

# VHDL

Circuitos Lógicos Programables



Laboratorio de  
**Sistemas Embebidos**



- **Indice**

- Lenguajes descriptores de hardware (HDLs)
- VHDL
  - Introducción, Objetivos, Historia
  - Características
  - Entidad de diseño (Entidad y arquitectura)
  - Operadores
  - Tipos
  - Objetos
  - Expresiones concurrentes y secuenciales
  - Subprogramas
  - Funciones de resolución
  - Funciones de conversión

# Lenguajes descriptores de hardware

- **Introducción**

Un lenguaje descriptor de hardware es un lenguaje para la descripción formal y el diseño de circuitos electrónicos. Puede ser utilizado para describir el funcionamiento de un circuito, su diseño y organización, y para realizar las pruebas necesarias para verificar el correcto funcionamiento.

Ejemplos de este tipo de lenguaje son VHDL, Verilog, ABEL, AHDL, SystemC, SystemVerilog).

# Lenguajes descriptores de hardware

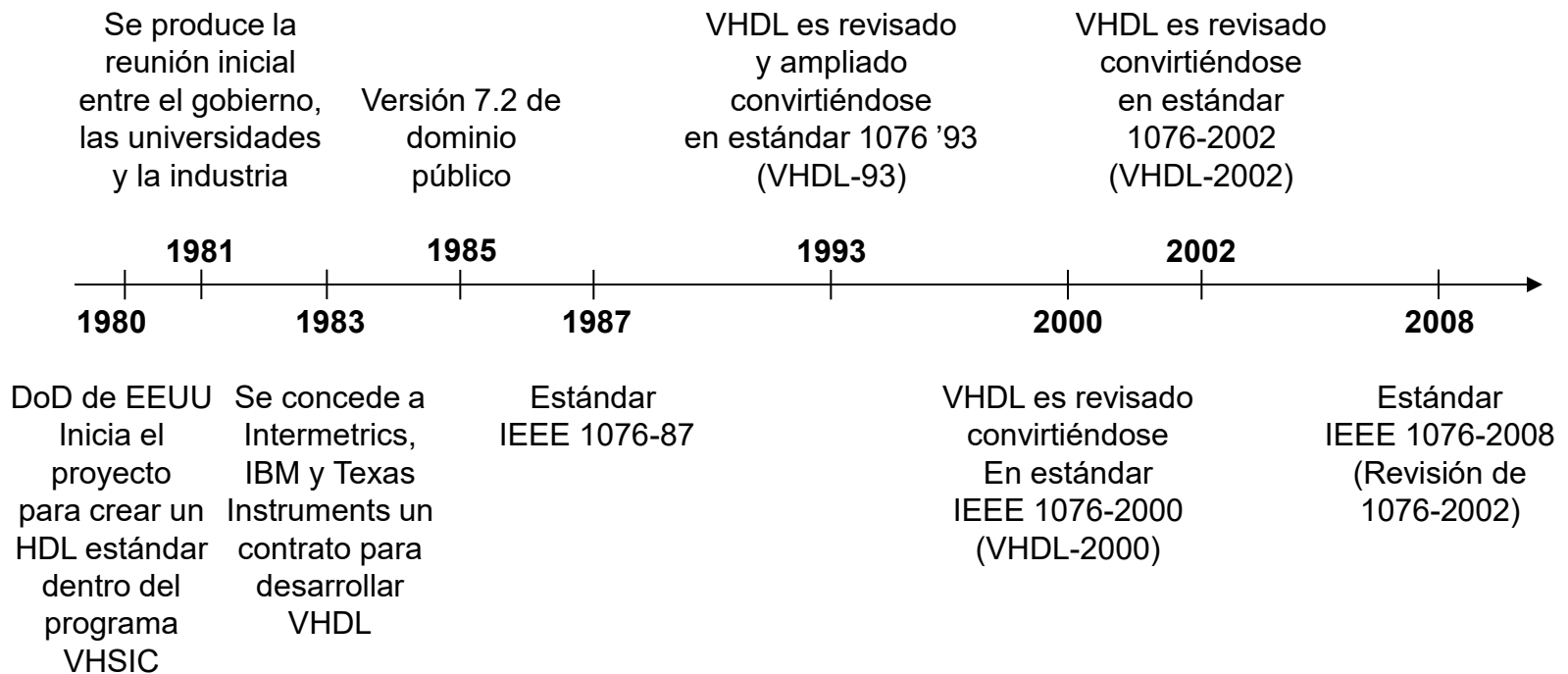
- **Objetivos**

- Especificación de circuitos electrónicos
- Documentación
- Simulación/Verificación de los circuitos antes de ser implementados

- **Introducción**

- VHDL significa **V**ery High Speed Integrated Circuit (VHSIC) **H**ardware **D**escription **L**anguage.
- Fue desarrollado originalmente por pedido del departamento de defensa de Estados Unidos.
- El lenguaje está especificado en la norma IEEE-1076
- Es utilizado para modelar circuitos digitales, especificando su comportamiento, y para simulación.
- La descripción de los circuitos puede hacerse en base a tres modelos: Comportamiento, Estructural y Flujo de datos

## • Historia



- **Características**

- Existen cuatro tipos de objetos en VHDL: constantes, variables, señales y archivos.
- VHDL es un lenguaje “*strong typing*”, lo que implica que no pueden asignarse valores a señales o variables que no sean del tipo declarado para esa señal o variable.
- Para realizar lo antes mencionado se debe llamar a una función de conversión de tipo.
- Permite dividir un diseño de hardware en módulos más chicos.

- **Características**

- Permite la definición de nuevos tipos
- El código suele más extenso que el desarrollado en Verilog (se busca que el mismo sirva como documentación)
- Posee librerías estándar
- No todo lo descrito en VHDL podrá ser utilizado para configurar una FPGA (**no todo VHDL es sintetizable**).
- No es “case sensitive”. Es lo mismo entity que ENTITY.
- Los comentarios se expresan con -- (doble guión).
- Las instrucciones finalizan con el carácter ; (punto y coma).



# VHDL: Entidad de diseño

La entidad de diseño es la principal abstracción de hardware en VHDL. Representa una porción del diseño de un hardware que tiene entradas y salidas definidas y realiza una determinada función. Puede representar un sistema completo, un subsistema, una plaqueta, un chip, una macrocelda, una compuerta lógica o cualquier nivel de abstracción que se encuentre entre los antes mencionados.<sup>(\*)</sup>

<sup>(\*)</sup> IEEE Std 1076™-2008 - IEEE Standard VHDL Language

# VHDL: Entidad de diseño

Una entidad de diseño puede ser descripta en términos de una jerarquía de bloques, cada uno de los cuales representa una porción del diseño completo. El bloque superior (*top-level*) en esta jerarquía es la propia entidad de diseño; tal bloque es un bloque externo y puede ser utilizado como componente de otros diseños. Los bloques anidados en la jerarquía son bloques internos. (\*)

(\*) IEEE Std 1076™-2008 - IEEE Standard VHDL Language

# VHDL: Entidad de diseño

## Entidad de diseño

Declaración de entidad

Declaración de la  
interfaz

Cuerpo de arquitectura

Descripción del  
funcionamiento

# VHDL: Declaración de entidad

- Una declaración de entidad define la interfaz entre una entidad de diseño dada y el entorno en el cual es usada. (\*)
- Define cuáles son los puertos y los modos de los mismos



(\*) IEEE Std 1076™-2008 - IEEE Standard VHDL Language

# VHDL: Declaración de entidad

- Sintaxis

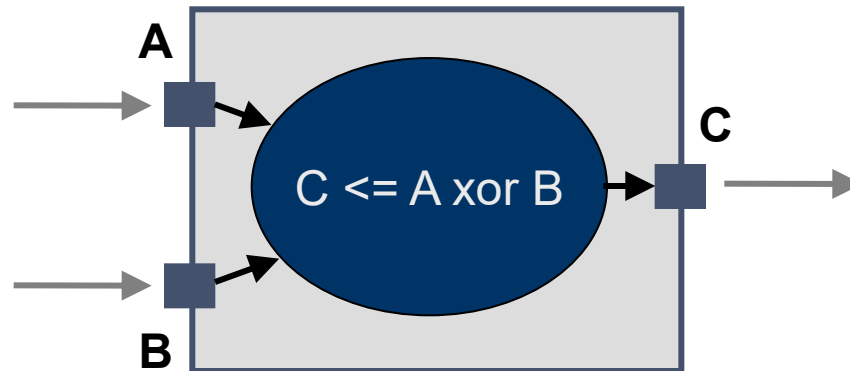
```
entity nombre_entidad is  
    [generic (generic_variable_declarations) ;]  
    [port (lista_de_interfaz) ;]  
    [declaraciones]  
[begin  
    [sentencias]]  
end [entity] [nombre_entidad]; (*)
```

(\*) IEEE Std 1076™-2008 - IEEE Standard VHDL Language

# VHDL: Cuerpo de arquitectura

- **Introducción**

- Un cuerpo de arquitectura define el cuerpo de una entidad de diseño. Especifica las relaciones entre las entradas y las salidas de esta, y puede estar expresada en términos de estructura, flujo de datos o comportamiento. (\*)



(\*) IEEE Std 1076™-2008 - IEEE Standard VHDL Language

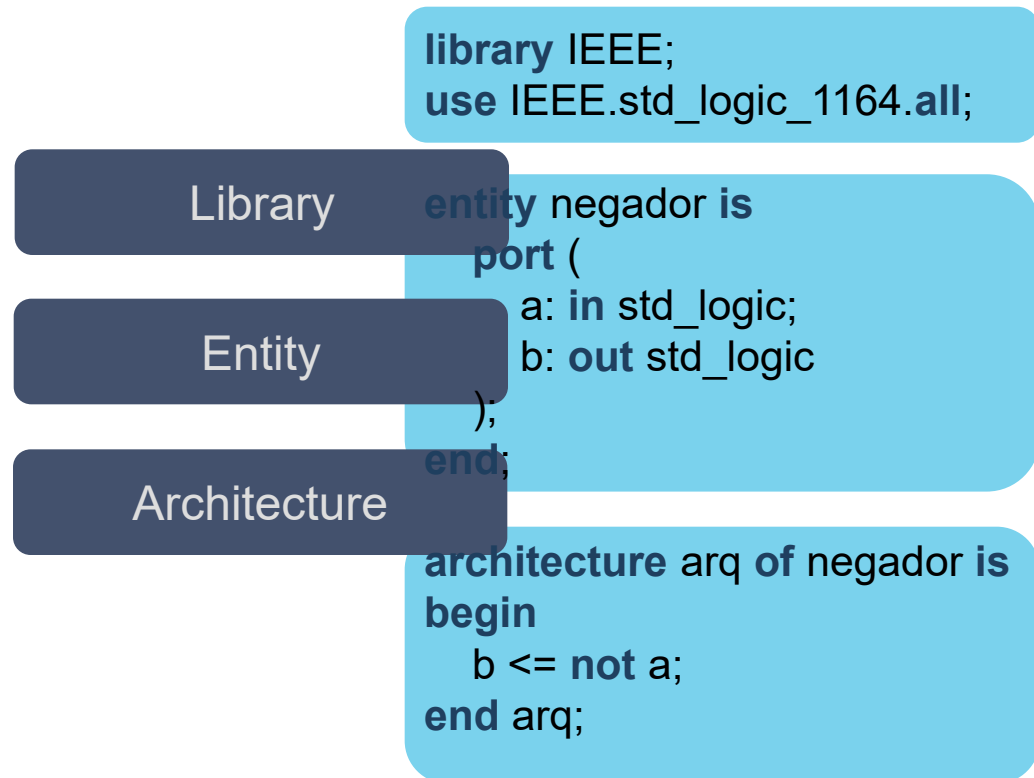
# VHDL: Cuerpo de arquitectura

- Sintaxis

```
architecture nombre_arquitectura of nombre_entidad is  
    [declaraciones]  
begin  
    [sentencias]  
end [architecture] [nombre_arquitectura]; (*)
```

(\*) IEEE Std 1076™-2008 - IEEE Standard VHDL Language

- Estructura de un código VHDL



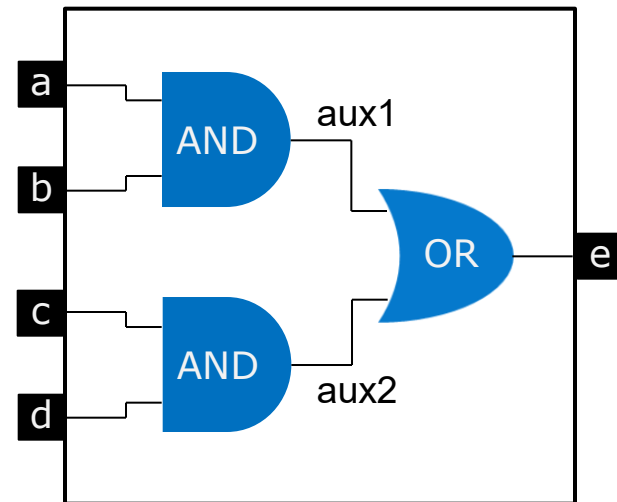


- Ejemplo en VHDL

```
entity circuito is
  port (
    a : in bit;
    b : in bit;
    c : in bit;
    d : in bit;
    e : out bit);
end circuito;
```

```
architecture arq of circuito is
  signal aux1: bit;
  signal aux2: bit;
begin
  aux1 <= a and b;
  aux2 <= c and d;
  e <= aux1 or aux2;
end arq;
```

$e \leq (a \text{ and } b) \text{ or } (c \text{ and } d);$



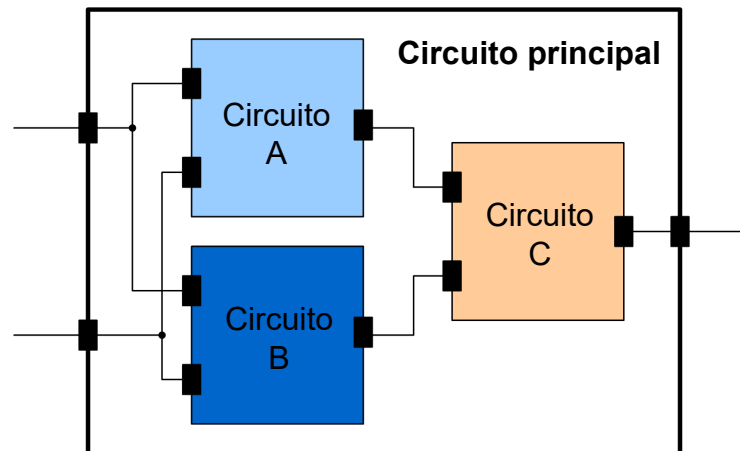
- **Instanciación de un componente**

La inclusión de un componente se realiza de la siguiente manera:

*nombre\_instancia: nombre\_componente*

***generic map**(lista\_genericos)*

***port map**(lista\_puertos);*



- **Instanciación de un componente**

Los módulos se pueden conectar especificando sus conexiones de dos modos:

- **Asociación posicional:** las conexiones a los puertos del módulo deben respetar el orden en que fueron definidos estos últimos.

```
sum1b_inst: sum1b port map(a, b, ci, s, co);
```

← No se indica el puerto

- **Asociación explícita:** las conexiones a los puertos del módulo pueden ser realizadas en cualquier orden.

```
sum1b_inst: sum1b  
port map(b_i => b, a_i => a, ci_i => ci, s_o => s, co_o => co);
```

Puerto      Señal conectada

- **Instanciación de un componente**

- **Asociación explícita:** las conexiones a los puertos del módulo pueden ser realizadas en cualquier orden.

```
sum1b_inst: sum1b
    generic(
        N => M
    )
    port map(
        b_i => b,
        a_i => a,
        ci_i => ci,
        s_o => s,
        co_o => co
    );
```

- Operadores en VHDL

## Operadores aritméticos

+	Suma
-	Resta
*	Multiplicación
/	División
**	Exponenciación

- Operadores en VHDL

## Operadores de relación

=	Igualdad
/=	Desigualdad
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

- **Operadores en VHDL**

## **Operadores lógicos**

not	Negación lógica
and	And lógica
or	Or Lógico
xor	Or exclusivo lógico

- Operadores en VHDL

## Operador de concatenación

&	Concatenación
---	---------------

Ej: Sean

```
signal a: std_logic_vector(3 downto 0) := "0010";
```

```
signal b: std_logic_vector(3 downto 0);
```

```
b <= a(3) & a(3 downto 1);
```

```
b <= '0' & a(3 downto 1);
```



# VHDL: Tipos

- La declaración de los tipos predefinidos se encuentra en el paquete STANDARD
- Existen 5 clases de tipos:
  - **Scalar** (entero, enumerado, físico, punto flotante)
  - **Composite** (array, record)
  - **File**
  - **Access**
  - **Protected**

# VHDL: Tipos escalares (ejemplos)

- Entero

```
type WORD_INDEX is range 31 downto 0;  
subtype HIGH_BIT_LOW is BYTE_LENGTH_INTEGER range 0 to 127;
```

\* El único tipo entero predefinido es **INTEGER**

- Enumerado

```
type BIT is ('0', '1');
```

\* Los tipos enumerados predefinidos son: CHARACTER, BIT, BOOLEAN, SEVERITY\_LEVEL, FILE\_OPEN\_KIND, Y FILE\_OPEN\_STATUS.

# VHDL: Tipos escalares (ejemplos)

- Físico

```
type DURATION is range -1E18 to 1E18
    units
```

```
    fs;                --femtosecond
    ps = 1000 fs;       --picosecond
    ns = 1000 ps;       --nanosecond
    us = 1000 ns;       --microsecond
    ms = 1000 us;       --millisecond
    sec = 1000 ms;      --second
    min = 60 sec;       --minute
```

```
end units;
```

\* El único tipo físico predefinido es **TIME**

# VHDL: Tipos escalares (ejemplos)

- Punto flotante

```
constant PI: REAL := 3.141592;
```

\* El único tipo punto flotante predefinido es **REAL**

# VHDL: Tipos compuesto (ejemplos)

- Array

```
type MY_WORD is array(0 to 31) of std_logic;  
type MEMORY is array(INTEGER range <>) of MY_WORD;
```

Estos tipos así declarados se pueden usar para describir una memoria de  $2^n$  palabras de 32 bits:

```
signal MY_MEMORY: MEMORY (0 to 2**N-1);
```

\* Los tipos array predefinidos son: STRING, BOOLEAN\_VECTOR, BIT\_VECTOR, INTEGER\_VECTOR, REAL\_VECTOR, and TIME\_VECTOR (definidos en el paquete **STANDARD**)

# VHDL: Tipos compuesto (ejemplos)

- Record

```
type DATE is
    record
        DAY : INTEGER range 1 to 31;
        MONTH : MONTH_NAME;
        YEAR : INTEGER range 0 to 5000;
    end record;
```

\* No existe ningún tipo record predefinido.

# VHDL: Tipo file (ejemplos)

- **File**

**file of** STRING

-- Define un tipo de archivo que puede  
-- contener un número indefinido de  
-- strings de largo arbitrario.

**file of** NATURAL

-- Define un tipo de archivo que puede  
-- contener sólo valores enteros positivos

# VHDL: Objetos

- **Objetos**

- En VHDL existen 4 clases de objetos:
  - **Constantes**
  - **Señales**
  - **Variables**
  - **Archivos**



# VHDL: Objetos

- **Constantes**

- Son utilizadas para el manejo de valores que se mantienen constantes durante el diseño. De esta manera se evita el uso de números que a simple vista parecen mágicos.
- Es una práctica muy recomendable el uso de nombres claros y que representen algo dentro del diseño.
- Para la asignación de un valor a una variable se utiliza el símbolo “:=”
- Se declaran del siguiente modo:

**constant** identificador: tipo [ := expresión];

# VHDL: Objetos

- **Generics**

- VHDL provee una estructura llamada generic, para pasar información dentro de una entidad y componente
- Ya que el generic no puede ser modificado dentro de la arquitectura funciona, de alguna manera, como una constante.
- Se declaran en la declaración de la entidad, antes de la declaración de puertos

# VHDL: Objetos

- Generics

```
entity nombre_entidad is
  generic(
    nombre_generico: tipo_dato := valor_default;
    ...
    nombre_generico: tipo_dato := valor_default
  );
  port (
    nombre_puerto: modo tipo_dato;
    ...
    nombre_puerto: modo tipo_dato
  );
end nombre_entidad;
```

# VHDL: Objetos

- Generics

## Ejemplo

```
entity sumador is
  generic(
    N: natural := 4
  );
  port (
    a: in std_logic_vector(N-1 downto 0);
    b: in std_logic_vector(N-1 downto 0);
    ci: in std_logic;
    sum: out std_logic_vector(N-1 downto 0);
    co: out std_logic
  );
end sumador;
```

# VHDL: Objetos

- Generics

## Ejemplo

```
architecture sumador_arq of sumador
    sum_aux: unsigned(N+1 downto 0);
begin
    sum_aux <= unsigned ('0' & a & ci) + unsigned ('0' & b & '1');
    sum <= std_logic_vector(suma_aux(N downto 1));
    co <= sum_aux(N+1);
end sumador_arq;
```

# VHDL: Objetos

- **Señales**

- Representan las conexiones del hardware. En otras palabras, representan los cables.
- En las simulaciones toman el valor asignado un delta de tiempo después (salvo que se indique un retardo por medio de la sentencia **after**).
- Para la asignación de un valor a una señal se utiliza el símbolo “<=”
- Se declaran del siguiente modo:  
**signal** identificador: tipo [ := expresión];

# VHDL: Objetos

- Señales: Ejemplos

Se supone definidas las siguientes variables

```
signal word: std_logic_vector(3 downto 0);
```

```
signal single_bit: std_logic;
```

Se podría tener las siguientes asignaciones:

- word <= "1110";

- single\_bit <= '1';

- word <= (**others** => '0');

- word <= single\_bit & word(3 **downto** 1);

# VHDL: Objetos

- **Variables**

- Son utilizadas para almacenar valores que pueden sufrir cambios.
- Cuando un valor es asignado a una variable esta lo toma inmediatamente.
- Para la asignación de un valor a una variable se utiliza el símbolo “:=”
- Se declaran del siguiente modo:

**variable** identificador: tipo [ := expresión];



# VHDL: Objetos

- Variables: Ejemplos

**variable** aux: integer := 3; -- declaración de variable con  
-- asignación inicial de un valor

aux := aux + 1; -- asignación de un valor a una variable

# VHDL: Objetos

- **Archivos**

- Estos objetos no son sintetizables.
- Se utilizan en simulaciones.
- Se puede acceder a ellos tanto para lectura como para escritura
- Se declaran del siguiente modo:

**file** identificador : subtipo [info\_apertura];

info\_apertura = [**open** modo\_de\_apertura] **is** nombre\_archivo

nombre\_archivo = string

# VHDL: Objetos

- Archivos

- Ejemplos (\*):

```
type IntegerFile is file of INTEGER;  
file F1: IntegerFile;  
-- No existe llamado implícito a FILE_OPEN.
```

```
file F2: IntegerFile is "test.dat";  
-- Existe un llamado implícito a:  
-- FILE_OPEN (F2, "test.dat"). El modo de apertura por defecto  
-- es READ_MODE.
```

```
file F3: IntegerFile open WRITE_MODE is "test.dat";  
-- Existe un llamado implícito a:  
-- FILE_OPEN (F3, "test.dat", WRITE_MODE);
```

(\*) IEEE Std 1076™-2008 - IEEE Standard VHDL Language

# VHDL: Ejemplos

- **Implementemos nuestros primeros circuitos!!!**
  - Circuito negador
  - Circuito sumador de 1 bit
  - Circuito multiplexor simple

# VHDL: Expresiones concurrentes y secuenciales

- **Expresiones concurrentes**

Su evaluación se efectúa al mismo tiempo.

- Instrucciones concurrentes.
- Procesos. Se utilizan para implementar una expresión concurrente utilizando una serie de expresiones secuenciales

- **Expresiones secuenciales**

Se evalúan en orden, una a continuación de la otra.

- Se sitúan dentro de los procesos

# VHDL: Sentencias secuenciales

- **Sentencia IF**

La sentencia if selecciona una o ninguna secuencia de sentencias.

Sintaxis:

```
[ if_label : ]  
  if condition then  
    sequence_of_statements  
  { elsif condition then  
    sequence_of_statements }  
  [ else  
    sequence_of_statements ]  
end if [ if_label ] ;
```

# VHDL: Sentencias secuenciales

- Sentencia IF

Ejemplo:

```
if x1= '0' and X2 = '1' then  
    sal <= '1';  
elsif x3= '1' then  
    sal <= '1';  
else  
    sal <= '0';  
end if;
```

# VHDL: Sentencias secuenciales

- **Sentencia CASE**

La sentencia case selecciona una entre varias alternativas de secuencia de sentencias.

Sintaxis:

```
[ label: ] case expression is  
  when choices => sequential_statements  
    when choices => sequential_statements  
    ...  
end case [ label ];  
choices = choice | ...  
choice = constant_expression | range | others
```



# VHDL: Sentencias secuenciales

- Sentencia CASE

Ejemplo:

```
C1: case sel is  
    when 1 => A <= '0';  
           B <= '0';  
    when 2 => A <= '1';  
           B <= '1';  
    when 3 => A <= '0';  
           B <= '1';  
    when others => A <= '1';  
           B <= '0';  
end case C1;
```

# VHDL: Sentencias secuenciales

- **Sentencia LOOP**

La sentencia Loop incluye una secuencia de sentencias que se ejecutarán reiteradamente, ninguna o varias veces.

Sintaxis:

```
[ loop_label : ]  
  [ iteration_scheme ] loop  
    sequence_of_statements  
  end loop [ loop_label ] ;
```

```
iteration_scheme ::=  
  while condition  
  | for loop_parameter_specification
```

```
parameter_specification ::=  
  identifier in discrete_range
```

# VHDL: Sentencias secuenciales

- **Sentencia FOR LOOP**

La sentencia For Loop contiene una secuencia de sentencias que se repetirán mientras el parametro\_loop se mantenga dentro del rango mencionado (*range*) en el encabezado de la sentencia .

Sintaxis:

```
[ etiqueta: ] for parametro_loop in range loop  
    sequential_statements  
end loop [ etiqueta ];
```

- El parámetro parametro\_loop no necesita ser declarado. La declaración del loop lo hace implícitamente.
- El parámetro parametro\_loop es una constante dentro de un loop lo que implica que no puede asignársele ningún valor dentro del mismo.

# VHDL: Sentencias secuenciales

- Sentencia FOR LOOP

Ejemplo:

```
A1: for i in vect'range loop  
    valor := valor and vect(i);  
end loop;
```

# VHDL: Sentencias secuenciales

- Sentencia **WHILE LOOP**

La sentencia While Loop contiene una secuencia de sentencias que se repetirán mientras la condición se mantenga verdadera.

Sintaxis:

```
[ etiqueta: ] while condition loop  
    sentencias_secuenciales  
end loop [ etiqueta ];
```

Ejemplo:

```
A1: while i < 8 loop  
    valor := valor and vect(i);  
    i := i + 1;  
end loop;
```

# VHDL: Sentencias secuenciales

- **Sentencia NEXT**

La sentencia Next es utilizada para dar por completa la ejecución de una de las iteraciones de un loop, pasando a la siguiente.

Permite saltar parte de la iteración de un loop. Si la condición especificada después de la palabra **when** es verdadera o no existe condición alguna entonces la sentencia es ejecutada. Con esto se logra saltar todas las sentencias que se encuentran por debajo hasta el final del loop y pasar el control a la primera sentencia de la próxima iteración.

Sintaxis:

```
[ label: ] next [ loop_label ] [ when condition ];
```

# VHDL: Sentencias secuenciales

- Sentencia NEXT

Ejemplo:

```
L1: loop
  k:= 0;
  L2: for CountValue in 1 to 8 loop
    next when CountValue = N;
    if A(k) = 'U' then
      next L1;
    end if;
    k:= k + 1;
  end loop L2;
end loop L1;
```

Salta a la próxima  
iteración del loop L2

Salta al inicio del  
loop L1

# VHDL: Sentencias secuenciales

- **Sentencia EXIT**

Sentencia secuencial que es usada para terminar la ejecución de una sentencia loop. Si existe una condición la sentencia exit será ejecutada cuando esta sea verdadera y el control pasará a la primera sentencia luego del final del loop.

Sintaxis:

```
[ label: ] exit [ loop_label ] [ when condition ];
```




# VHDL: Sentencias secuenciales

- Sentencia EXIT

Ejemplo:

```
L1: loop
  L2: for i in aux'range loop
    if aux(i) = '0' then
      exit L1;
    end if;
    exit when i = N;
  end loop L2;
  ...
end loop L1;
```



The diagram illustrates the execution flow of the VHDL code. It features two dark blue rectangular callouts with white text. The first callout, labeled 'Sale del loop L1', has a white arrow pointing to the 'exit L1;' statement in the code. The second callout, labeled 'Sale del loop L2', has a white arrow pointing to the 'exit when i = N;' statement in the code.

# VHDL: Sentencias secuenciales

- Sentencia **WAIT**

La sentencia wait provoca la suspensión de un process o un procedimiento.

Sintaxis:

```
[ label : ] wait [ sensitivity_clause ] [ condition_clause ] [ timeout_clause ] ;
```

```
sensitivity_clause ::= on sensitivity_list
```

```
sensitivity_list ::= signal_name { , signal_name }
```

```
condition_clause ::= until condition
```

```
condition ::= boolean_expression
```

```
timeout_clause ::= for time_expression
```

# VHDL: Sentencias secuenciales

- Sentencia WAIT

Ejemplos:

*wait for 25 ns;*

*wait on signal\_A;*

*wait on signal\_A for 100 ns;*

*wait until ena = '1';*

# VHDL: Sentencias concurrentes

- Sentencia **WHEN-ELSE**

Es una asignación condicional de señal.

Sintaxis:

```
target <= options conditional_waveforms ;
```

```
conditional_waveforms ::=
```

```
{ waveform when condition else }
```

```
waveform [ when condition ]
```

# VHDL: Sentencias concurrentes

- Sentencia WHEN-ELSE

Ejemplo:

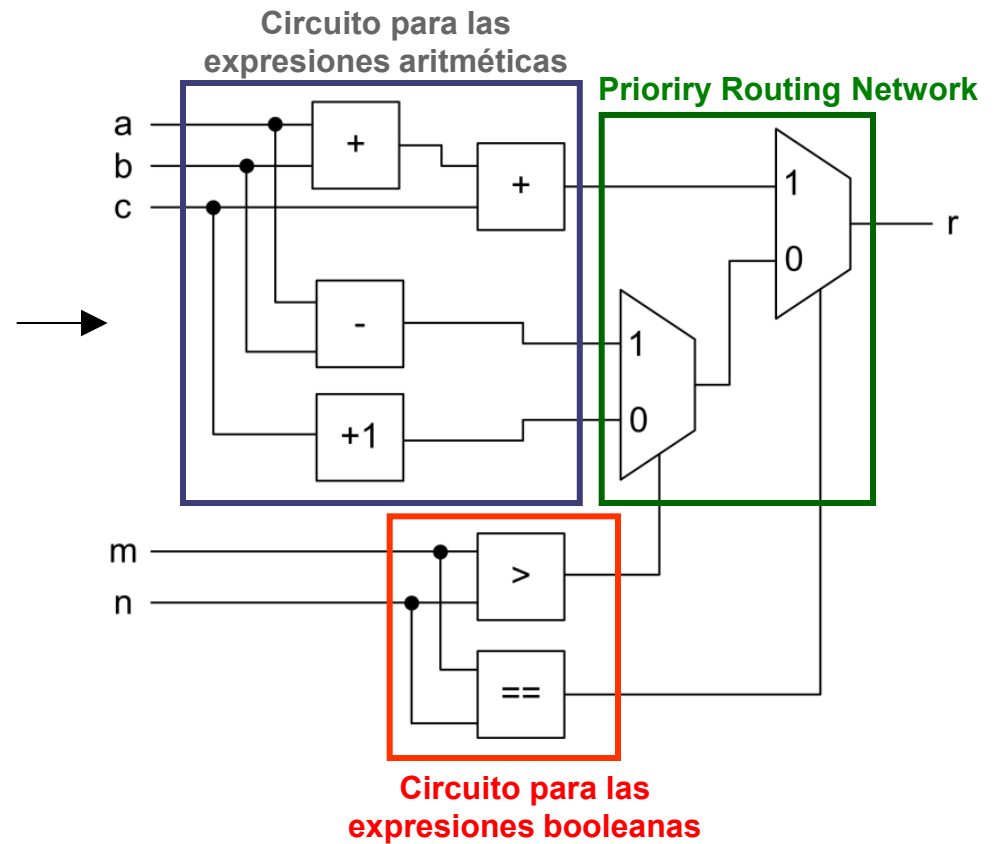
```
entity mux is
  port (
    ent: in std_logic_vector(0 to 3);
    sel: in std_logic_vector(0 to 1);
    o: out bit
  );
end;
architecture mux_arq of mux is
begin
  o <= ent(0) when sel = "00" else
    ent(1) when sel = "01" else
    ent(2) when sel = "10" else
    ent(3) when sel = "11" else
    '0';
end;
```

# VHDL: Sentencias concurrentes

- Sentencia WHEN-ELSE

Síntesis:

```
r <= a + b + c when m = n else  
      a - b when m > n else  
      c + 1;
```



# VHDL: Sentencias concurrentes

- Sentencia **WITH-SELECT**

Es una asignación de señal seleccionada.

Sintaxis:

```
with expression select  
    target <= options selected_waveforms ;
```

```
selected_waveforms ::=  
    { waveform when choices , }  
    waveform when choices
```

# VHDL: Sentencias concurrentes

- Sentencia WITH-SELECT

Ejemplo:

```
entity mux is
  port (
    ent: in std_logic_vector(0 to 3);
    sel: in std_logic_vector(0 to 1);
    o: out std_logic
  );
end;
architecture m of mux is
  begin
    with sel select
      o <= ent(0) when sel = "00",
        ent(1) when sel = "01",
        ent(2) when sel = "10",
        ent(3) when sel = "11";
  end;
```

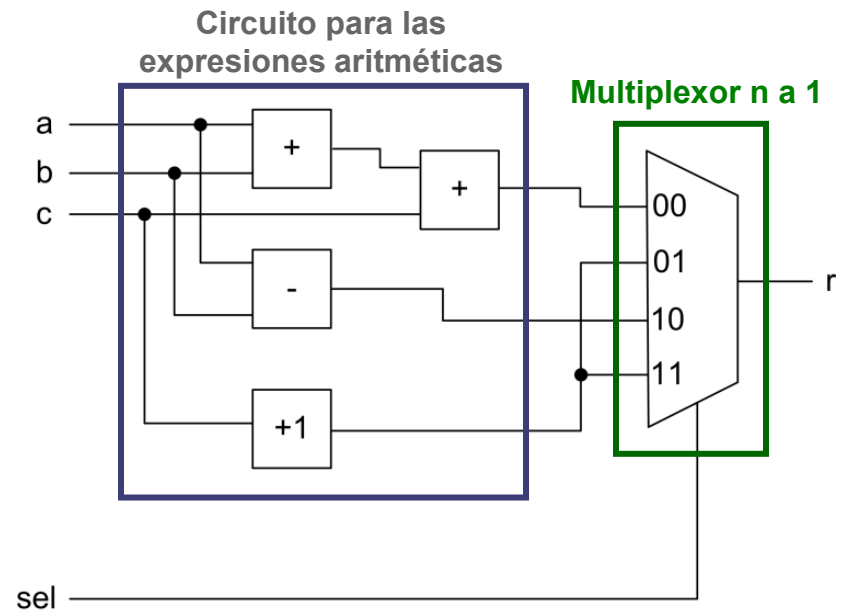
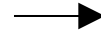


# VHDL: Sentencias concurrentes

- Sentencia WITH-SELECT

Síntesis:

```
with sel select  
  r <= a + b + c when "00",  
  a - b when "10",  
  c + 1 when others;
```



# VHDL: Sentencias concurrentes

- **Sentencia PROCESS**

Define un proceso secuencial independiente que representa el comportamiento de alguna parte del diseño.

Sintaxis:

```
[ process_label : ]  
    process [ ( process_sensitivity_list ) ] [ is ]  
        process_declarative_part  
    begin  
        process_statement_part  
    end [ postponed ] process [ process_label ] ;
```

# VHDL: Sentencias concurrentes

- Sentencia **PROCESS**: Lista de sensibilidad

Sintaxis:

```
[ process_label : ]  
  process [ ( process_sensitivity_list ) ] [ is ]  
    process_declarative_part  
  begin  
    process_statement_part  
  end [ postponed ] process [ process_label ] ;
```

*Nota: si existe una lista de sensibilidad se asume que el process contiene una sentencia **wait** implícita como última sentencia en su parte descriptiva.*

# VHDL: Sentencias concurrentes

- Sentencia **PROCESS**: Parte declarativa

Sintaxis:

```
[ process_label : ]  
  process [ ( process_sensitivity_list ) ] [ is ]  
    process_declarative_part  
  begin  
    process_statement_part  
  end [ postponed ] process [ process_label ] ;
```

```
process_declarative_part ::=  
  { process_declarative_item }
```

```
process_declarative_item ::=  
  subprogram_declaration  
  | subprogram_body  
  | type_declaration  
  | subtype_declaration ...
```

# VHDL: Sentencias concurrentes

- Sentencia **PROCESS**: Parte descriptiva

Sintaxis:

```
[ process_label : ]  
    process [ ( process_sensitivity_list ) ] [ is ]  
        process_declarative_part  
    begin  
        process_statement_part  
    end [ postponed ] process [ process_label ] ;
```

```
process_statement_part ::=  
    { sequential_statement }
```

# VHDL: Sentencias concurrentes

- Sentencia PROCESS

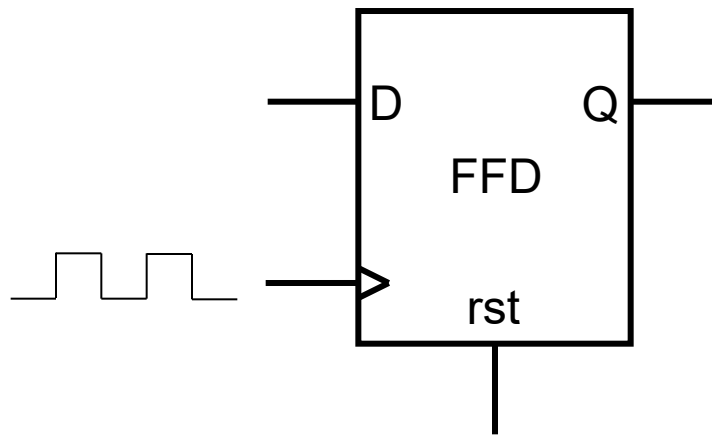
Ejemplo:

```
entity mux is  
  port (  
    a, b: in std_logic;  
    sel: in std_logic;  
    o: out std_logic  
  );  
end;  
  
architecture mux_arq of mux is  
begin  
  process(a, b, sel)  
  begin  
    if (sel = '0') then  
      o <= a;  
    else  
      o <= b;  
    end if;  
  end process;  
end mux_arq;
```

# VHDL: Sentencias concurrentes

- Sentencia **PROCESS**

Ejercicio: Implementar un flip-flop D utilizando un process activado por el flanco ascendente del reloj y por un reset asincrónico.



# VHDL: Sentencias concurrentes

- Sentencia **PROCESS**

Ejercicio: Implementar un flip-flop D utilizando un process activado por el flanco ascendente del reloj y por un reset asincrónico.

```
entity ffd is  
  port(  
    clk: in std_logic;  
    rst: in std_logic;  
    ena: in std_logic;  
    D: in std_logic;  
    Q: out std_logic  
  );  
end ffd;
```

```
architecture ffd_arq of ffd is  
  begin  
    process(clk, rst)  
      begin  
        if rst = '1' then  
          Q <= '0';  
        elsif rising_edge(clk) then  
          if ena = '1' then  
            Q <= D;  
          end if;  
        end if;  
      end process;  
  end ffd_arq;
```



# VHDL: Sentencias concurrentes

- Sentencia **PROCESS**

Ejercicio: Implementar un flip-flop D con reset sincrónico utilizando un process activado por el flanco ascendente del reloj.

```
entity ffd is  
  port(  
    clk: in std_logic;  
    rst: in std_logic;  
    ena: in std_logic;  
    D: in std_logic;  
    Q: out std_logic  
  );  
end ffd;
```

```
architecture ffd_arq of ffd is  
  begin  
    process(clk)  
      begin  
        if rising_edge(clk) then  
          if rst = '1' then  
            Q <= '0';  
          elsif ena = '1' then  
            Q <= D;  
          end if;  
        end if;  
      end process;  
  end ffd_arq;
```

# VHDL: Sentencias concurrentes

- **Sentencia GENERATE**

A generate statement provides a mechanism for iterative or conditional elaboration of a portion of a description.

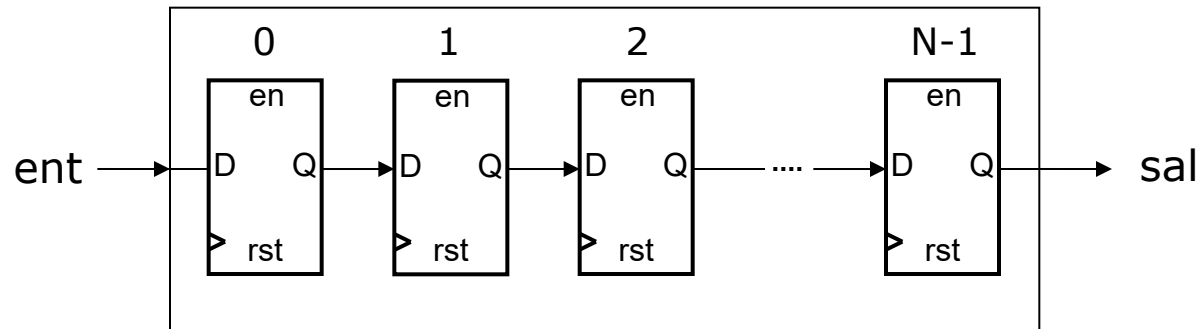
Sintaxis:

```
generate_label :  
    generation_scheme generate  
        [ { block_declarative_item }  
    begin ]  
        { concurrent_statement }  
    end generate [ generate_label ] ;  
  
generation_scheme ::=  
    for generate_parameter_specification  
    | if condition
```

# VHDL: Sentencias concurrentes

- Sentencia **GENERATE**

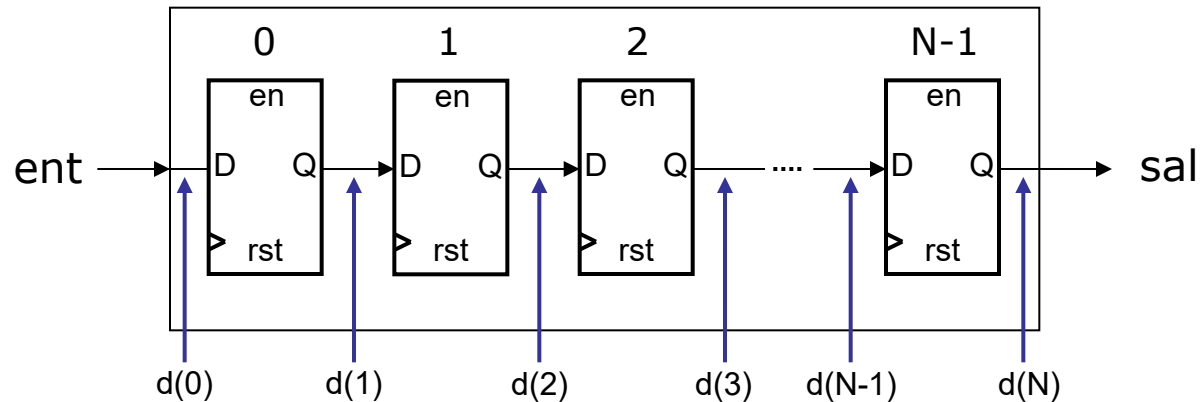
Ejemplo: Implementación de un registro de desplazamiento de N bits



# VHDL: Sentencias concurrentes

- Sentencia **GENERATE**

Ejemplo: Implementación de un registro de desplazamiento de N bits



# VHDL: Sentencias concurrentes

- Sentencia **GENERATE**

Ejemplo: Implementación de un registro de desplazamiento de N bits

```
entity shift_reg is
  generic (N: natural := 4);
  port ( clk: in std_logic; rst: in std_logic; ent: in std_logic;
        sal: out std_logic);
end;
architecture estruc of shift_reg is
  component ffd
    port ( rst: in std_logic := '0';
          ck: in std_logic;
          d: in std_logic;
          q: out std_logic);
  end component;
  signal d: std_logic_vector(0 to N);
begin
  shift_reg_i : for i in 0 to N-1 generate
    ff_inst : ffd port map(ck => clk , d => d(i), rst => rst; q => d (i + 1));
  end generate;
  d(0) <= ent;
  sal <= d(N) ;
end;
```

# VHDL: Subprogramas

- **Subprogramas**

- Los subprogramas pueden existir como sólo cuerpos de procedimiento o cuerpos de función.
- También puede existir una declaración de procedimiento y una declaración de función.
- Cuando un subprograma se incluye en un paquete (package) la declaración del mismo se sitúa en la parte declarativa de dicho paquete y el cuerpo del subprograma en el cuerpo del mismo.

# VHDL: Declaración de subprogramas

- **Subprogramas**

- Una declaración de subprograma declara un procedimiento o una función (indicado por la correspondiente palabra reservada, ***procedure*** o ***function***).
- Es opcional. En caso de no existir el propio cuerpo del procedimiento se utiliza como declaración.

# VHDL: Declaración de subprogramas

- **Declaración de procedure**

- Se utiliza para declarar la interfaz de llamada a un procedimiento

- **Declaración de función**

- Se utiliza para declarar la interfaz de llamada y retorno de una función



# VHDL: Declaración de subprogramas

- Sintaxis

***procedure*** nombre\_procedure [(lista\_parametros)]

***function*** nombre\_funcion [(lista\_parametros)] ***return*** tipo

# VHDL: Declaración de subprogramas

- **Procedure: Parámetros formales**

- Los parámetros formales se detallan en la *lista\_parametros* separados por “;”
- Cada parámetro formal es básicamente una declaración de un objeto que es local al procedure.
- Las definiciones de tipo usados en los parámetros formales deben ser visibles en el lugar donde el procedure es declarado.

# VHDL: Declaración de subprogramas

- **Procedure: Parámetros formales**

- Constantes
- Variables
- Señales
- Archivos

Si el modo es **IN** y no se especificó ninguna clase de objeto se asume **constante**.

Si el modo es **INOUT** o **OUT** se asume **variable**

- **Procedure: Modos de los parámetros formales**

- IN
- INOUT
- OUT

Por defecto el modo es **IN**.

El tipo **Archivo** no tiene modo.

# VHDL: Declaración de subprogramas

- **Función: Parámetros formales**

- Los parámetros formales se detallan en la *lista\_parametros* separados por “;”
- Cada parámetro formal es básicamente una declaración de un objeto que es local a la función.
- Las definiciones de tipo usados en los parámetros formales deben ser visibles en el lugar donde la función es declarada.

# VHDL: Declaración de subprogramas

- **Función: Parámetros formales**

- Constantes
- Señales
- Archivos

Si no se especificó ninguna clase de objeto se asume **constante**.

- **Función: Modos de los parámetros formales**

- IN

Por defecto el modo es **IN**.  
El tipo **Archivo** no tiene modo.

# VHDL: Cuerpo de subprograma

- **Subprogramas**

- Un cuerpo de subprograma especifica la ejecución de un subprograma
- Los items declarados luego de la especificación del subprograma declaran objetos que serán utilizados localmente dentro del cuerpo del subprograma.

Los nombres de estos objetos **no son visibles** fuera del subprograma.

# VHDL: Cuerpo de subprograma

- Sintaxis

```
procedure nombre_procedure [(lista_parametros)] is  
    parte_declarativa  
begin  
    parte_descriptiva  
end;
```

```
function nombre_funcion [(lista_parametros)] return tipo is  
    parte_declarativa  
begin  
    parte_descriptiva  
end;
```

# VHDL: Cuerpo de subprograma

- Ejemplo: Conversión a entero (1)

```
function word_to_int(word: std_logic_vector) return integer is  
    variable result : integer := 0;  
begin  
    for index in word'range loop  
        if (word(index) = '1') then  
            result := result + 2**index;  
        end if;  
    end loop;  
    return result;  
end word_to_int;
```



# VHDL: Cuerpo de subprograma

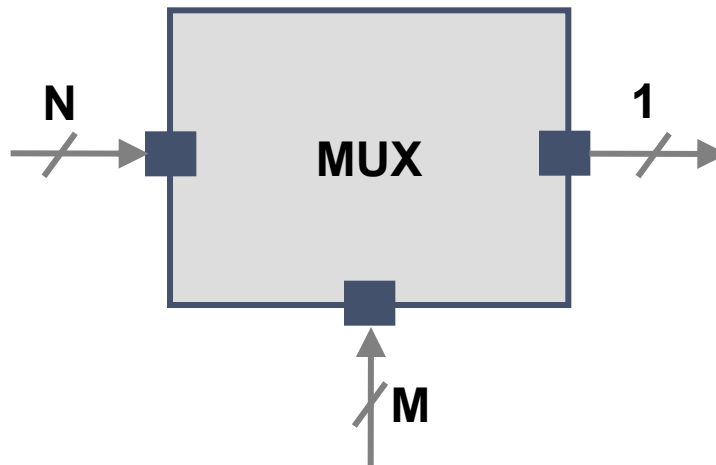
- Ejemplo: Conversión a entero (2)

```
function word_to_int(word: std_logic_vector) return integer is  
    variable result : integer := 0;  
begin  
    for index in word'range loop  
        result := result * 2;  
        if (word(index) = '1') then  
            result := result + 1;  
        end if;  
    end loop;  
    return result;  
end word_to_int;
```

# VHDL: Cuerpo de subprograma

- **Ejercicio:**

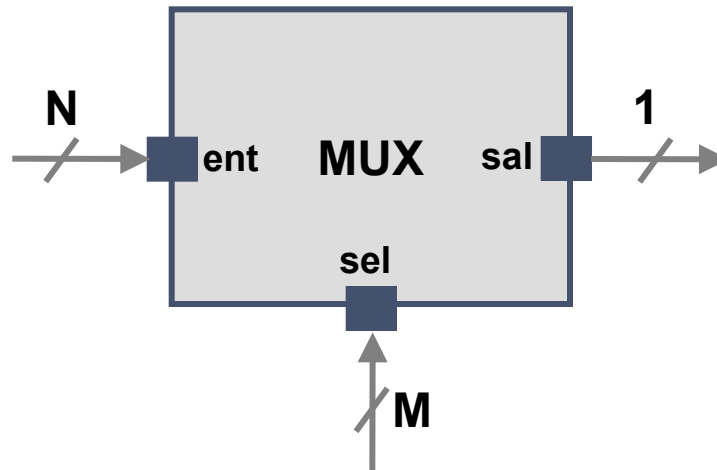
Utilizar la función de conversión a entero antes mostrada para describir el funcionamiento en VHDL de un multiplexor de N bits de entrada con selección de M bits ( $N = 2^M$ ) y salida de un bit



# VHDL: Cuerpo de subprograma

- Ejercicio:

```
entity mux is  
  generic(N, M: integer);  
  port(  
    ent: in std_logic_vector(N-1 downto 0);  
    sel: in std_logic_vector(M-1 downto 0);  
    sal: out std_logic  
  );  
end;
```



# VHDL: Cuerpo de subprograma

- Ejercicio:

```
architecture m of mux is  
  function word_to_int(word: std_logic_vector) return integer is  
    variable result : integer := 0;  
  begin  
    for index in word'RANGE loop  
      if (word(index) = '1') then  
        result := result + 2**index;  
      end if;  
    end loop;  
    return result;  
  end word_to_int;  
  
  begin  
    sal <= ent(word_to_int(sel));  
  end;
```

# VHDL: Cuerpo de subprograma

- **Sobrecarga**

- VHDL permite que dos subprogramas posean el mismo nombre, con la condición de que el número o el tipo de los parámetros sea distinto.
- Para determinar de qué procedimiento se trata en el caso de una llamada a un procedimiento sobrecargado es usada la cantidad de parámetros reales, su orden, y los tipos, y los nombres de los correspondientes parámetros formales
- Para el caso de funciones también se utiliza el tipo del resultado

# VHDL: Función de resolución

- **Definición**

- Es una función que define cómo los valores de múltiples fuentes de una señal dada se resuelven en un valor único para esa señal. (\*)
- Una señal con una función de resolución asociada se denomina señal resuelta.

(\*) IEEE Std 1076™-2008 - IEEE Standard VHDL Language

# VHDL: Función de resolución

- Ejemplo de un tipo no resuelto (BIT)

```
entity testbench is  
end testbench;
```

```
architecture behavior of testbench is  
    signal a: bit;  
begin  
    a <= '1';  
    a <= '0';  
end;
```

```
Running ISim simulation engine ...  
This is a Lite version of ISE Simulator(ISim).  
✗ ERROR: Resolution function required for the Net /testbench/a driven by:  
/testbench//pru_tipo_no_resuelto.vhd:7  
/testbench//pru_tipo_no_resuelto.vhd:8
```

# VHDL: Función de resolución

- Ejemplo de un tipo resuelto (STD\_LOGIC)

```
library IEEE;  
use IEEE.std_logic_1164.all;  
  
entity pru is  
end;  
  
architecture p of pru is  
    signal aux: std_logic_vector(15 downto 0);  
begin  
  
    aux <= (others => '0');  
    aux <= b"0111_0101_0001_0000";  
  
end;
```



# VHDL: Función de resolución

- Ejemplo de un tipo resuelto

