

# Introducción a VHDL

## Parte I

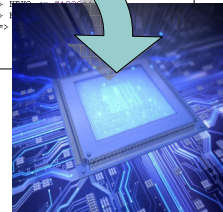
Martín Vázquez - Lucas Leiva  
Mar 2013



Universidad Nacional del Centro  
de la Provincia de Buenos Aires

## VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
  
ENTITY LabEx03 IS  
  PORT( c : IN STD_LOGIC_VECTOR(2 DOWNTO 0);  
        HEXO : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));  
END LabEx03;  
  
ARCHITECTURE Behavior OF LabEx03 IS  
BEGIN  
  PROCESS(c)  
  BEGIN  
    CASE c IS  
      WHEN "000" => HEXO <= "000000";  
      WHEN "001" => HEXO <= "000001";  
      WHEN "010" => HEXO <= "000010";  
      WHEN "011" => HEXO <= "000011";  
      WHEN OTHERS => HEXO <= "000000";  
    END CASE;  
  END PROCESS;  
END Behavior;
```



- Lenguaje de alto nivel para el modelado de circuitos.
- Alternativa al uso de esquemáticos.



## Historia del lenguaje VHDL



- Propuesto por el DoD de EEUU (proyecto VHSIC) en 1980.
- Primera versión en 1985.
- Estandarizado por IEEE en 1987.
- Primer éxito: Programa F-22 (1986-1993).
- Revisiones: 1993, 2000, 2002, 2008.



F-22

### En números:

- U\$S 17M desarrollo del lenguaje
- U\$S 16M desarrollo de herramientas



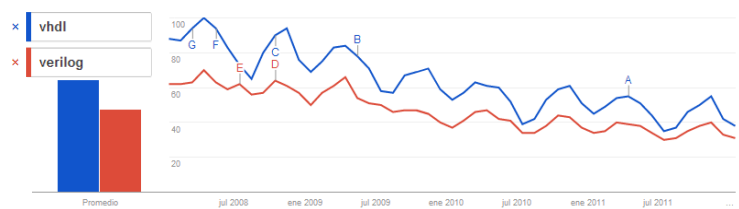
## Comparación con otros lenguajes

- Similitudes
  - VHDL es un lenguaje estructurado.
  - Reutilización de módulos.
  - Portable.
- Diferencias
  - El paradigma es diferente: **orientado a procesos**.
  - Las sentencias no siempre son ejecutadas secuencialmente.
  - En VHDL la información temporal es explícita.
  - VHDL no se compila en un ejecutable, sino que se sintetiza en un circuito digital.



## Diseñadores VHDL

- o Empresas de tecnología avanzada
  - AT&T, USA-IBM, INTEL, Texas Instruments.
- o Europa.
- o Latinoamérica.
- o Korea (~75%).



Fuente: Google Trends, 2012

Introducción a VHDL - Parte I

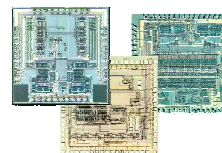
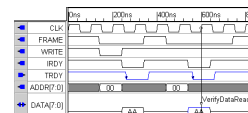
5



## Uso de VHDL

- o Documentar diseños.
- o Especificar requerimientos.
- o Simular.
  - Comprobar funcionalidad.
  - Detectar errores.
- o Sintetizar.
  - Implementación hardware del diseño.

```
43 ARCHITECTURE fnc OF light_control_vhdl IS
44
45     TYPE STATE_TYPE IS (
46         rest_state,
47         data_stop,
48         request,
49         request_ack
50     );
51
52     -- Declare constant and port state signals
53     CONSTANT max_count : INTEGER := 100;
54     SIGNAL rest_state : STATE_TYPE;
55 END fnc;
```



Introducción a VHDL - Parte I

6

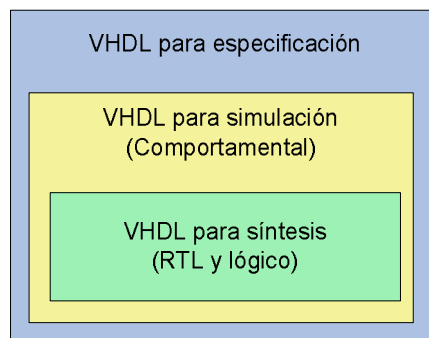


## Ventajas de VHDL

- Lenguaje estandarizado.
  - Soporte gubernamental e industrial.
  - Portable.
- Permite el diseño modular y jerárquico.
  - Reusabilidad.
- Posee una metodología de diseño.
- Admite diferentes niveles de abstracción.
  - Nivel Comportamental.
  - RTL (Register-Transfer-Level).
  - Nivel Lógico o de compuertas.



## Descripción en VHDL



Es conveniente describir en VHDL en el nivel de abstracción mas alto que sea sintetizable (RTL).



## Estructura de un diseño VHDL

- Un diseño en VHDL cuenta con dos unidades de diseño obligatorias
  - Entidad (**ENTITY**)
  - Arquitectura (**ARCHITECTURE**)

```
ENTITY nombre_entidad IS
```

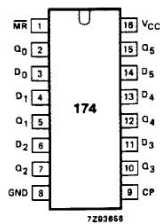
```
END nombre_entidad;
```

```
ARCHITECTURE nombre_arquitectura OF nombre_entidad IS
```

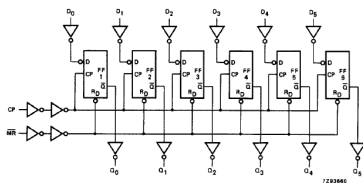
```
END nombre_arquitectura;
```



## Unidades de diseño VHDL: Entidad y arquitectura



- Una unidad hardware se visualiza como una “caja negra”.
  - Interfaz completamente definida.
    - Puertos (modo, tipo)
  - Interior oculto.
    - En VHDL “ENTITY”



- Descripción del diseño.
  - En VHDL “ARCHITECTURE”



## PORTS: Puertos de una entidad

### Ports = Canales de comunicación

- Un **nombre** que debe ser único dentro de la entidad
- Una lista de propiedades como:
  - La dirección del flujo de datos, entrada, salida, bidireccional y se conoce como **MODO** del puerto
  - los valores que puede tomar el puerto: `1`, `0` o (`Z`), etc., los valores posibles dependen de lo que se denomina **TIPO** de señal
- Los puertos son una clase especial de señales que adicionalmente al tipo de señal añade el modo



## Tipos

- VHDL es un lenguaje fuertemente tipado.
- Todos los objetos poseen un tipo en la declaración.
- Las asignaciones sólo pueden hacerse entre objetos del mismo tipo.
- Los tipos predefinidos son:
  - Escalares: *integer*, *floating point*, *enumerated*
  - Compuestos: *array*, *record*
  - Punteros: *access*
  - Archivos: *file*
- Tipo *Std\_logic*: valor presente en un cable de 1 bit
  - Declarado en *IEEE.standard\_logic\_1164*
  - Extension a bus: *Std\_logic\_vector*



## Tipos básicos

- **BIT**: sólo puede tomar los valores de `0` o `1`. Para modelar señales digitales.
- **BIT\_VECTOR**: array unidimensional (vector) de bits. Para modelar buses.
- **INTEGER**: tipo entero. Usado como valor índice en lazos, constantes o valores genéricos.
- **BOOLEAN**: tipo lógico. Puede tomar valor TRUE o FALSE.
- **REAL**: tipo para números en coma flotante.
- **ENUMERATED**: enumeración. Conjunto de valores definido por el usuario.

Ej: `type estados is (inicio, lento, rápido);`



## Tipo std\_logic

- Los dos valores del tipo `bit` no alcanzan para modelar todos los estados de una señal digital en la realidad
- El paquete `IEEE.standard_logic_1164` define el tipo `std_logic`; que representa todos los estados de una señal real
  - U** No inicializado, valor por defecto
  - X** Desconocido fuerte, salida con múltiples fuentes en corto
  - 0** Salida de una puerta con nivel lógico bajo
  - 1** Salida de una puerta con nivel lógico alto
  - Z** Alta impedancia
  - W** Desconocido débil, terminación del bus
  - L** 0 débil, resistencia de pull-down
  - H** 1 débil, resistencia de pull-up
  - No importa, usado como comodín para síntesis



## Tipo std\_logic

- Para describir buses `std_logic_vector` (un array de `std_logic`)
- `std_logic` y `std_logic_vector` son estándares industriales
- Todos los valores son válidos en un simulador VHDL, sin embargo solo: ``1'`, ``0'`, ``Z'`, ``L'`, ``H'` y ``-'` se reconocen para la síntesis
- En el paquete **IEEE.standard\_logic\_1164** aparecen otros dos tipos: `std_ulogic` y `std_ulogic_vector`. Son los mismos pero sin haber pasado por la función de resolución:
  - Esta función decide cuál debe ser el valor de la señal cuando tienen dos fuentes que le asignan valores distintos.
  - Por ejemplo, si una fuente asigna un ``1'` y la otra un ``L'`, la función de resolución dice que la señal se queda a ``1'`



## Arrays

Los vectores se pueden definir tanto en rangos ascendentes como descendentes

```
signal a: bit_vector (0 to 3); -- rango ascendente
```

```
signal b: bit_vector (3 downto 0); -- rango descendente
```

```
a <= "0101"; b <= "0101";
```

Produce como resultado

```
a(0) = `0'; a(1) = `1'; a(2) = `0'; a(3) = `1';
```

```
b(0) = `1'; b(1) = `0'; b(2) = `1'; b(3) = `0';
```

Una manera rápida de asignar valores a los vectores son los *aggregates*

```
a <= ( 0 => `1', 1 => c and d, others => `Z');
```





## Ports: Modos de un puerto

Indican la dirección y si el puerto puede leerse o escribirse dentro de la entidad

- **IN** Una señal que entra a la entidad y no sale. La señal puede ser leída pero no escrita
- **OUT** Una señal que sale de la entidad y no es usada internamente. La señal no puede ser leída dentro de la entidad
- **BUFFER** Una señal que sale de la entidad y también es realimentada dentro de la entidad. Esta señal puede ser leída
- **INOUT** Una señal que es bidireccional entrada/salida de la entidad



## Descripción de la entidad: Ejemplo

```
LIBRARY ieee;  
USE ieee.std_logic_1164.ALL  
ENTITY mi_componente IS PORT (  
    clk, rst :      IN std_logic;  
    d:              IN std_logic_vector( 7 DOWNTO 0 );  
    q:              OUT std_logic_vector( 7 DOWNTO 0 );  
    co :            OUT std_logic;  
END mi_componente;
```

MODO

TIPO



## Estructura de un diseño VHDL (2)

- o Una arquitectura en VHDL se compone de:
  - Parte declarativa.
  - Cuerpo de la arquitectura.

```
ENTITY nombre_entidad IS
END nombre_entidad;
ARCHITECTURE nombre_arquitectura OF nombre_entidad IS
    declaraciones de señales (signals)
    declaraciones de componentes
BEGIN
    sentencias procesos (process)
    sentencias concurrentes
    instanciación de componentes
END nombre_arquitectura;
```

Introducción a VHDL - Parte I

19



## Señal (*signal*)

- o Modela un cable.
- o Puede conectar puertos (E/S) de entidades y “construcciones básicas” dentro de la arquitectura.
- o Se declara **en la parte declarativa de la arquitectura**.
- o **Su asignación ( <= ) no es inmediata**, sólo tiene efecto cuando avanza el tiempo

```
signal idle: std_logic <= `0';
```

Introducción a VHDL - Parte I

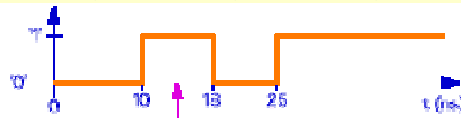
20



## Fundamentos: Conceptos de señal

- Las señales permiten comunicación entre procesos
- Cada señal posee una historia. Los valores pasados, presente y futuros (previstos) se almacenan en forma permanente.

```
s <= '1' after 10 ns, '0' after 18 ns, '1' after 25 ns;
```



tiempo actual de simulación

0	10	18	25
0	1	0	1

valores pasados

valor actual

valores futuros

Introducción a VHDL - Parte I

21

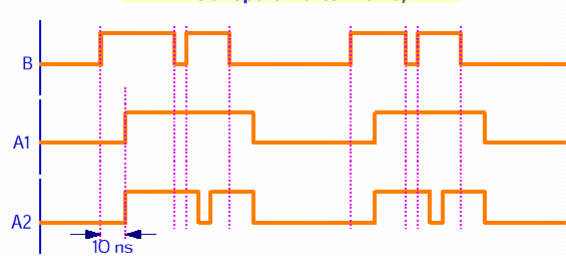


## Fundamentos: Tipos de retardo

- Las señales poseen dos tipos de retardo: **inercial** (por defecto) y **de transporte**

```
A1 <= B after 10 ns;
```

```
A2 <= transport B after 10 ns;
```



Introducción a VHDL - Parte I

22



## Operadores definidos

- o **Lógicos:** and, or, not, xor y xnor
- o **Relacionales:** = (igual), /= (distinto), < (menor), <= (menor o igual), > (mayor) y >= (mayor o igual)
- o **Adición:** + (suma), - (resta) y & (concatenación de vectores)
- o **Multiplicativos:** \* (multiplicación), / (división), rem (resto) y mod (módulo)
- o **Signos** (unarios): + , -
- o **Desplazamiento** (**bit\_vector**): sll, srl, sla, sra, rol y ror
- o **Misceláneos:** abs (valor absoluto), \*\* (exponenciación), not (negación unario)

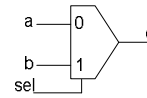


## Diseño concurrente en VHDL

- VHDL soporta la noción de “ejecución concurrente”
- Los métodos para describir la concurrencia dentro de una arquitectura son:
  - **Instanciaciones**
  - **Asignaciones concurrentes**
  - **Procesos.** Un proceso es un conjunto de sentencias secuenciales que se ejecutan según el orden.
- El orden relativo de procesos y asignaciones concurrentes dentro de una arquitectura no es relevante
- Las señales se utilizan para controlar la activación de los procesos



# Construcciones básicas



- **Procesos**

- Constituidos por instrucciones que se ejecutan y evalúan secuencialmente.

```
mux: process (a, b, sel)
begin
    if (sel='0') then c<=a;
    else c<=b;
    end if;
end process;
```

- **Sentencias concurrentes**

- En la arquitectura se encuentran fuera de los procesos y se evalúan concurrentemente (asignaciones concurrentes).

```
x <= b when sel='1' else
a;
```

- **Instancias de componentes** (otras entidades).

- Se evalúan concurrentemente.

```
MUX1: Mux port map (
    a => a,
    b => b,
    sel => sel,
    c => c);
```

Diseño jerárquico

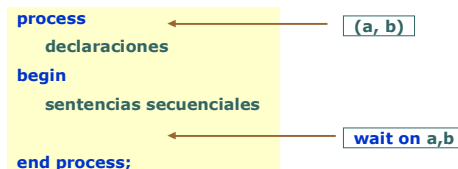


# Procesos: Ejecución

- Semánticamente un proceso es un bucle infinito de instrucciones secuenciales.
- El tiempo no avanza durante la ejecución del proceso.
- ¿Cómo avanza el tiempo si un proceso se activa?

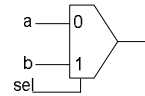
- **Instrucción wait**

- **wait on** a,b (cambio en a o b),
- **wait for** retraso (un cierto tiempo),
- **wait until** condición



- **Lista de sensibilidad:** conjunto de señales que activan el proceso cuando se produce un cambio en alguna de ellas.

# Procesos



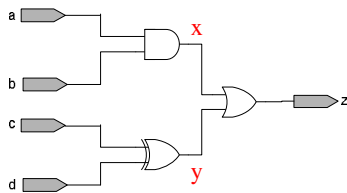
- Un proceso puede ser **Explícito** o **Implícito**.
- **Proceso Implícitos:** La lista de sensibilidad se obtiene de la propia expresión:
  - Asignaciones concurrentes.
  - Instanciación de componentes.
- **Proceso Explícitos:** La lista de sensibilidad se declara explícitamente.

```
x <= b when sel='1' else
a;
```

```
MUX1: Mux port map (
a => a,
b => b,
sel => sel,
c => c);
```

```
mux: process (a, b, sel)
begin
if (sel='0') then c<=a;
else c<=b;
end if;
end process;
```

# Señales y procesos: Ejemplo



Declaración de señales

```
library ieee;
use ieee.std_logic_1164.all;
entity simple is
port (a,b,c,d : in std_logic;
z: out std_logic);
end simple;

architecture logica of simple is
signal x, y : in std_logic;

begin
z <= x or y;
x <= a and b;
y <= c xor d;
end logica;
```

Procesos implícitos

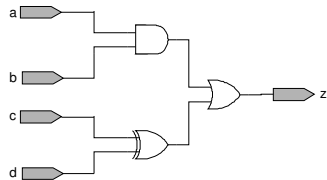
a, b, c, d y z son señales por defecto.

x e y son señales que necesitan ser declaradas.

El orden en la declaración de los procesos no influye en su ejecución.



## Instrucciones concurrentes



```
x <= a and b;  
y <= c xor d after 10 ns;  
z <= x or y after 5 ns;
```

### Asignaciones concurrentes

- Una instrucción se ejecuta cuando cambia una de sus entradas
- La asignación de los nuevos valores se realiza una vez ejecutadas todas las instrucciones cuyas entradas han cambiado.
- Si como consecuencia del nuevo valor cambia alguna de las entradas de otra instrucción, se ejecuta dicha instrucción



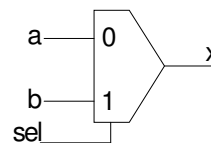
## Asignaciones Concurrentes

Asignación por ecuaciones booleanas:

```
x <= (a and not sel) or (b and sel);
```

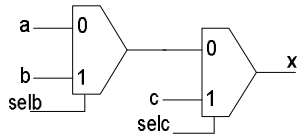
Asignación condicionales de las señales:

```
x <= a when sel='0' else  
b;
```

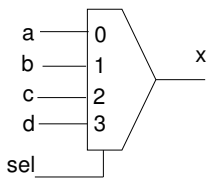


procesos  
implícitos

## Asignaciones condicionales



```
x <= c when selc='1' else
  b when selb='1' else
  a;
```



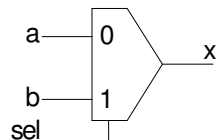
```
with sel select
  x <= a when sel="00",
  b when sel="01",
  c when sel="10",
  d when others;
```

procesos implícitos

La cláusula others cubre todas las posibilidades que no son descritas.

## Instrucciones secuenciales

- Las sentencias dentro de un proceso son evaluadas secuencialmente
- Un proceso puede estar *activo* o *inactivo*
- Típicamente un proceso posee una lista de sensibilidad: El proceso pasa a estar *activo* cuando cambia un valor de esta lista
- También puede tener una o varias sentencias wait, que hacen que el proceso pase a *inactivo*
- El tiempo avanza cuando el proceso pasa a inactivo

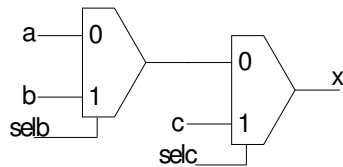


```
process (a, b, sel)
begin
  if sel='0' then
    x <= a;
  else
    x <= b;
  end if;
end process;
```





## Sentencias secuenciales IF-THEN-ELSE



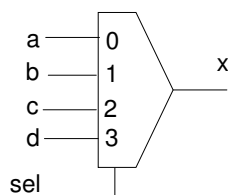
proceso  
explícito

```
process (selc, selb, a, b, c)
begin
    if (selc = '1') then
        x <= c;
    elsif (selb = '1') then
        x <= b;
    else
        x <= a;
    end if;
end process;
```

Con prioridad: las condiciones son evaluadas en orden de arriba hacia abajo. La primera condición *TRUE* hace que se evalúe la instrucción asociada. Si todas las condiciones son falsas se evalúa la instrucción asociada *ELSE*.



## Sentencias secuenciales CASE



proceso  
explícito

```
process (sel, a, b, c, d)
begin
    case sel is
        when "00" =>
            x <= a;
        when "01" =>
            x <= b;
        when "10" =>
            x <= c;
        when others =>
            x <= d;
    end case;
end process;
```

Las condiciones son evaluadas de una vez, sin prioridad. La cláusula *OTHERS* sirve para evaluar los casos no especificados.

## Bucles secuenciales: LOOP

Bucle infinito con instrucción de salida

```
[label_loop] loop
    sentencias secuenciales
end loop [label_loop];
```

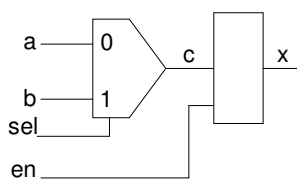
Bucle infinito con condición

```
while <condición> loop
    sentencias secuenciales
end loop;
```

Bucle **for** iterativo

```
for <identificador> in <rango> loop
    sentencias secuenciales
end loop;
```

## Asignación de señales: Ejemplo



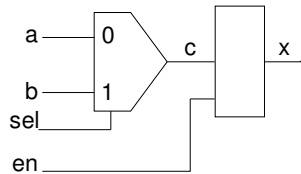
**INCORRECTO:**

Cuando  $en=1$ , a  $x$  se le asigna el valor previo de  $c$ .

```
library ieee;
use ieee.std_logic_1164.all;
entity mux2latch is
    port (a,b,sel, en : in std_logic;
          x: buffer std_logic);
end mux2latch;
architecture comp of mux2latch is
    signal c : std_logic;
begin
    mux: process (a, b, sel, en)
    begin
        if (sel='0') then c<=a;
        else c<=b;
        end if;
        x <=(x and (not en)) or (c and en);
    end process; -- c cambia aquí
end comp;
```



## Asignación de señales: Solución



**CORRECTO:** proceso con sentencias secuenciales y una asignación concurrente.

Cuando  $en='1'$ , a  $x$  se le asigna el valor actualizado de  $c$ .

```
library ieee;
use ieee.std_logic_1164.all;
entity mux2latch is
    port (a,b,sel, en : in std_logic;
          x: buffer std_logic);
end mux2latch;
architecture comp of mux2latch is
    signal c : std_logic;
begin
    mux: process (a, b, sel)
    begin
        if (sel='0') then c<=a;
        else c<=b;
        end if;
    end process; -- c cambia aquí
    x <=(x and (not en)) or (c and en);
end comp;
```



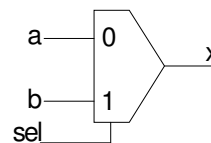
## Asignaciones Concurrentes

Asignación por ecuaciones booleanas:

```
x <= (a and not sel) or (b and sel);
```

Asignación condicionales de las señales:

```
x <= a when sel='0' else
    b;
```



procesos implícitos



## Resumen: Procesos y Señales

- Una arquitectura puede tener tantos procesos como queramos, y todos ejecutan en paralelo.
- **Los procesos se activan cuando alguna de las señales en su lista de sensibilidad cambia.**
- **El tiempo no avanza durante su ejecución.**
- **El tiempo sólo avanza cuando el proceso se suspende.**
- **Las señales modelan cables, y sólo pueden cambiar de valor si el tiempo avanza.**

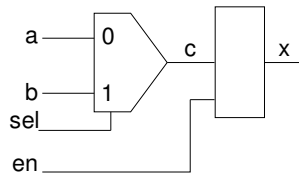


## VARIABLES

- Almacenamiento local de un dato temporal.
- Pueden ser de los mismos tipos de una señal (`signal`).
- **Visible solo en el interior de un proceso.**
- **Se declara en el proceso.**
- **Su asignación ( `:=` ) es inmediata.**

```
variable idle: std_logic := '0';
```

## Asignación de señales: Solución con variable



**CORRECTO:** la asignación a la variable es inmediata.

```
architecture comp of mux2latch is
begin
  mux: process (a, b, sel)
    variable c : std_logic;
  begin
    if (sel='0') then c:=a; -- inmediato
    else c:=b;
    end if;
    x <=(x and (not en)) or (c and en);
  end process;
end comp;
```

## Objetos disponibles en VHDL

**Constante** es una asociación de un nombre a un determinado valor

```
constant pi: real := 3.1416;
```

**Variable** es un almacenamiento local de un dato temporal visible solo en el interior de un proceso. Su valor es modificado por una asignación ( := )

```
variable parada: std_logic := `0`;
```

**Señal** es la modelización de un cable, el cual puede conectar puertos (E/S) de entidades y también conectar procesos dentro de la arquitectura. Es una forma de onda que cambia con el tiempo, por lo que su asignación ( <= ) no es inmediata, sólo tiene efecto cuando avanza el tiempo

```
signal parada: std_logic <= `0`;
```



## Señales Vs. Variables

	Señales	Variables
<b>Sintaxis</b>	destino <= fuente	destino := fuente
<b>Utilidad</b>	modelan nodos físicos del circuito	representan almacenamiento local
<b>Visibilidad</b>	global (comunicación entre procesos)	local (dentro de proceso)
<b>Comportamiento</b>	se actualizan cuando avanza el tiempo (se suspende el proceso)	se actualizan inmediatamente



## Asignación de señales en buses

```
signal tmp: std_logic_vector(7 downto 0);
```

Todos los bits

```
tmp <= "10100011";
```

```
tmp <= x"A3"; -- VHDL 93
```

Un solo bit

```
tmp(7) <= `1`;
```

Un rango de bits

```
tmp(3 downto 0) <= "0011";
```



## Uso correcto de las asignaciones a señales

- Agregados para compactar y hacer código más rápido en simulación

```
y <= others => '0';  
y <= ('0', '1', a xor b, a and b);  
y <= (3 => a xor b, 1 => '1', 2 => '0', a and b);  
y <= y(2 downto 0) & '0';
```

- No usar bucles para inicializar señales

```
for i in 0 to 4 loop  
  a(i) <= '0';  
end loop;
```



```
a(i) <= others => '0';
```

- No usar bucles para desplazar datos

```
for i in 3 downto 0 loop  
  if (i /= 0) then  
    d(i) <= d(i-1);  
  else  
    d(i) <= '0';  
  end if;  
end loop;
```

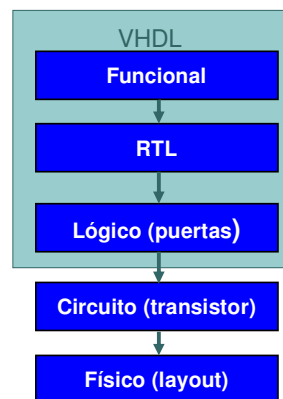


```
d <= d(2 downto 0) & '0';
```



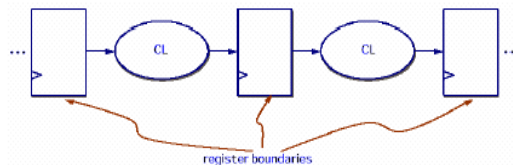
## Niveles de abstracción

- Funcional o Comportamental:
  - Conjunto de instrucciones secuenciales.
  - No se detallan relojes o retrasos.
  - No siempre es sintetizable.
- RTL (Register Transfer Level):
  - Es la entrada para la síntesis.
  - Las operaciones se realizan en un ciclo de reloj específico.
  - No se detallan retrasos.
- Lógico o de puertas:
  - Es la salida de la síntesis.
  - Expresado en término de ecuaciones lógicas o puertas y elementos de una biblioteca (genérica o específica).
  - Se incluye información de retraso para cada puerta.



## RTL: Register Transfer Level

- Flujo de datos entre registros y bloques funcionales



- Tienen en cuenta el ciclo de reloj
- Independiente de la tecnología
- Definición del sistema en términos de:
  - registros, lógica combinacional y operaciones

## Estilos de modelado: Algorítmico

- Descripciones similares a programas de software.
- Refleja funcionalidad del circuito mediante procesos concurrentes que poseen descripciones secuenciales del algoritmo

```
entity comparador is
  port (A, B :in bit;
        C:out bit);
end comparador;
```

```
architecture algoritmo of comparador is
begin
  process
  begin
    if (A=B) then C <= '1';
    else C <= '0';
    end if ;
    wait on A,B;
  end process ;
end algoritmo;
```





## Estilos de modelado: Flujo de Datos

- Descripciones basadas en expresiones y ecuaciones.
- Reflejan el flujo de información y las dependencias entre datos y operaciones

```
entity comparador is
    port (A, B :in bit;
          C:out bit);
end comparador;
```

```
architecture dataflow of comparador is
begin
    C <= (A and B) or (not (A) and not (B));
end algoritmo;
```



## Estilos de modelado: Estructural

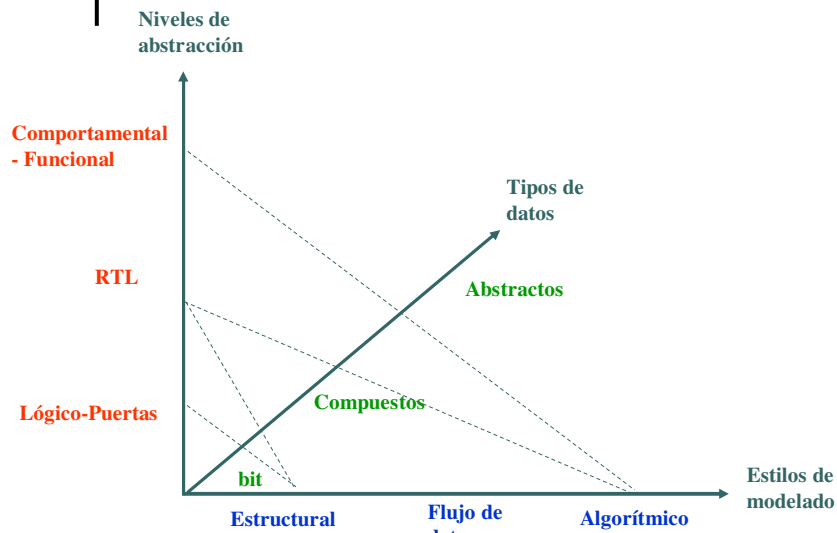
- Una unidad de alto nivel se divide en unidades nivel mas bajo
- Descripción que contiene los componentes y las conexiones entre los mismos

```
entity comparador is
    port (A, B :in bit;
          C:out bit);
end comparador;
```

```
architecture structural of comparador is
    component XOR2
        port (O: out bit; I1, I2: in bit);
    end component;
    component INV
        port (O: out bit; I: in bit);
    end component;
    signal S: bit;

begin
    C C1: XOR port map (O =>S, I1 =>A, I2 =>B);
    C2: INV port map (C, S);
end algoritmo;
```

## Nivel abstracción/Estilo de modelado



Introducción a VHDL - Parte I

51

## Paquetes

- Proveen una forma conveniente de almacenar y compartir información en un modelo.
- Se componen de:
  - Declaracion (Obligatorio).
    - Declaraciones de Tipos.
    - Declaraciones de Subprogramas.
  - Cuerpo (Opcional).
    - Definiciones de subprogramas.
- Ejemplos : Standard, Textio.

Introducción a VHDL - Parte I

52

## Paquete: Ejemplo

### Declaración

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
PACKAGE filt_cmp IS
  TYPE state_type IS (idle, tap1, tap2, tap3, tap4);
  COMPONENT acc
    port(xh : in std_logic_vector(10 downto 0);
         clk, first: in std_logic;
         yn : out std_logic_vector(11 downto 4));
  END COMPONENT;
  FUNCTION compare (SIGNAL a , b : integer) RETURN boolean;
END filt_cmp;
PACKAGE BODY filt_cmp IS
  FUNCTION compare (SIGNAL a , b : integer) RETURN boolean IS
    VARIABLE temp : boolean;
  Begin
    If a < b then
      temp := true;
    else
      temp := false;
    end if;
    RETURN temp;
  END compare;
END filt_cmp;
```

### Cuerpo

Introducción a VHDL - Parte I

53

## Librerías

- Contiene un paquete o una colección de paquetes.
- Librería Standar:
  - Package STD:
    - Bit, Boolean, Integer, Real, Time;
    - Operadores para los tipos soportados.
  - Package TEXTIO:
    - Funciones de archivos.
- La librería de trabajo (*work*) es en donde las módulos VHDL son compilados (o sintetizados).

Introducción a VHDL - Parte I

54