o componentes complejos
- VHDL suministra los mecanismos para hacer una descripción jerárquica
- VHDL suministra los mecanismos para describir estructuras repetitivas fácilmente



Sumador completo: 2 semi_sum + una puerta OR

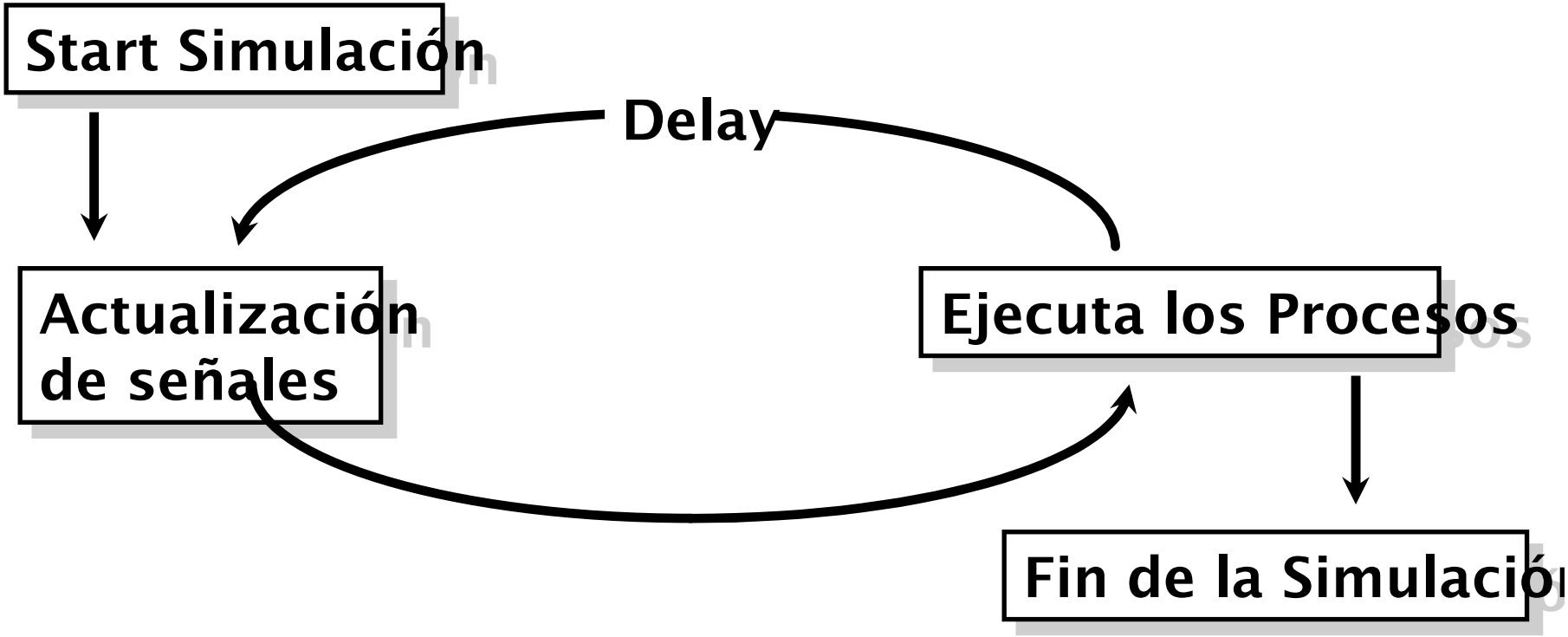
Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Estilos. Descripción Comportamental

- VHDL suministra dos estilos para describir el comportamiento de un componente
  - Flujo de datos: basado en sentencias de asignación concurrentes de señales
  - Comportamental: el *Proceso* es usados para describir comportamientos mediante algoritmo lenguaje a muy alto nivel
    - ejemplos variables, sentencias loops, if-then-else
- Un Modelo comportamental puede suministrar un pequeño parecido a la implementación del sistema
  - No necesariamente implica estructura

Ciclo de Simulación

- VHDL usa el siguiente ciclo de simulación para modelar los estímulos y respuesta natural del Hardware digital

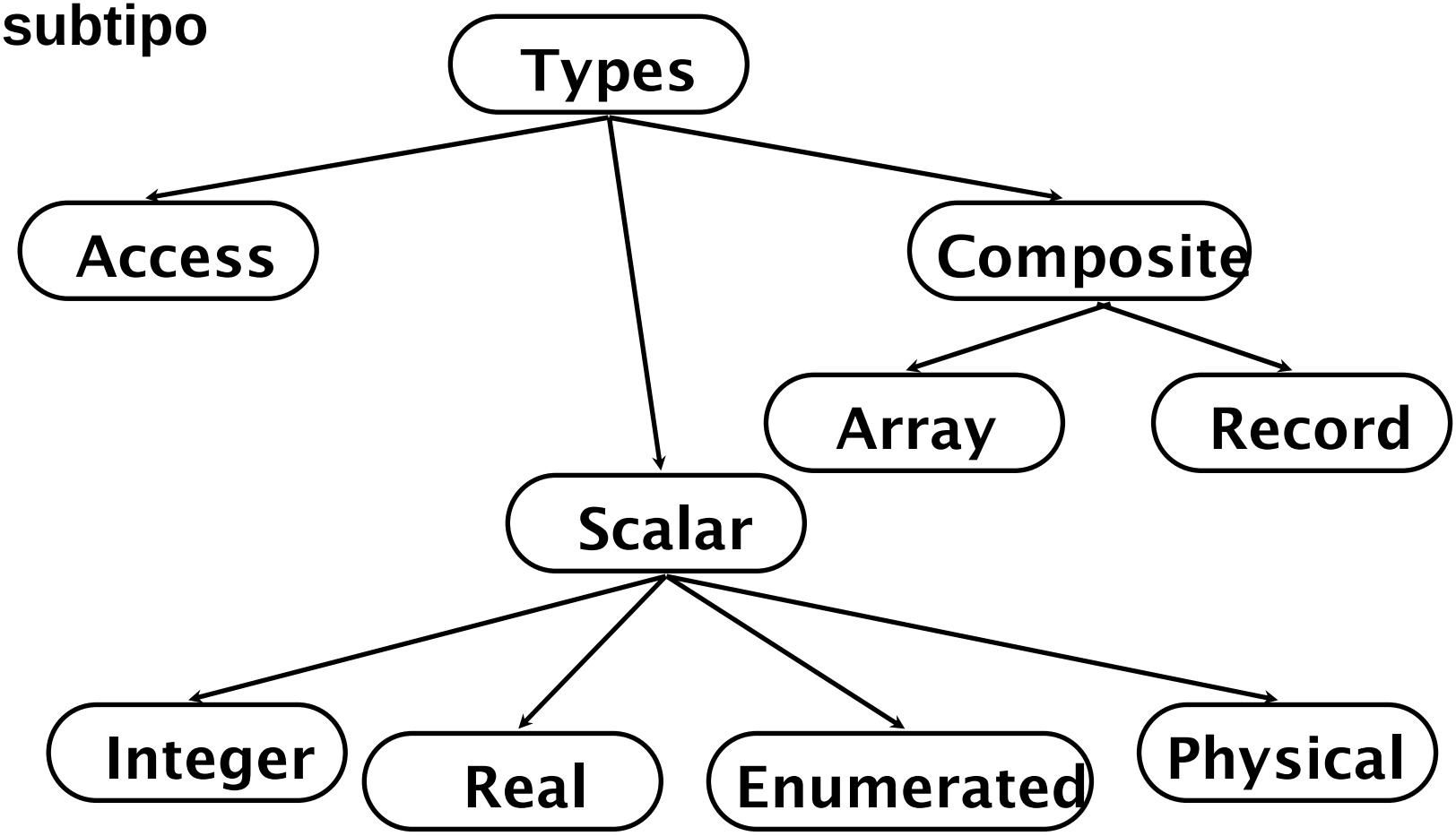


Contenidos del Modulo

- Introducción
- Ejemplo de diseño en VHDL
- Modelo de componentes en VHDL
  - Declaración de la Entidad
  - Descripción de la arquitectura
- Construcciones Básicas en VHDL
  - Tipos de datos, Objetos
  - Sentencias Secuénciales y Concurrentes
  - Paquetes y Librerías
  - Atributos, Operadores
- Resumen

Tipos de Datos

- **Todas las declaraciones de VHDL, ports, signals, y variables deben especificar su correspondiente tipo o subtipo**



Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Tipos de Datos
Declaración

- **Sintaxis de declaración**

```
TYPE nombre IS <definición del tipo> ;
```

- **Ejemplos**
  - **definición**

```
Type DiasMes is range 1 to 31;
Type PuntosCardinales is (norte, sur este, oeste);
```

  - **uso**

```
Constant DiasEnero: DiasMes:=31;
Variable Diahoy: DiasMes;
Signal Dirección: PuntosCardinales;
Port (Entrada: in PuntosCardinales;
      Salida : in Puntoscardinales);
```

Tipos de Datos en VHDL

Tipos Escalar

- **Enteros**
  - **Rango: - 2,147,483,647 a 2,147,483,647**
  - **Ejemplo de asignación a una variable del tipo entero:**

```
ARCHITECTURE test_int OF test IS
BEGIN
    PROCESS (X)
      VARIABLE a: INTEGER;
    BEGIN
      a := 1;   -- OK
      a := -1;   -- OK
      a := 1.0;   -- ilegal
    END PROCESS;
END test_int;
```

Tipos de Datos en VHDL

Tipos Escalar (Cont.)

- **Reales**
  - **Rango: -1.0E38 a 1.0E38**
  - **Ejemplo de asignación a una variable del tipo real :**

```
ARCHITECTURE test_real OF test IS
BEGIN
    PROCESS (X)
      VARIABLE a: REAL;
    BEGIN
      a := 1.3;  -- OK
      a := -7.5;  -- OK
      a := 1;  -- illegal
      a := 1.7E13;  -- OK
      a := 5.3 ns;  -- illegal
    END PROCESS;
END test_real;
```

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Tipos de Datos en VHDL

Tipos Escalar (Cont.)

- **Enumerados**
  - **El usuario define la lista de posibles valores de dicho tipo**
  - **Ejemplo declaración y uso de un tipo enumerado :**

```
TYPE binary IS ( ON, OFF );
--.... más declaraciones ...
ARCHITECTURE test_enum OF test IS
BEGIN
    PROCESS (X)
      VARIABLE a: binary;
    BEGIN
      a := ON;  -- OK
      ... Más declaraciones ...
      a := OFF;  -- OK
      ... Más declaraciones ...
    END PROCESS;
END test_enum;
```

Tipos de Datos en VHDL

Tipos Escalar (Cont.)

- **Físicos**
  - **Requiere de sus unidades asociadas**
  - **El Rango debe ser especificado**
  - **Ejemplo de una declaración del tipo físico:**

```
TYPE resistance IS RANGE 0 TO 10000000

UNITS
ohm;   -- ohm
Kohm = 1000 ohm;   -- i.e. 1 KΩ
Mohm = 1000 kohm;   -- i.e. 1 MΩ
END UNITS;
```

  - **Time es el único tipo físico predefinido en VHDL**

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Tipos de Datos en VHDL
Tipos Compuestos

- **Array**
  - Son usados para agrupar elementos de un mismo tipo en un objeto de VHDL
  - El Rango puede estar sin restringir en la declaración
    - Pero debería ser restringido cuando es usado
  - Ejemplo para un array unidimencional (vector):

```
TYPE data_bus IS ARRAY(0 TO 31) OF BIT;
```

0 ... indices de elem ... 31

0	...valores del array...	1

```
VARIABLE X : data_bus;
VARIABLE Y : BIT;

Y := X(12);  -- Y toma el valor del elemento cuyo
indice es 12
```

Tipos de Datos en VHDL

Tipos compuestos (Cont.)

- **Ejemplo usando DOWNTO :**

```
TYPE reg_type IS ARRAY(15 DOWNTO 0) OF BIT;
```


```
VARIABLE X : reg_type;
VARIABLE Y : BIT;

Y := X(4);
```

- **DOWNTO palabra es usada si el índice mas significativo es el de la izquierda**

Tipos de Datos en VHDL

Tipos compuestos (Cont.)

- **Records (registros)**
  - **Son usados para agrupar elementos de posiblemente distintos tipos en un objeto de VHDL**
  - **Los Elementos son indexados vía nombre y campo**
  - **Ejemplo de registro o record:**

```
TYPE binary IS ( ON, OFF );
TYPE switch_info IS
   RECORD
     status : BINARY;
     IDnumber : INTEGER;
END RECORD;

VARIABLE switch : switch_info;
switch.status := ON;  -- Estado del  switch
switch.IDnumber := 30;  -- nº de identificación del
```

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Tipos de Datos en VHDL
Subtipos

- **Subtype**
  - **Permite definir un subconjunto de algún tipo**
  - **Puede incluir el rango entero del tipo base**
  - **Asignaciones fuera del rango del subtipo son ilegales**
    - **Error en tiempo de ejecución**
  - **Sintaxis de la declaración de Subtype:**

```
SUBTYPE nombre IS type_base RANGE <rango>;
```

  - **Ejemplo Subtype :**

```
SUBTYPE primeros_diez IS INTEGER RANGE 0 TO 9;
```

Tipos de Datos en VHDL

- **Useful Built-in Types**
  - *integer*: ***defined as in other languages, though its width is inferred***
  - *boolean*: ***as in other languages***

```
signal signal-name: boolean := true;
signal int-name: integer range start to end;
```

Tipos de Datos en VHDL

- **IEEE Defined Types:**
  - **std_logic** ***used to define signals in hardware***
  - ***std_logic_vector used to define buses (collection of multiple signals)***

std_logic
std_logic_vector

signal *signal-name:* **std_logic;**
signal *bus-name:* **std_logic_vector (***start* **to** *end***);**
signal *bus-name:* **std_logic_vector (***start* **downto** *end***);**

Tipos de Datos en VHDL
Resumen

- Todas las declaraciones de VHDL puertos, señales, y variables tienen que llevar su tipo o subtipos
- Tipos de datos en VHDL son :
  - Access -- Punteros para reserva de memoria dinámica
  - Escalar -- Enteros, Reales, Enumerados, y físicos
  - Compuestos -- Array (vectores), y registros
- Un conjunto de tipos de datos están definidos en VHDL
  - El usuario puede definir también sus propios tipos y subtipos

Objetos en VHDL

- Existen cuatros tipos de objetos en VHDL
  - Constantes
  - Variables
  - Señales
  - Ficheros
- Reglas de visibilidad:
  - Los objetos declarados
    - en un paquete están disponibles para toda la descripción VHDL que usa el paquete
    - en una entidad están disponibles para todas las arquitectura asociadas con la entidad
    - en una arquitectura están disponibles para todas las construcciones en la arquitectura
    - en a proceso están disponibles solamente dentro del proceso

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Objetos en VHDL
Constantes

- **Nombres asignados a un valor especifico de un tipo**
- **Permite una actualización fácil y legibilidad**
- **La declaración de una constante puede omitir el valor de forma que sea aplazado su asignación**
- **Facilita la reconfiguración**
- **Sintaxis de la declaración :**

```
CONSTANT nombre_constante : type_name [:= valor];
```

- **Declaración ejemplos :**

```
CONSTANT PI : REAL := 3.14;
CONSTANT velocidad : INTEGER;
```

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Objetos en VHDL
Variables

- **Suministra un mecanismo para almacenamiento local**
  - **Ejemplo contadores, valores intermedios**
- **Su ubicación está en el proceso**
  - **VHDL '93 proporciona variables globales,**
- **Todas las asignaciones de las variables tienen lugar inmediatamente**
  - **No tiene delta delay o retardos de usuario.**
- **Sintaxis Declaración :**

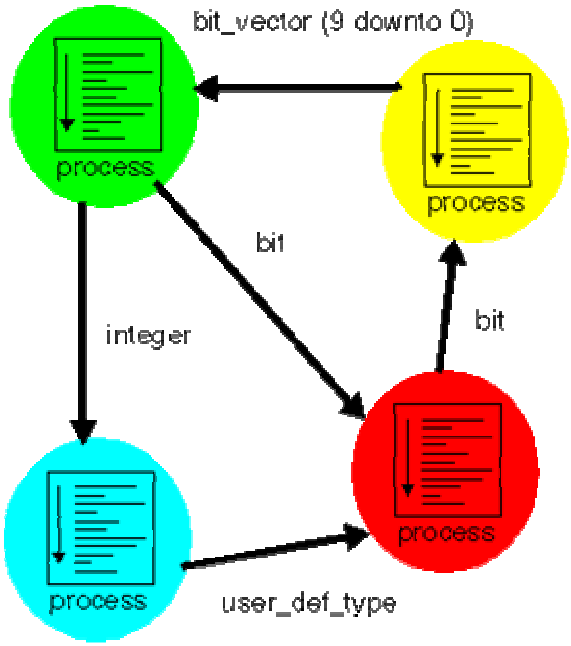
```
VARIABLE variable_name : type_name [:= value];
```

- **Ejemplos de Declaración :**

```
VARIABLE opcode : BIT_VECTOR(3 DOWNTO 0) := "0000";
VARIABLE freq : INTEGER;
```

Objetos en VHDL
Señales

- Usadas para la comunicación entre componentes de VHDL
- Realmente, las señales física en un sistema a menudo son mapeadas en señales de VHDL
- TODAS las asignaciones de señales en VHDL requiere un delta ciclo o un delay especificado por el usuario antes de tomar su nuevo valor



Objetos en VHDL
Señales (cont.)

- **Sintaxis de la declaración :**

```
SIGNAL signal_name : type_name [:= value];
```

- **Ejemplo declaración y asignación:**

```
SIGNAL brdy : BIT;
brdy <= '0' AFTER 5ns, '1' AFTER 10ns;
```

Señales y Variables

- **Este ejemplo resalta las diferencias entre señales y variables**

```
ARCHITECTURE test1 OF mux IS
    SIGNAL x : BIT := '1';
    SIGNAL y : BIT := '0';
BEGIN
    PROCESS (in_sig, x, y)
      BEGIN
        x <= in_sig XOR y;
        y <= in_sig XOR x;
      END PROCESS;
END test1;
```

```
ARCHITECTURE test2 OF mux IS
    SIGNAL y : BIT := '0';
BEGIN
    PROCESS (in_sig, y)
      VARIABLE x : BIT := '1';
    BEGIN
      x := in_sig XOR y;
      y <= in_sig XOR x;
    END PROCESS;
END test2;
```

- **Asumiendo una transición de 1 a 0 sobre la señal *in_sig*, Cuales son los resultados para y en ambos casos?**

Objetos en VHDL
Señales frente a variables

- **La clave entre variables y señales es tiempo de retardo en la asignación**

```
ARCHITECTURE sig_ex OF test IS
    PROCESS (a, b, c, out_1)
    BEGIN
      out_1 <= a NAND b;
      out_2 <= out_1 XOR c;
    END PROCESS;
END sig_ex;
```

Time	a	b	c	out_1	out_2
0	0	1	1	1	0
1	1	1	1	1	0
1+d	1	1	1	0	0
1+2d	1	1	1	0	1

Objetos en VHDL

Señales frente a variables(Cont.)

```
ARCHITECTURE var_ex OF test IS
BEGIN
    PROCESS (a, b, c)
    VARIABLE out_3 : BIT;
    BEGIN
      out_3 := a NAND b;
      out_4 <= out_3 XOR c;
    END PROCESS;
END var_ex;
```

Time	a	b	c	out_3	out_4
0	0	1	1	1	0
1	1	1	1	0	0
1+d	1	1	1	0	1

Objetos en VHDL
Ficheros

- Files suministra una forma de comunicación entre el diseño en VHDL y el servidor o PC
- La declaración hace que un fichero esté disponible en el diseño
- Los ficheros pueden ser abiertos para escritura o lectura
- El Paquete STANDARD define las rutinas básicas I/O para trabajar con ficheros
- El Paquete TEXTIO define rutinas para manejar ficheros de texto

Secuencial frente a concurrente

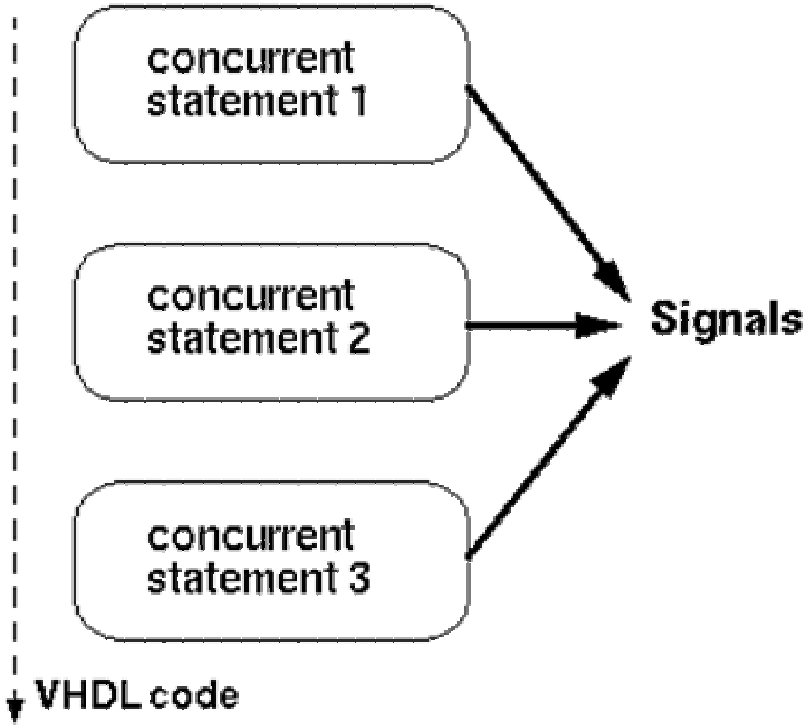
- **VHDL es inherentemente un lenguaje concurrente**
  - **Todos los procesos VHDL son ejecutados concurrentemente**
  - **Las asignaciones Concurrentes de señales son procesos**
- **Las declaraciones en VHDL que se ejecutan secuencial mente están dentro de un proceso *process***

Sentencias Concurrentes

- La unidad básica de concurrencia es el proceso
- Mecanismos para alcanzar concurrencia:
  - Los procesos se comunican con otro a través de las señales
  - La asignación de señales requiere retardo antes de que su nuevo valor sea asumido
  - El tiempo de simulación avanza cuando todos los procesos activos sean completados
- Las construcciones concurrentes en VHDL son:
  - Bloque, proceso, assert, asignación de señales, Llamadas a procedimientos, instanciación de componentes

Sentencias Concurrentes

- **Las sentencias concurrentes son ejecutadas al mismo tiempo independientemente del orden en el que aparezcan**



Sentencias Concurrentes
Asignación condicional de señales

- **SEÑAL <= VALOR;**
- **SEÑAL <= VALOR _1 *when* CONDICION_1 *else***
- **VALOR _2 *when* CONDICION_2 *else***
- **. . .**
- **VALOR _n;**

CONDICION Es una expresión boleana

Equivalente a la construcción *if* ..., *elsif* ..., *else*

Sentencias Concurrentes
Asignación condicional de señales : Ejemplo

```
entity condi_asig is
port (A, B, C, X : in   bit_vector (3 downto 0);
      Z_CONC : out bit_vector (3 downto 0);
      Z_SEQ    : out bit_vector (3 downto 0));
end condil_asig;

architecture Ejemplo of condi_asig is

begin

  -- versión Concurrente

  Z_CONC <= B when X = "1111" else
            C when X > "1000" else
            A;

end Ejemplo;
```

⇔

```
Vesión secuencial
process (A, B, C, X)
 begin
   if  (X = "1111") then
     Z_SEQ <= B
   elsif  (X > "1000") then
     Z_SEQ <= C;
   else
     Z_SEQ <= A;
   end if;
 end process;
```

Sentencias Concurrentes
Asignación de señal con selección

- **with EXPRESION select**
- **SIGNAL<= VALOR_1 when ELECCION_1,**
- **VALOR _2 when ELECCION _2 | ELECCION _3,**
- **VALOR _3 when ELECCION _4 to ELECCION _5,**
- **. . .**
- **VALOR _n when others;**

Las ELECIONES no se deben solapar

Todas las elecciones deben ser cubiertas

Valores simples, rango de Valores

operaciones lógicas ("|" significa "or")

"when others" cubre las restantes opciones

Equivalente a la construcción *case ..., when ...*

Sentencias Concurrentes
Asignación de señal con selección
Ejemplo

```
-- Version Concurrente
  with X select
     Z_CONC <= A when 0,
                  B when 7 | 9,
                  C when 1 to 5,
                  0 when others;
```

⇔

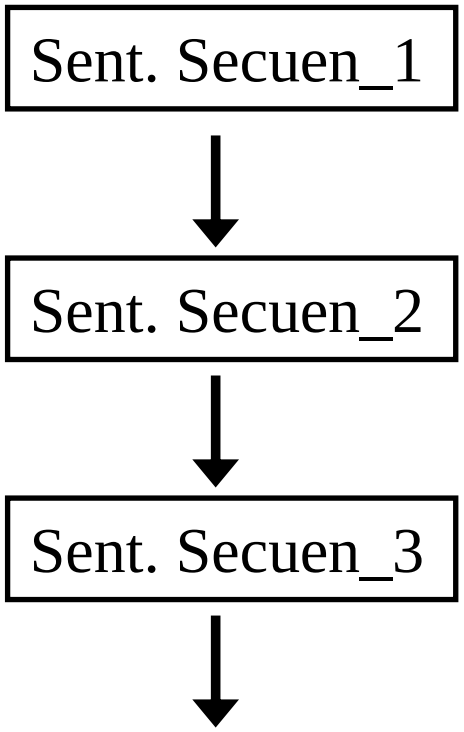
```
-- Versión secuencial
  process (A, B, C, X)
  begin
     case X is
        when 0        => Z_SEQ <= A;
        when 7 | 9    => Z_SEQ <= B;
        when 1 to 5   => Z_SEQ <= C;
        when others => Z_SEQ <= 0;
  end process;
```

Sentencias Concurrentes
Resumen

- La asignación concurrente describe funcionalidad de multiplexores
  - Asignación condicional: Las decisiones son basadas sobre varias señales
  - Asignación con selección: Las decisiones son basadas envalores de una simple señal
- Equivalencias
  - Asignación condicional <=> *if ..., elsif ..., else ..., end if*
  - Asignación con selección <=> *case ..., when ..., end case*

Sentencias Secuenciales

- **Todas las declaraciones que están dentro de un *process* se ejecutan secuencialmente**
- **Son ejecutadas de acuerdo al orden en el que aparezcan tal como los lenguajes de programación**




```
ARCHITECTURE secuencial OF test_mux IS
BEGIN
    select_proc : PROCESS (x,y)
    BEGIN
    IF (select_sig = '0') THEN
        z <= x;
    ELSIF (select_sig = '1') THEN
        z <= y;
    ELSE
        z <= "XXXX";
    END IF;
    END PROCESS select_proc;
END sequential;
```

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Sentencias Secuenciales
Sentencia IF

```
if CONDICION then
-- Decla. secuenciales
end if;
```

```
if CONDICION then
-- Decla. secuenciales
Else
-- Decla. secuenciales
end if;
```

```
if CONDICION then
   -- Decla. secuenciales
elsif CONDICION then
   -- Decla. secuenciales
. . .
else
   -- Decla. secuenciales
end if;
```

- **CONDICION es una expresión boleana**
- **La cláusula *elsif***
  - **Es opcional e impone prioridad**
- **La cláusula *else***
  - **Opcional y es ejecutado si todas las condiciones son falsas**

Sentencias Secuenciales Ejemplos

```
-- Ejemplo1
process (A, B, C, X)
   begin
      Z <= A;
      if  (X = "1111") then
         Z <= B;
      elsif  (X > "1000") then
         Z <= C;
      end if;
   end process;
end EXAMPLE1;
```

```
--Ejemplo 2
process (A, B, C, X)
   begin
      if  (X = "1111") then
         Z <= B;
      elsif  (X > "1000") then
         Z <= C;
      else
         Z <= a;
      end if;
   end process;
```

Sentencias Secuenciales
Sentencia Case

```
case  EXPRESION is

  when  VALOR_1 =>
    -- Decla Secuenciales

  when VALOR _2 | VALOR _3  =>
    -- Decla Secuenciales

  when VALOR _4 to VALOR _N =>
    -- Decla Secuenciales

  when  others =>
    -- Decla Secuenciales

end  case ;
```

Las opciones no deben solaparse

Todas las opciones deben ser cubiertas

Valores, simples o rango

("|" significa "or")

"when others" Cubre el resto opciones

Sentencias Secuenciales
sentencia case ejemplo

```
process (A, B, C, X)
begin
  case  X is
    when  0 =>          Z <= A;
    when  7 |  9 =>     Z <= B;
    when  1 to 5 =>     Z <= C;
    when others =>      Z <= 0;
  end case;
end process;
```

Sentencias Secuenciales
sentencia For loops

```
entity FOR_LOOP is
   port (A : in    integer range 0 to 3;
         Z  : out bit_vector (3 downto 0));
end FOR_LOOP;
```

El parámetro del Loop esta declarado implícitamente

No puede ser declarado externamente

Su acceso es solo de lectura

Recorre todos los valores dentro del rango especificado Enteros enumerados

```
architecture Ejemplo of FOR_LOOP is
begin
   process (A)
   begin
      Z <= "0000"; -- Asigna. por defecto
      for  I in  0 to 3 loop
         if (A = I) then
            Z(I) <= `1`;
         end if;
      end loop;
   end process;
end Ejemplo;
```

Sentencias Secuenciales
sentencia sintaxis Loop

```
[LOOP_etiqueta :]for INTIFICADOR in RANGO_DISCRE loop
   -- Declaraciones Secuen.
end loop [LOOP_LABEL];
```

```
[LOOP_ etiqueta :] while CONDICION loop
   -- Declaraciones Secuen.
end loop [LOOP_ etiqueta ];
```

Etiqueta opcional pero facilita la legibilidad

FOR loop identificador , no es declarado, Solo lectura, Solo visible dentro el loop

Síntesis: Loop for debe tener rango fijo y - 'while' usualmente no es sintetizable

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Sentencias Secuenciales loop ejemplos 1

```
entity CONV_INT is
    port (VECTOR: in
            bit_vector(7 downto 0);
          RESULT:  out integer);
 end CONV_INT;
```

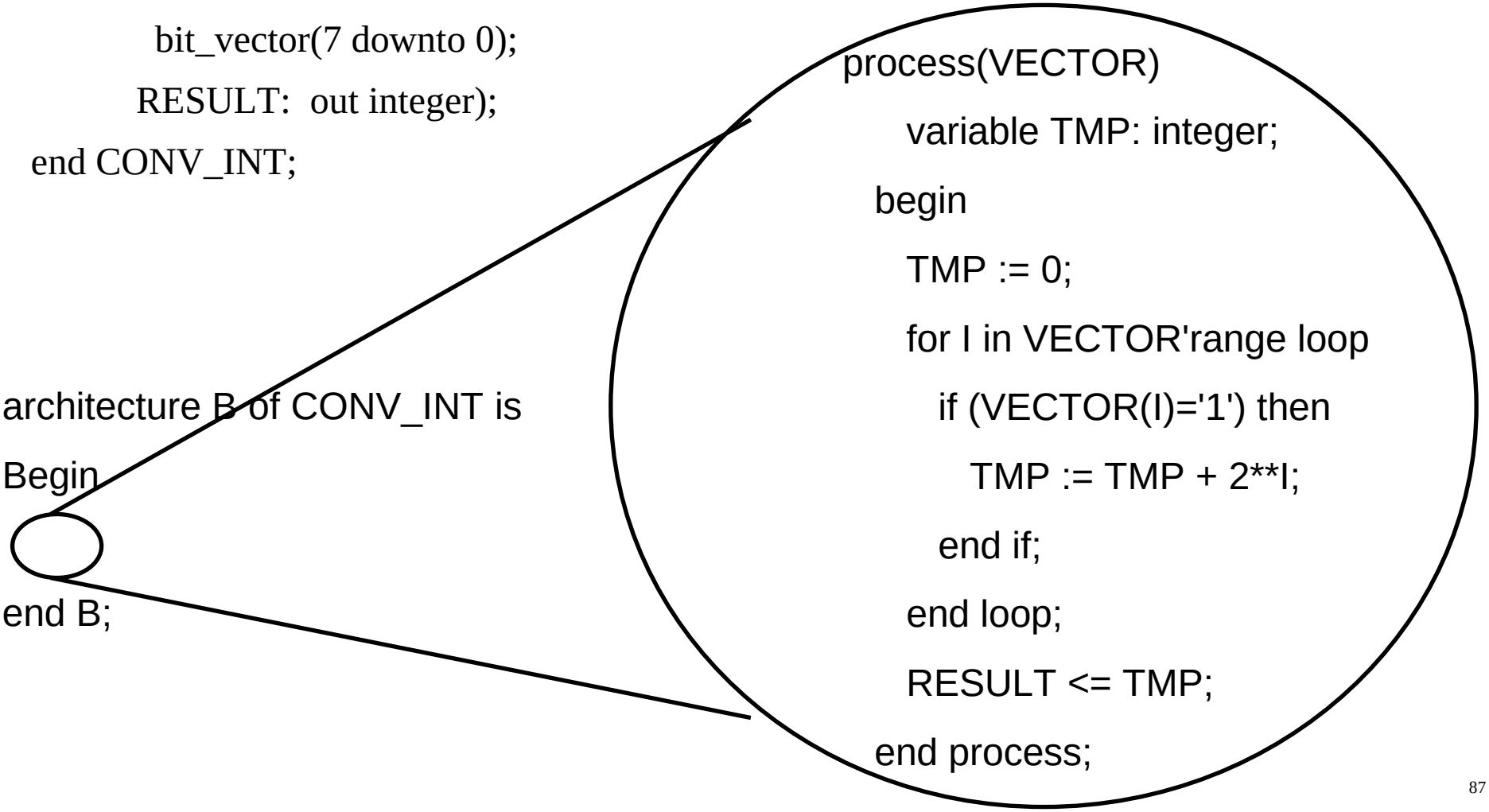
```
architecture A of CONV_INT is
Begin
end A;
```

```
process(VECTOR)
    variable TMP: integer;
begin
    TMP := 0;
    for I in 7 downto 0 loop
      if (VECTOR(I)='1') then
        TMP := TMP + 2**I;
      end if;
    end loop;
    RESULT <= TMP;
end process;
```

Sentencias Secuenciales loop ejemplos 2

```
entity CONV_INT is
    port (VECTOR: in
            bit_vector(7 downto 0);
          RESULT:  out integer);
  end CONV_INT;
```

```
architecture B of CONV_INT is
Begin
```



```
end B;
```

```
process(VECTOR)
    variable TMP: integer;
begin
    TMP := 0;
    for I in VECTOR'range loop
      if (VECTOR(I)='1') then
        TMP := TMP + 2**I;
      end if;
    end loop;
    RESULT <= TMP;
end process;
```

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Sentencias Secuenciales loop ejemplos 3

```
entity CONV_INT is
    port (VECTOR: in
            bit_vector(7 downto 0);
          RESULT:  out integer);
  end CONV_INT;
```

```
architecture C of CONV_INT is
Begin

end C;
```

```
process(VECTOR)
    variable I, TMP: integer;
begin
    TMP := 0;
    I := VECTOR'high;
    while (I >= VECTOR'low) loop
      if (VECTOR(I)='1') then
        TMP := TMP + 2**I;
      end if;
      I := I - 1;
    end loop;
    RESULT <= TMP;
end process;
```

Sentencias Secuenciales
Sentencia wait

'*wait*' La declaración para la ejecución del proceso
El proceso continuará cuando la instrucción se cumpla

Tipos diferentes de *wait*:

wait Por un tiempo específico — *Wait for Tiempo especifico*

wait Por un evento de una señal — *Wait on Lista de señales*

wait Por que una condición sea Cierta — *Wait Until Condición*

(requires an event)

Si está indefinido el proceso nunca se reactivará

Wait: La declaración no puede ser usada con la lista de sensibilidad en el proceso

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Sentencias Secuenciales
Sentencia wait ejemplo

```
entity FF is
    port (D, CLK : in   bit;
          Q      : out bit);
end FF;

architecture BEH_1 of FF is
Begin
    process
    begin
        wait on CLK;
        if (CLK = '1') then
            Q <= D;
        end if;
    end process;
end BEH_1;
```

Sentencias Secuenciales
Sentencia wait ejemplo

```
entity FF is
    port (D, CLK : in   bit;
          Q      : out bit);
end FF;

architecture BEH_2 of FF is
Begin
end BEH_2;
```

```
process
  begin
    wait until CLK=`1`;
    Q <= D;
end process;
```

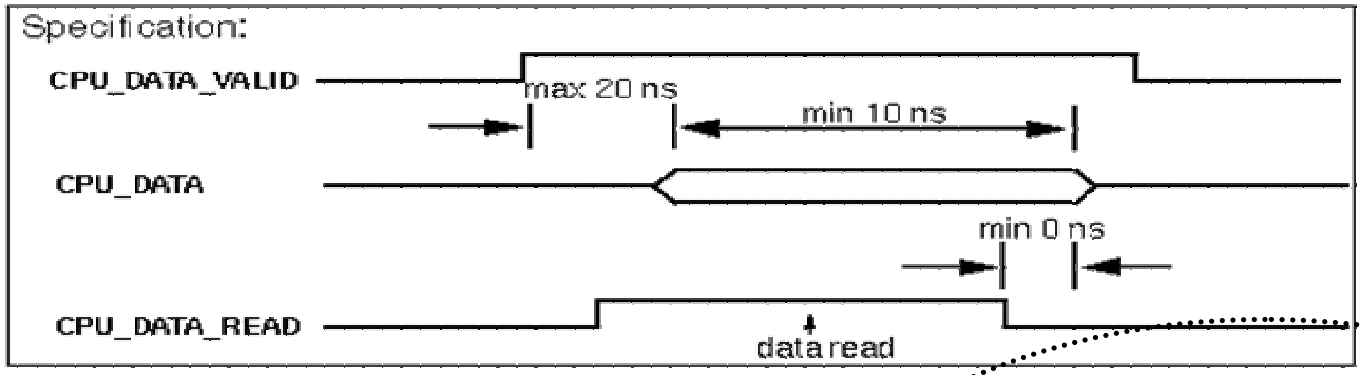
Sentencias Secuenciales
Sentencia wait ejemplo

```
STIMULUS: process
begin
    SEL     <= `0`;
    BUS_B <= "0000";
    BUS_A <= "1111";
    wait for 10 ns;
    SEL <= `1`;
    wait for 10 ns;
    SEL <= `0`;
    wait for 10 ns;
    wait;
end process STIMULUS;
```

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Sentencias Secuenciales
Sentencia wait y modelado de comportamiento

El comportamiento temporal puede trasladarse directamente a VHDL




```
READ_CPU : process
Begin
        wait until CPU_DATA_VALID = `1`;
        CPU_DATA_READ <= `1`;
        wait for 20 ns;
        LOCAL_BUFFER <= CPU_DATA;
        wait for 10 ns;
        CPU_DATA_READ <= `0`;
end process READ_CPU;
```

Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

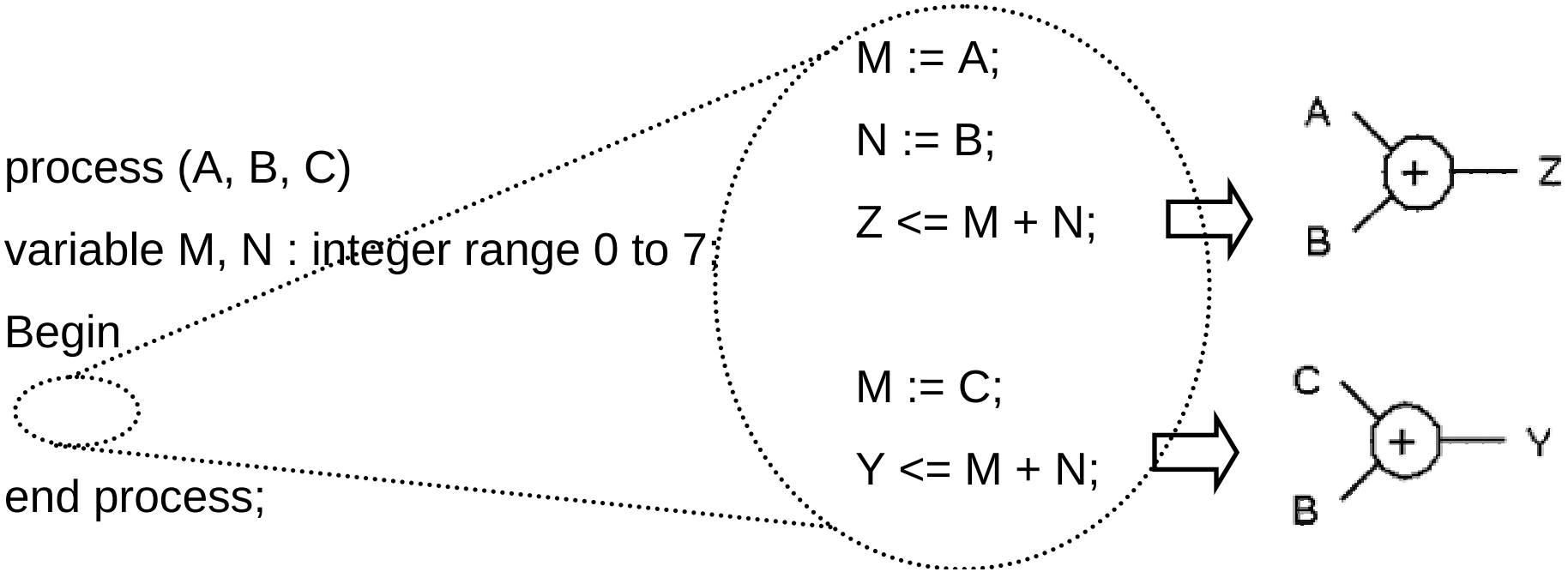
Sentencias Secuenciales Variables

Las Variables están disponibles en los procesos solamente

- Declaradas en la parte declarativa y Visibles solo en el proceso
- Asignación inmediata y Mantiene el ultimo valor

Asignaciones posibles (Los tipos deben coincidir)

- Señal a variable
- variable a señal



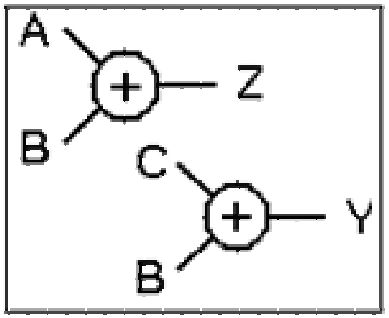
Laboratorio de Comunicaciones y Teledetección
Dpto. Física Fundamental y Experimental, Electrónica y Sistemas

Sentencias Secuenciales
Señales frente a Variables

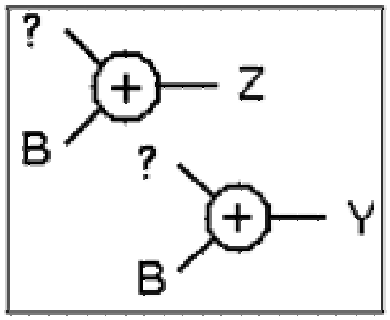
Los Valores de las Señales son asignados después de la ejecución

La última asignación es realizada M<= A es sobre escrita por M <= C;

```
process (A, B, C)
        variable M, N : integer;
    begin
        M :=  A;
        N :=  B;
        Z <= M + N;
        M :=  C;
        Y <= M + N;
end process;
```

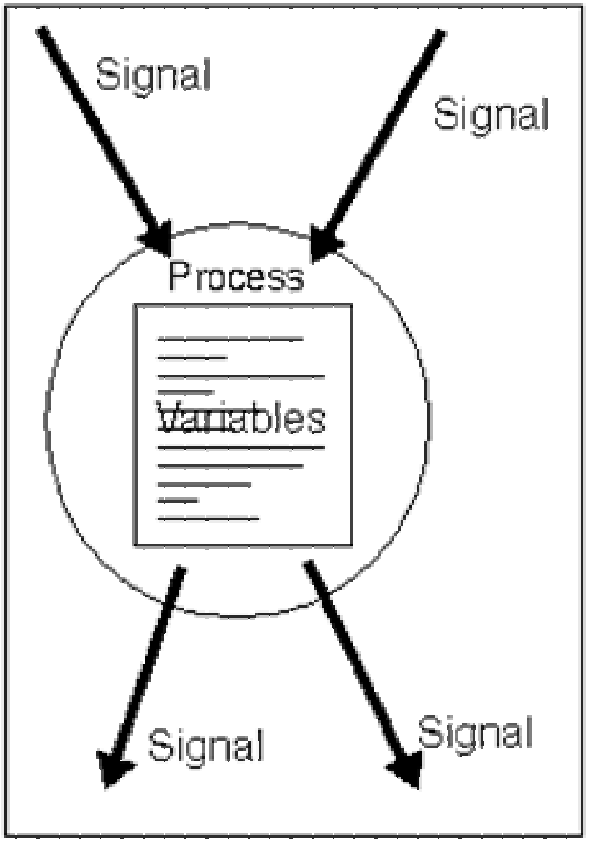



```
signal A, B, C, Y, Z, M,N : integer;
process (A, B, C, M, N)
    begin
        M <=  A;
        N <=  B;
        Z <= M + N;
        M <=  C;
        Y <= M + N;
end process;
```



Sentencias Secuenciales Uso de las Variables

- Para resultados intermedios en la implementación de algoritmos
  - Asignación de señal a variable
  - Ejecución del algoritmo
  - Asignación de Variable a señal
- Variables almacena su valor hasta la próxima llamada del proceso



Paquetes y Librerías

- **Las Construcciones de usuario dentro de arquitecturas y entidades no son visibles en otros componentes de VHDL**
  - **El marco o ámbito de subprogramas, definiciones de los usuarios: tipos de datos, constantes y señales están limitados a los componentes de VHDL en los cuales se han hecho**
  - **Los *Paquetes y librerías* suministran un mecanismo para poder re-usar construcciones ya hechas**
  - **Cosas declaradas en paquetes pueden ser usadas (incluidas) en otros componentes de VHDL**

Paquetes

- Los Paquetes tienen dos partes
  - Declaración del Paquete – Contiene las declaraciones de los objetos definidos en el paquete
  - Cuerpo del Paquete – Contiene las definiciones necesarias para ciertos objetos definidos en la declaración del paquete
    - Ejemplo descripción de un subprograma
- Ejemplos de cosas en VHDL incluidos en un Paquete :
  - Declaraciones Básicas
    - Tipos, subtipos
    - Constantes
    - Subprogramas
    - Cláusula Use
  - Declaración de señales
  - Declaración de atributos
  - Declaración de componentes

Paquetes
Declaración

- **Ejemplo e la declaración de un Paquete:**

```
PACKAGE mi_material IS
    TYPE binary IS ( ON, OFF );
    CONSTANT PI : REAL := 3.14;
    CONSTANT My_ID : INTEGER;
    PROCEDURE add_bits3(SIGNAL a, b, en : IN BIT;
                SIGNAL temp_result, temp_carry : OUT BIT);
END mi_material;
```

- **Notar que algunas cosas solo requieren declaración mientras otras necesitan información adicional detallada en el cuerpo**
  - **Par definición de tipos y subtipos, La declaración es suficiente**
  - **Los subprogramas requiere declaración y descripción**

Paquetes
Paquete Cuerpo (Body)

- **Incluye la información necesaria (Descripción Funcional) para los objetos declarados en la declaración del paquete he**
  - **Ejemplo Descripción de subprogramas, asignación de constantes**

```
PACKAGE BODY mi_material IS
   CONSTANT My_ID : INTEGER := 2;

   PROCEDURE add_bits3(SIGNAL a, b, en : IN BIT;
            SIGNAL temp_result, temp_carry : OUT BIT) IS
   BEGIN    -- this function can return a carry
      temp_result <= (a XOR b) AND en;
      temp_carry <= a AND b AND en;
   END add_bits3;
END mi_material;
```

Paquetes
Cláusula Use

- **Los Paquetes tienen que ser hechos visibles antes de que su contenidos puedan ser usados**
  - **La Cláusula USE hace el que el paquete sea visible para las entidades, arquitecturas, y otros paquetes**

```
--Usa solamente las declaraciones binary y
--add_bits3
USE mi_material.binary, mi_material.add_bits3;

... ENTITY declaración...
... ARCHITECTURE declaración...
```

```
--Usa todas las declaraciones en paquete mi_material
USE mi_material.ALL;

... ENTITY declaración...
... ARCHITECTURE declaración...
```

Librerías

- Son Análogas a los directorios de ficheros
  - Una librería de VHDL contiene (analizado o compilado) entidades, arquitecturas, y paquetes
- Facilita la administración y revisión
- Las librerías se acceden vía un nombre lógico asignado
  - La unidad del diseño actual es compilado dentro la librería *Work*
  - Ambas librerías *Work* y *STD* están siempre visibles
  - Existen muchas mas librerías suministradas por el vendedor del simulador de VHDL
    - Ejemplo privadas y estándar (IEEE).

```
Library IEEE;
use IEEE.STD_Logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

```
library LPM;
use LPM.lpm_components.ALL;
```

Atributos

- Los atributos dan información acerca de ciertas cosas en VHDL
  - Ejemplo. types, subtypes, procedures, functions, signals, variables, constants, entities, architectures, configurations, packages, components
  - Sintaxis General :

```
nombre'atributo_identicador
```

- VHDL Tiene deferentes ya predefinidos:
  - X'EVENT -- TRUE Cuando hay un evento sobre la Señal X
  - X'LAST_VALUE – retorna el valor previo de la señal X
  - Y'HIGH – retorna el valor más alto del rango de Y
  - X'STABLE(t) -- TRUE Cuando no ocurren eventos sobre la señal X en el tiempo pasado t

Atributos
Register Example

- The following example shows how attributes can be used to make an 8-bit register
- Especificaciones :
  - Triggers on rising clock edge
  - Latches only on enable high
  - Has a data setup time of x_setup
  - Has propagation delay of prop_delay

```
ENTITY 8_bit_reg IS
    GENERIC (x_setup, prop_delay : TIME);
    PORT(enable, clk : IN qsim_state;
         a : IN qsim_state_vector(7 DOWNTO 0);
         b : OUT qsim_state_vector(7 DOWNTO 0));
END 8_bit_reg;
```

- qsim_state type is being used - includes logic values 0, 1, X, and Z

Atributos
Register Example (Cont.)

- The following architecture is a first attempt at the register
- The use of 'STABLE is to used to detect setup violations in the data input

```
ARCHITECTURE first_attempt OF 8_bit_reg IS
BEGIN
  PROCESS (clk)
  BEGIN
  IF (enable = '1') AND a'STABLE(x_setup) AND
         (clk = '1') THEN
              b <= a AFTER prop_delay;
  END IF;
  END PROCESS;
END first_attempt;
```

- What happens if *a* does not satisfy its setup time requirement of *x_setup*?

Atributos

Register Example (Cont.)

- The following architecture is a second and more robust attempt
- The use of 'LAST_VALUE ensures the clock is rising from a value of ‘0’

```
ARCHITECTURE behavior OF 8_bit_reg IS
BEGIN
  PROCESS (clk)
  BEGIN
  IF (enable = '1') AND a'STABLE(x_setup) AND
        (clk = '1') AND (clk'LAST_VALUE = '0') THEN
             b <= a AFTER prop_delay;
  END IF;
  END PROCESS;
END behavior;
```

- An ELSE cláusula could be added to define the behavior when the requirements are not satisfied

Operadores

- **Operators can be chained to form complex expressions, e.g. :**

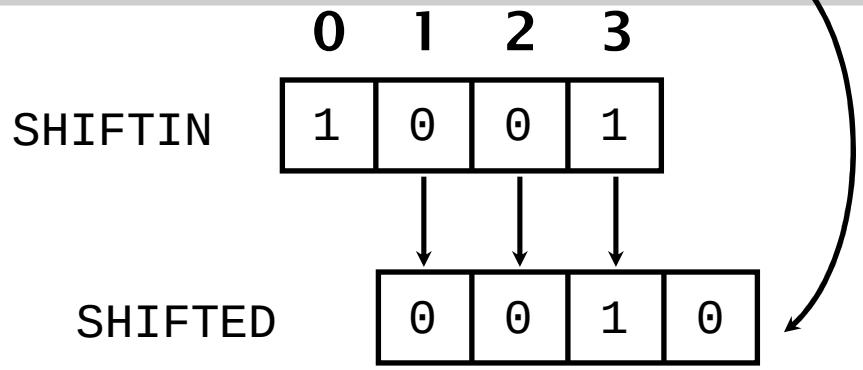
```
res <= a AND NOT(B) OR NOT(a) AND b;
```

  - **Can use parentheses for readability and to control the association of operators and operands**
- **Defined precedence levels in decreasing order :**
  - **Miscellaneous operators -- **, abs, not**
  - **Multiplication operators -- *, /, mod, rem**
  - **Sign operator -- +, -**
  - **Addition operators -- +, -, &**
  - **Shift operators -- sll, srl, sla, sra, rol, ror**
  - **Relational operators -- =, /=, <, <=, >, >=**
  - **Logical operators -- AND, OR, NAND, NOR, XOR, XNOR**

Operadores
Examples

- **The concatenation operator &**

```
VARIABLE shifted, shiftin : BIT_VECTOR(0 TO 3);
...
shifted := shiftin(1 TO 3) & '0';
```



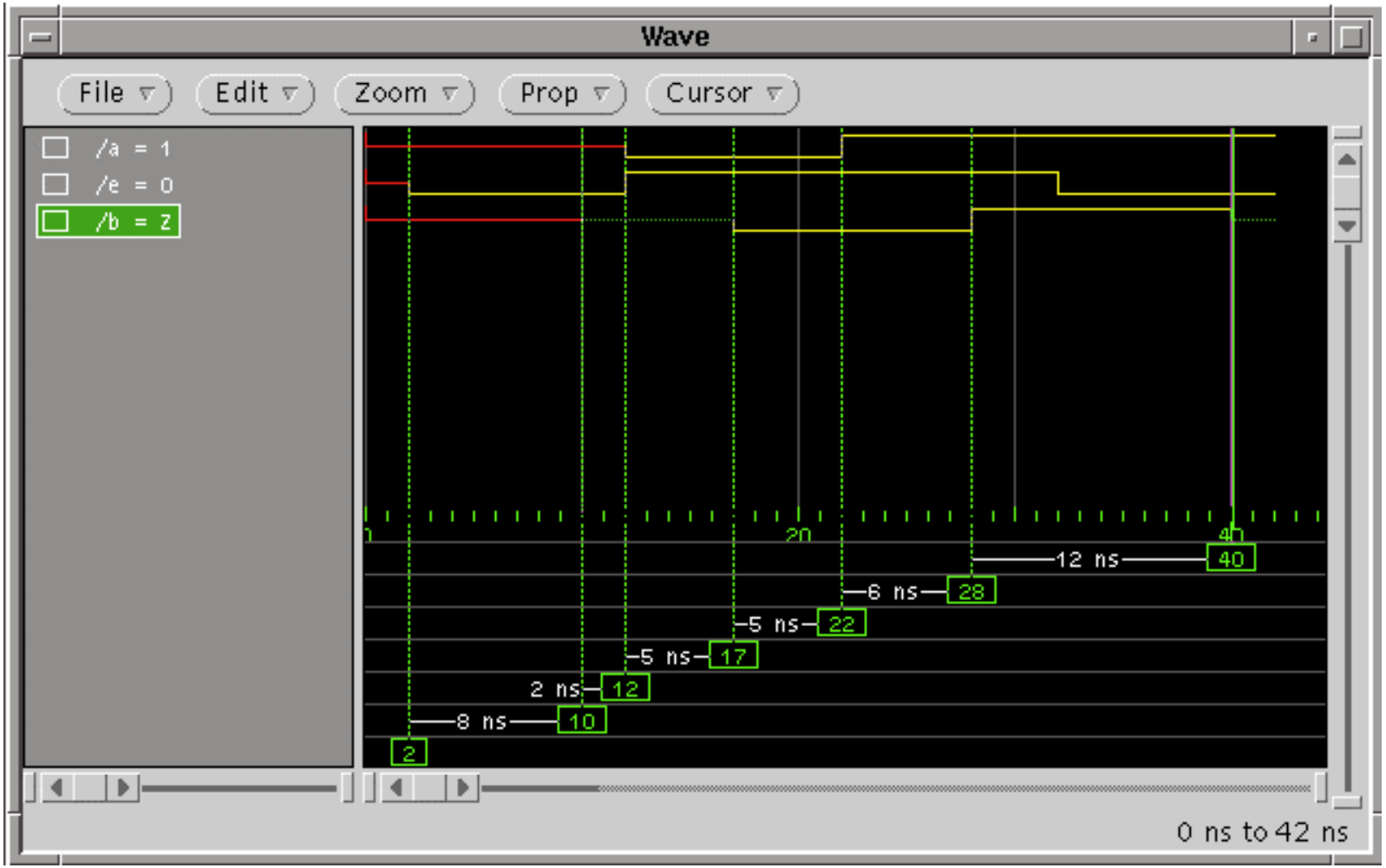
- **The exponentiation operator ****

```
x := 5**5  -- 5^5, OK
y := 0.5**3 -- 0.5^3, OK
x := 4**0.5 -- 4^0.5, Illegal
y := 0.5**(-2) -- 0.5^(-2), OK
```

Modulo Outline

- Introduction
- VHDL Design Example
- VHDL Model Components
- Basic VHDL Constructs
- **Examples**
- Summary

Tri-State Buffer Simulation Results



Summary

- VHDL is a worldwide standard for the description and modeling of digital hardware
- VHDL gives the designer many different ways to describe hardware
- Familiar programming tools are available for complex and simple problems
- Sequential and concurrent modes of execution meet a large variety of design needs
- Packages and libraries support design management and component reuse

References

[Bhasker95] Bhasker, J. A VHDL Primer, Prentice Hall, 1995.

[Calhoun95] Calhoun, J.S., Reese, B.,. *"Class Notes for EE-4993/6993: Special Topics in Electrical Engineering (VHDL)"*, Mississippi State University, http://www.erc.msstate.edu/, 1995.

[Coelho89] Coelho, D. R., The VHDL Handbook, Kluwer Academic Publishers, 1989.

[Gajski83] Gajski, Daniel D. and Kuhn, Robert H., "Guest Editors Introduction - New VLSI Tools", IEEE Computer, pp 11-14, IEEE, 1983; © IEEE 1983

[Hein98] Hein, et al, "VHDL Modeling Terminology and Taxonomy," Version3.0, July 29, 1998.

[IEEE] All referenced IEEE material is used with permission.

[Lipsett89] Lipsett, R., C. Schaefer, C. Ussery, VHDL: Hardware Description and Design, Kluwer Academic Publishers, , 1989.

[LRM93] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993.

[Navabi93] Navabi, Z., VHDL: Analysis and Modeling of Digital Systems, McGraw-Hill, 1993.

[Menchini94] Menchini, P., "*Class Notes for Top Down Design with VHDL*", 1994.

[MG90] An Introduction to Modeling in VHDL, Mentor Graphics Corporation, 1990.

[MG93] Introduction to VHDL, Mentor Graphics Corporation, 1993.

[Perry94] Perry, D. L., VHDL, McGraw-Hill, 1994.

[Richards97] Richards, M., Gadient, A., Frank, G., eds. *Rapid Prototyping of Application Specific Signal Processors*, Kluwer Academic Publishers, Norwell, MA, 1997

[Smith88] Smith, David, "What is Logic Synthesis", *VLSI Design and Test*, October, 1988.

References, cont.

[USE/DA94] USE/DA Standards Survey, 1994.

[VI93] VHDL International Survey, 1993.

[Walker85] Walker, Robert A. and Thomas, Donald E., "A Model of Design Representation and Syntheses", 22nd Design Automation Conference, pp. 453-459, IEEE, 1985

[Waxman89A] Waxman, R., Saunders, L.F., and Carter, H.C., “Abolishing the Tower of Babel,” Spectrum, Vol. 26, Number 5, May 1989, pp. 40-44.

[Waxman89B] R. Waxman and L. Saunders, The Evolution of VHDL, Invited Paper, INFORMATION PROCESSING '89, G.X. Ritter (ed.), Elsevier Science Publishers B.V. (North Holland), copyright IFIP, 1989, PP. 735-742.

[Williams94] Williams, R. D., "Class Notes for EE 435: Computer Organization and Design", University of Virginia, http://www.ee.virginia.edu/research/CSIS/, 1994.