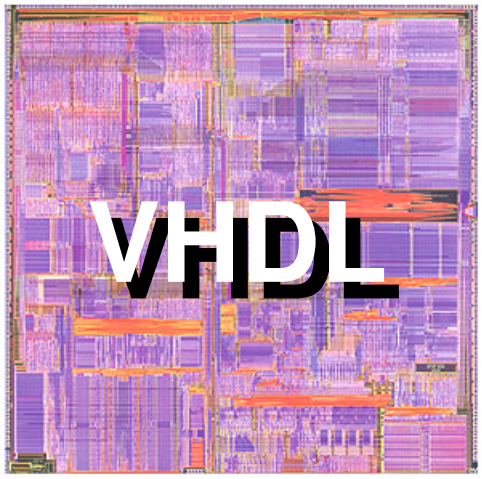
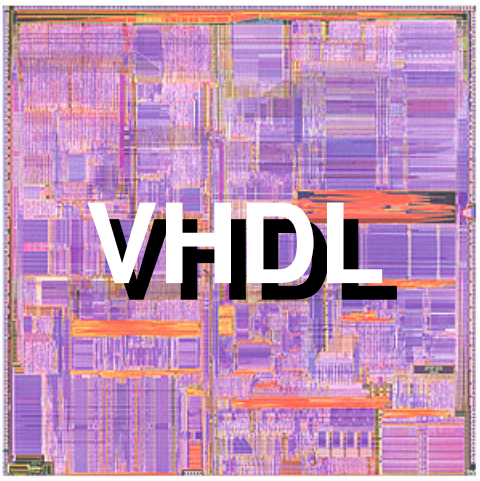
Sergio Lopez-Buedo, Elías Todorovich
Octubre 2012

VHDL Avanzado



- Entidades parametrizadas
- Instanciaciones Iterativas y Condicionales
  - Generate
  - Generic + Generate
- Atributos
- Alias
- Funciones y procedimientos
- Records
- Mixed Language Design

VHDL Avanzado



Entidades parametrizadas
Instanciaciones Iterativas y Condicionales
- Generate
- Generic + Generate

Atributos
Alias
Funciones y procedimientos
Records
Mixed Language Design

GENERIC: Módulos parametrizables

- Si hay que describir modelos de una misma entidad con diferentes anchos de operandos, seria una mala práctica hacerlo de forma fija para cada una. Es conveniente describir entidades más generales: modelos genéricos.

- En la declaración de una entidad no sólo se pueden declarar sus puertos, sino que también se pueden especificar una colección de parámetros genéricos.

- Se utiliza la palabra clave GENERIC

Valor por defecto opcional

```
entity CPU is
 generic (BusWd: integer := 16);
 port(DataBus: inout std_logic_vector(BusWd-1 downto 0));
end entity CPU;
```

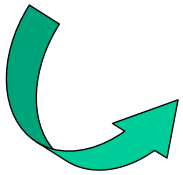
GENERIC: Módulos parametrizables

- Para especificar los valores de los parámetros, cuando se instancie el componente, se usa GENERIC MAP

```
CPU1 : CPU generic map (BusWd => 16)
           port map (DataBus => Bus);
```

- El GENERIC funciona igual que una constante
  - Normalmente son enteros para definir los anchos de las entradas, pero para simulación y modelos post PAR también se parametrizan tiempos

```
entity or2 is
 generic (Tpd: Time);
 port(a,b: in std_logic; y: out std_logic);
end entity CPU;

architecture simple of or2 is
begin
 y <= a and b after Tpd;
end architecture simple;
```



```
gate1: entity work.or2 (simple)
 generic map (Tpd => 1 ns);
 port map( … );

gate2: entity work.or2 (simple)
 generic map (Tpd => 2 ns);
 port map( … );
```


GENERIC: Ejemplo

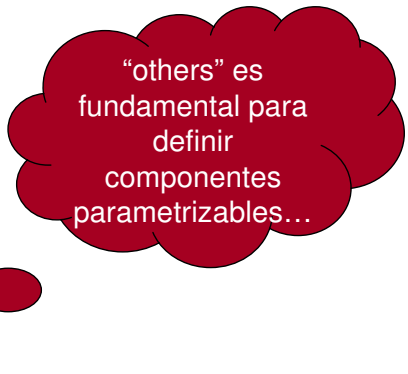
- Ejemplo: Sumador con ancho de operandos parametrizable

```
entity adder is
        generic ( L : integer );
        port (
                A,B : in std_logic_vector(L-1 downto 0);
                S : out std_logic_vector(L-1 downto 0);
        );
end adder;


architecture uno of adder is
begin

        S <= A + B;

end uno;
```

GENERIC: Otro Ejemplo

- Ejemplo: Conversor S/P con ancho de datos parametrizable

```
entity CONVSP is
    generic ( L : integer := 8 );
    port (
          S, CLK, RESET : in std_logic;
          Q : out std_logic_vector(L-1 downto 0);
    );
end CONVSP;

architecture PARAM of CONVSP is
begin
    process (CLK, RESET)
          variable AUX: std_logic_vector (L-1 downto 0);
    begin
          if RESET = '1' then
                AUX := (others => '0');
          elsif rising_edge (CLK) then
                AUX := AUX (L-2 downto 0) & S;  -- Carga Serie
          end if;
          Q <= AUX;
    end process;
end PARAM;
```



Generic: Valores por defecto

- Siempre que se pongan valores por defecto en los generics al declarar el componente, no será necesario hacer un **generic map** para cada parámetro
- Ejemplo:

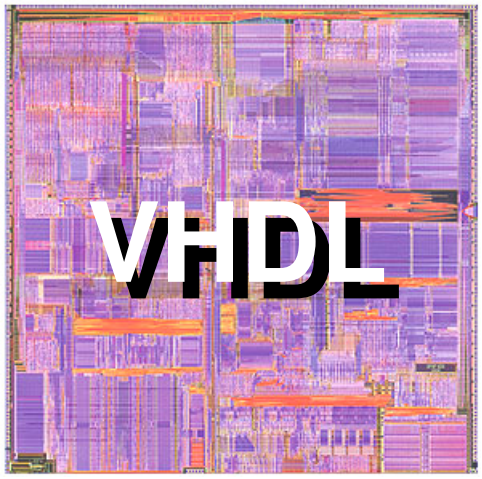
```
U1 : convsp
generic map ( L => 16 )
port map ( S => S, CLK => CLK, RESET => RESET, Q => Q);

U2 : convsp -- L = 8
port map ( S => S1, CLK => CLK, RESET => RESET, Q => Q2);
```

Generics en Xilinx

- Los parámetros genéricos son muy usados en los modelos de simulación de Xilinx
  - Ejemplo: dando valores iniciales a una memoria BlockRAM

```
ram_1024_x_18: RAMB16_S18
 --synthesis translate_off
 generic map ( INIT_00 => X"0F07…0000",
               INIT_01 => X"0F804…EF07",
               INIT_02 => X"0000…CF07",
               ...
               INITP_07 => X"C000…0000")
 --synthesis translate_on
 port map (   DI => "0000000000000000",
             DIP => "00",
              EN => '1',
              WE => '0',
             SSR => '0',
             CLK => clk,
            ADDR => address,
              DO => instruction(15 downto 0),
             DOP => instruction(17 downto 16));
```


VHDL Avanzado



Entidades parametrizadas

Instanciaciones Iterativas y Condicionales

- **Generate**
- **Generic + Generate**

Atributos

Alias

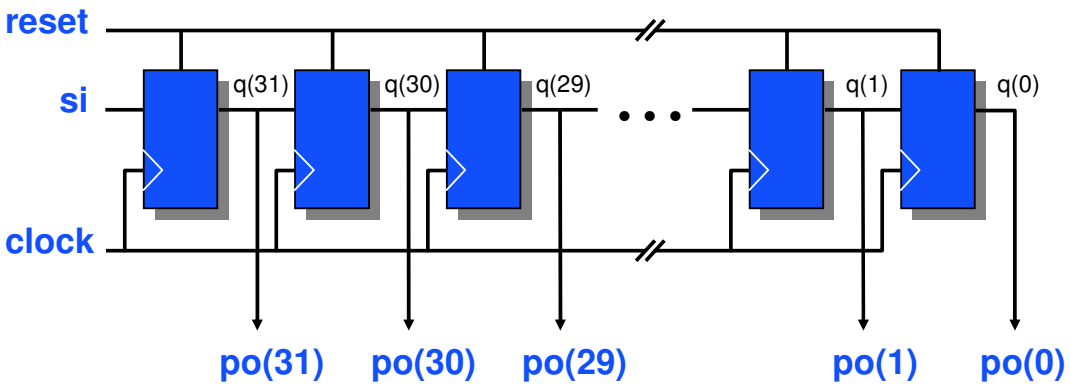
Funciones y procedimientos

Records

Mixed Language Design

Ejemplo de estructura repetitiva

- Ejemplo: Conversor serie-paralelo de 32 bits



GENERATE: Instanciación iterativa y condicional

- La instrucción GENERATE permite generar código iterativamente o condicionalmente
- Generación iterativa
  - Es ideal para circuitos repetitivos, como arrays de componentes

```
etiqueta: FOR parametro IN rango GENERATE
{ sentencias concurrentes }
END GENERATE;
```

- Generación condicional
  - Menos usada, por ejemplo para el primer elemento en un array

```
etiqueta: IF condicion GENERATE
{ sentencias concurrentes }
END GENERATE;
```

Ejemplo de estructura repetitiva: solución con GENERATE

- Se usa para crear estructuras iterativas, tales como *arrays* de componentes

```
ENTITY sipo IS PORT (
    clock, reset: IN std_logic;
    si: IN std_logic;
    po: OUT std_logic_vector(31 DOWNTO 0));
END sipo;

USE WORK.rtlpkg.ALL;

ARCHITECTURE archsipo OF sipo IS
    SIGNAL q: std_logic_vector(31 DOWNTO 0);
BEGIN
    beg: dsrff PORT MAP (si, zero, reset, clock, q(31));
    gen: FOR i IN 0 TO 30 GENERATE
        nxt: dsrff PORT MAP (q(i+1), zero, reset, clock, q(i));
    END GENERATE;
    po <= q;
END archsipo;
```


Instanciación Condicional

- Es útil para instanciar los componentes de los extremos:
  - Los componentes del medio se conectan con sus vecinos pero los de los extremos no tienen vecinos…
- Otro uso es para incluir o no componentes opcionales de un diseño:
  - Si se tienen recursos suficientes puede pensarse en operar en paralelo (multiplicación, división, etc.).
  - Instrumentación: componentes para facilitar el *debug* o *test* de un diseño.

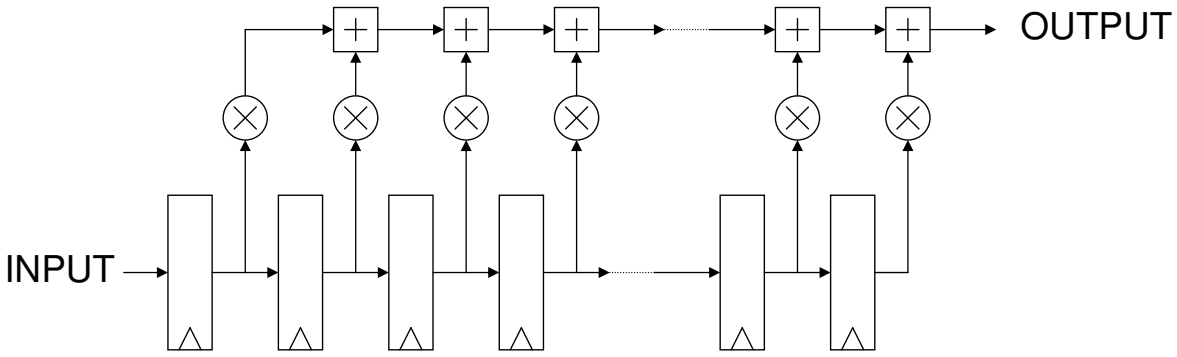
Ejemplo de Instanciación Condicional

- En un microprocesador se puede tener un multiplicador HW o no, según el valor de una constante:

```
gen_multiplier1 : if C_IMPL_MUL = 1 generate
   i_comb_mltplr : comb_mltplr
     generic map (
       DWIDTH     => DWIDTH)
     port map (
       mltplcnd_i => s_mltplcnd,
       mltplctr_i => s_mltplctr,
       product_o  => s_product);
 end generate gen_multiplier1;

 gen_multiplier0 : if C_IMPL_MUL /= 1 generate
   s_product <= (others => '0');
 end generate gen_multiplier0;
```


GENERATE + GENERIC: Módulos regulares y parametrizables

- GENERATE y GENERIC se pueden utilizar juntos para construir módulos parametrizables regulares
- Ejemplo: Filtro FIR



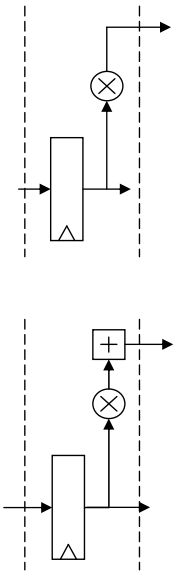
Ejemplo de filtro FIR: Entidad

```
entity fir_generic is
generic (
        taps : integer := 8;
        width : integer := 16
);
port (
        clk : in std_logic;
        input : in std_logic_vector(width-1 downto 0);
        output : out std_logic_vector(width-1 downto 0);
        all_coeffs : in std_logic_vector(taps*width-1 downto 0)
);
end fir_generic;
```


Ejemplo de filtro FIR: Señales

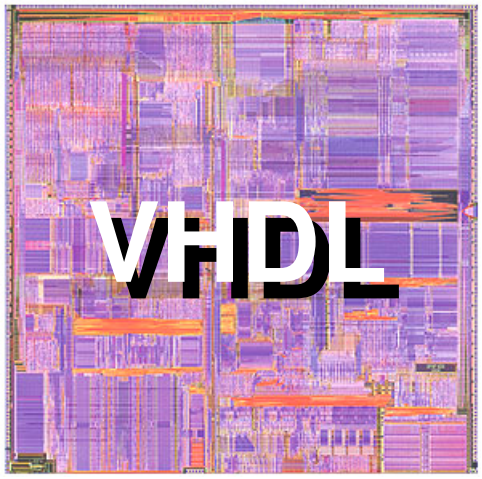
```
architecture Behavioral of fir_generic is

type fir_signals_t is array (taps-1 downto 0) of
        std_logic_vector(width-1 downto 0);
signal coeff : fir_signals_t;
signal delayed_input : fir_signals_t;
signal add : fir_signals_t;

type mult_signals_t is array (taps-1 downto 0) of
        std_logic_vector(2*width-1 downto 0);
signal mult : mult_signals_t;

begin

g1 : for i in 0 to taps-1 generate
        coeff(i) <= all_coeffs( ((i+1)*width)-1 downto (i*width) );
end generate;
```

Ejemplo de filtro FIR: Generates



```
g2 : for i in 0 to taps-1 generate
        g3 : if i = 0 generate
                process(clk)
                begin
                        if rising_edge(clk) then
                                delayed_input(i) <= input;
                        end if;
                end process;
                mult(i) <= delayed_input(i) * coeff(i);
                add(i) <= mult(i)(width-1 downto 0);
        end generate;
        g4 : if i > 0 generate
                process(clk)
                begin
                        if rising_edge(clk) then
                                delayed_input(i) <= delayed_input(i-1);
                        end if;
                end process;
                mult(i) <= delayed_input(i) * coeff(i);
                add(i) <= mult(i)(width*2-1 downto width) + add(i-1);
        end generate;
end generate;

output <= add(taps-1);
```


VHDL Avanzado



- Entidades parametrizadas
- Instanciaciones Iterativas y Condicionales
  - Generate
  - Generic + Generate
- **Atributos**
- Alias
- Funciones y procedimientos
- Records
- Mixed Language Design

Atributos predefinidos

- En VHDL, a todos los objetos (named entities) se le pueden asociar atributos.
- Sirven para anotar información adicional en el modelo.
- Existen atributos predefinidos, y también atributos definidos por el usuario
- La sintaxis para acceder a los atributos es emplear '

```
if clk'event and clk='1' then
```


Atributos predefinidos

- Atributos de tipos Escalares:
  - Con low y high se pueden obtener los números máximos y mínimos que se pueden representar con un tipo

```
integer'low vale -2147483648
integer'high vale 2147483647
```

  - Con image se obtiene una representacion imprimible (texto)

```
integer'image
```

  - Otros Atributos:

```
left    rigth    ascending        descending
value   pos      val      succ    pred
leftof  rightof  base
```

Atributos predefinidos

- Atributos de tipos Array y arrays
  - Mediante atributos se pueden obtener las dimensiones y características de un array

```
signal sig : std_logic_vector(7 downto 0) := "10010010";
...
sig'left          vale '1'
sig'right         vale '0'
sig'ascending     vale false
sig'range         vale 7 downto 0
```

  - Range se puede emplear en bucles for

```
for i in sig'range loop
```

  - Otros Atributos:

```
low      high     reverse_range     lenght
```


Atributos predefinidos

- Atributos de señales
  - Comprobar un tiempo de *setup*

```
assert sig'stable(tsetup) report "tiempo de setup falla";
```

  - Otros Atributos:

```
delayed          stable   quiet            transaction
event            active   last_event       last_active
last_value       driving  driving_value
```

Atributos predefinidos

- Atributos de items con nombre
  - Con path_name, simple_name e instance_name se pueden obtener los nombres de los objetos

```
sig'path_name vale :entidad::sig
sig'instance_name vale :entidad(arquitectura)::sig
```

  - Ejemplo: control de tiempo de set-up

```
Tsu_check: process (clk) is
begin
  if clk = '1' then
    assert d'last_event >= Tsu
         report "violación de tiempo de set-up en: " &
         Tsu_check'pah_name severity error;
  end if;
end process Tsu_check;
```


Atributos definidos por el usuario

- Aparte de los atributos predefinidos, el usuario puede definir atributos nuevos.
  - Solo tiene que especificarse su tipo:

```
attribute mi_atributo : string;
```

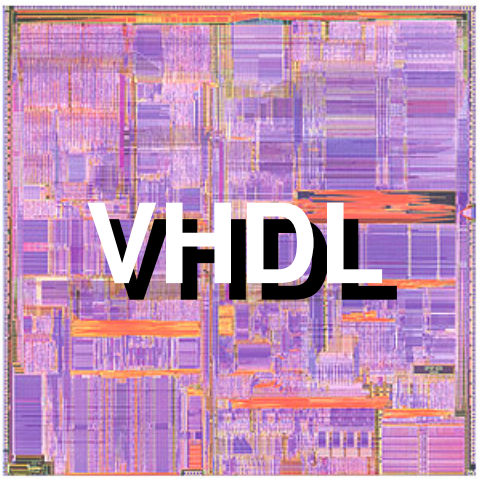
- Los atributos son el mecanismo que se utiliza para pasar información (opciones, valores iniciales, configuraciones...) a las herramientas de síntesis y/o implementación

```
attribute opt_mode: string;
attribute opt_mode of stopwatch : entity is "area";
```

- Los atributos resultan un mecanismo muy potente para especificar opciones en la síntesis:

```
attribute RLOC of gen_LUT2: label is "X3Y" & integer'image(i);
```

VHDL Avanzado



Entidades parametrizadas
Instanciaciones Iterativas y Condicionales
- Generate
- Generic + Generate

Atributos
Alias
Funciones y procedimientos
Records
Mixed Language Design

Alias

- Los Alias son simplemente nombres alternativos para otro nombre:

```
…D. Víctor José de la Sierra Tabares-Casas, de ahora
   en adelante "el comprador"…
```

- Los modelos en VHDL deben ser fáciles de leer y entender.
- Ejemplo: Unidad de Control de un microprocesador sencillo.

Ejemplo: Unidad de Control de un micro sencillo:

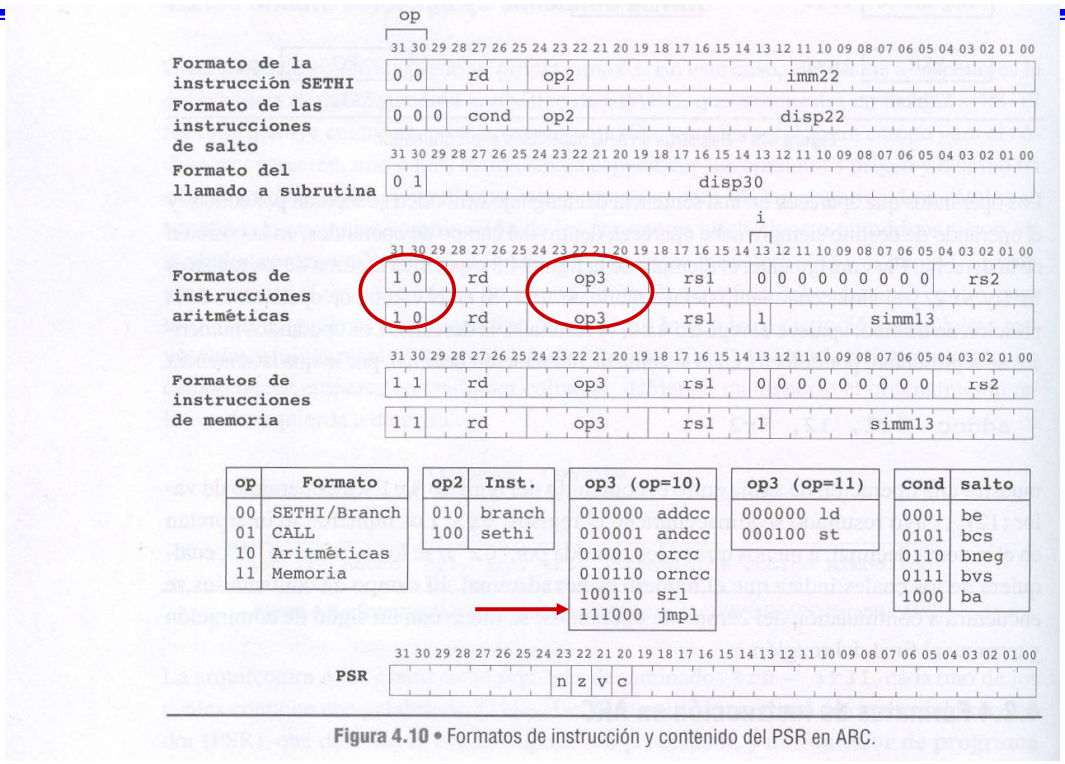


Figura 4.10 • Formatos de instrucción y contenido del PSR en ARC.

Alias

- PC_JUMP: se debe activar cuando haya una instrucción jmpl
  Versión 1:

```
entity CTRL is
   port( …
         IR : in STD_LOGIC_VECTOR(31 downto 0); -- Registro de
   Instrucción
 …);
end CTRL;
architecture PRACTICA of CTRL is
…
begin
   …
   PC_JUMP <= '1' when IR(31 downto 30)="10" and IR(24 downto
   19)="111000" else '0';
   …
end PRACTICA;
```

Alias

Versión 2: Uso de constantes

```
architecture PRACTICA of CTRL is
…
constant I_ARI : STD_LOGIC_VECTOR(1 downto 0) := "10"; -- instr.
   arit.
constant CO_JM : STD_LOGIC_VECTOR(5 downto 0) := "111000"; -- jumpl
…
begin
   …
   PC_JUMP <= '1' when IR(31 downto 30)=I_ARI and IR(24 downto
   19)=CO_JM else '0';
   …
end PRACTICA;
```


Alias

Versión 3: Si no existiesen los alias…

```
architecture PRACTICA of CTRL is
…
constant I_ARI : STD_LOGIC_VECTOR(1 downto 0) := "10"; -- instr. arit.
constant CO_JM : STD_LOGIC_VECTOR(5 downto 0) := "111000"; -- jumpl
signal OP :   STD_LOGIC_VECTOR(1 downto 0); -- Cod. Operación
signal OP3 :  STD_LOGIC_VECTOR(5 downto 0); -- Cod. Operación 3
…
begin
   OP  <= IR(31 downto 30);
   OP3 <= IR(24 downto 19);
   …
   PC_JUMP <= '1' when OP=I_ARI and OP3=CO_JM else '0';
   …
end PRACTICA;
```

Alias

Versión 4: VHDL profesional

```
architecture PRACTICA of CTRL is
…
constant I_ARI : STD_LOGIC_VECTOR(1 downto 0) := "10"; -- instr. arit.
constant CO_JM : STD_LOGIC_VECTOR(5 downto 0) := "111000"; -- jumpl
   alias OP:   STD_LOGIC_VECTOR(1 downto 0) is IR(31 downto 30); --
   Cod. Operación
…
alias OP:  STD_LOGIC_VECTOR(1 downto 0) is IR(31 downto 30); -- Cod. Operación
alias OP3:  STD_LOGIC_VECTOR(5 downto 0) is IR(24 downto 19); -- Cod. Operación 3
…
begin
   …
   PC_JUMP <= '1' when OP=I_ARI and OP3=CO_JM else '0';
   …
end PRACTICA;
```

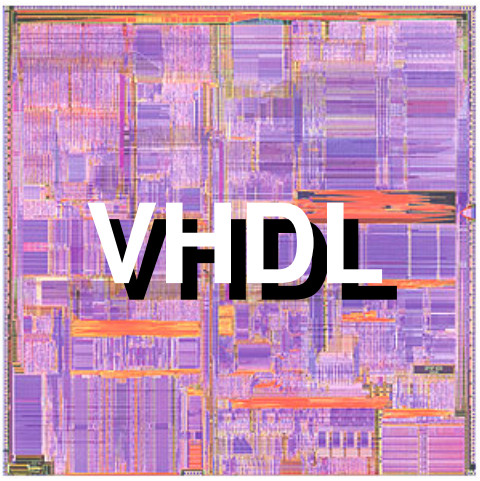

Alias

- El uso de alias puede llegar más lejos todavía

```
alias "<" is ieee.std_logic_unsigned."<"
             [std_logic_vector, std_logic_vector return boolean];
alias tipo_bus is ieee.std_logic_1164.std_logic_vector;
```

- Los alias se especifican en la parte declarativa de la arquitectura
- Generalmente es soportado por las herramientas de síntesis

VHDL Avanzado



Entidades parametrizadas
Instanciaciones Iterativas y Condicionales
- Generate
- Generic + Generate

Atributos
Alias
Funciones y procedimientos
Records
Mixed Language Design

Procedimientos en VHDL

- Los procedimientos se usan para encapsular una subrutina formada por instrucciones secuenciales:

```
procedure procedure_name ( formal_parameter_list ) is
  procedure_declarations
begin
  sequential statements
end procedure procedure_name;
```

- Al igual que en un proceso, dentro de un procedimiento se pueden declarar variables, tipos, constantes, subprocedimientos, archivos,...
- Los procedimientos se pueden llamar
  - desde procesos o
  - concurrentemente: En este caso se disparan cuando cambia alguno de sus parámetros declarados como entrada

Parámetros de los procedimientos

- Los parámetros de un procedimiento pueden tener tres direcciones

  In Inout Out

- Y pueden ser de tres clases:
  - Variable (por defecto para inout y out)
  - Constante (por defecto para in)
  - Señal
- Adicionalmente, se les debe dar un tipo:

```
procedure Procedure_1 (variable X, Y: inout Real);
procedure Proc_1 (constant In1: in Integer; variable O1: out Integer);
procedure Proc_2 (signal Sig: inout Bit);
```

- Finalmente, se puede dar un valor por defecto a un parámetro formal de modo IN.

Llamadas a Procedimientos

- Llamadas a procedimientos:

```
Procedure_1 (R, T);
```

- El efecto de esta sentencia es invocar el procedimiento `Procedure_1`. Esto implica:
  - Crear e inicializar variables locales,
  - Ejecutar las sentencias del cuerpo del procedimiento,
  - Terminada la última sentencia del procedimiento, decimos que el procedimiento retorna (return).
  - La siguiente sentencia que se ejecuta es la que está después de la llamada.

- A veces es útil retornar desde el medio de un procedimiento:

```
return
```

Ejemplo de procedimiento

- Ejemplo: Procedimiento para simular el envío de un byte por un puerto serie asíncrono (para probar una UART)

```
procedure send_data(constant data : in std_logic_vector(7 downto 0);
                    signal tx : out std_logic;
                    constant speed : in integer)
is
        constant tbit : time := 1000 ms / speed;
begin
        tx <= '0';
        wait for tbit;
        for i in 0 to 7 loop
                tx <= data(i); wait for tbit;
        end loop;
        tx <= '1';
        wait for tbit;
end;
```


Los procedimientos y la síntesis

- Los procedimientos pueden ser sintetizables.
  - La mayor limitación es que incluyan cláusulas wait.
- Al realizar la síntesis, se sustituye la llamada al procedimiento por las instrucciones secuenciales que éste contiene.
- Ejemplo:

```
procedure shift (constant data : in std_logic_vector;
                 constant desp : in integer;
                 signal res : out std_logic_vector) is
begin
        for i in data'range loop
                 if i-desp > 0 then
                         res(i) <= data(i-desp);
                 else
                         res(i) <= '0';
                 end if;
        end loop;
end;
```

Funciones en VHDL

- Las funciones difieren en los siguientes puntos con los procedimientos:
  - Todos sus parámetros son de entrada
  - No admiten variables como parámetros de entrada
  - Devuelven un valor (es su parámetro de salida)
  - No se pueden llamar concurrentemente

- Ejemplos de declaraciones:

```
function Func_1 (A,B,X: REAL) return REAL;
function "*" (a,b: Integer_new) return Integer_new;
function Add_Signals (signal In1,In2: REAL) return REAL;
```


Funciones puras e impuras

- Por defecto, las funciones son puras
  - O sea, que devuelven el mismo valor para los mismos parámetros de entrada
- Pero también se pueden declarar las funciones como impuras
  - Son las que pueden devolver distintos valores en sucesivas llamadas aunque los parámetros de entrada sean los mismos

```
variable number: Integer := 0;
impure function Func_5 (A: Integer) return Integer is
        variable counter: Integer;
begin
        counter := A * number;
        number := number + 1;
        return counter;
end Func_5;
```

Las funciones y la síntesis

- De igual manera a los procedimientos, las funciones también pueden ser sintetizables.
  - La mayor restricción es de igual forma las cláusulas wait.
- Ejemplo:

```
function INCREMENT (COUNT: std_logic_vector)
          return std_logic_vector
is
   variable TEMP: std_logic_vector(count'range);
begin
   if COUNT >= 255 then
      TEMP := (others => '0');
   else
      TEMP := COUNT + 1;
   end if;
   return TEMP;
end;
```


Funciones y procedimientos

- Overloading: Se puede definir funciones y procedimientos con
  - el mismo nombre pero
  - diferente número o tipo de parámetros formales.

- Ejemplo: librería NUMERIC_STD

```
function SHIFT_RIGHT (ARG: SIGNED; COUNT: NATURAL) return SIGNED;
function SHIFT_RIGHT (ARG: UNSIGNED; COUNT: NATURAL) return UNSIGNED;
```

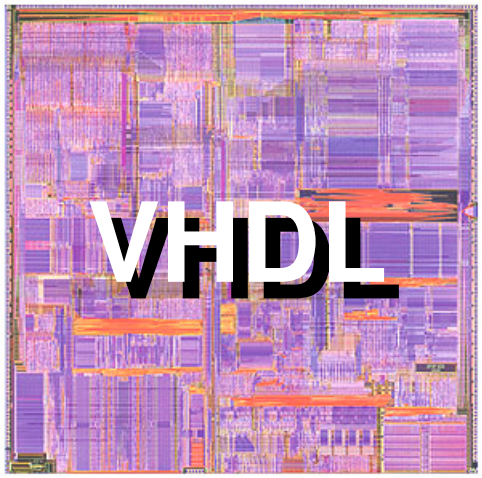
- Sobrecarga de símbolos de operadores

```
function "+" (L, R: UNSIGNED) return UNSIGNED;
function "+" (L, R: SIGNED) return SIGNED;
…
function "+" (L: SIGNED; R: INTEGER) return SIGNED;
```

Funciones y Procedimientos

- Para recordar:
  - Los **procedimientos** son generalizaciones de **sentencias**.
  - Las **funciones** son generalizaciones de **expresiones**.

VHDL Avanzado



Entidades parametrizadas
Instanciaciones Iterativas y Condicionales
- Generate
- Generic + Generate

Atributos
Alias
Funciones y procedimientos
Records
Mixed Language Design

Records

- Los records son tipos compuestos de VHDL
  - Conjuntos de datos de varios tipos
  - Opuestos a los arrays, que son conjuntos de un mismo tipo
- Modelo de declaración:

```
type record_type_name is record
    element_name : element type;
    element_name : element type;
    . . .
end record record_type_name;
```

- Ejemplo:

```
type RegName is (AX, BX, CX, DX);
type Operation is record
        Mnemonic : String (1 to 10);
        OpCode : Bit_Vector(3 downto 0);
        Op1, Op2, Res : RegName;
end record;
```


Records

- Para inicializarlos se listan todos sus contenidos ordenadamente entre paréntesis
  - También se pueden usar *aggregates*

```
variable Instr1, Instr2: Operation;
Instr1 := ("ADD AX, BX", "0001", AX, BX, AX);
Instr2 := ("ADD AX, BX", "0010", others => BX);
```

- Para acceder a los elementos de un record se usa el punto

```
variable Instr3 : Operation;
Instr3.Mnemonic := "MUL AX, BX";
Instr3.Op1 := AX;
```

- Los records pueden ser sintetizables.

Records en las interfaces

- Para un core IP complejo, la lista de puertos puede tener cientos de señales. Se complica:
  - Entender qué señales pertenecen a qué funcionalidades y
  - Agregar y quitar señales
- Cada modificación en la interfaz se debe hacer en tres sitios:
  - La declaración de la *entity*,
  - La declaración del *component* de esa entidad y
  - La instanciación del componente (*port map*).
- Mediante *records* la interfaz se hace más corta y legible y más mantenible.
- Las señales se agrupan por
  - funcionalidad y
  - dirección (*in* o *out*).

Records en las interfaces

- Ejemplo:

```
entity count8 is
port (
     clk : in std_logic;
     load : in std_logic;
     count : in std_logic;
     din : in std_logic_vector(7 downto 0);
     dout : out std_logic_vector(7 downto 0);
     zero : out std_logic);
end;
```

Records en las interfaces

- Los records se pueden declarar en un *package*.

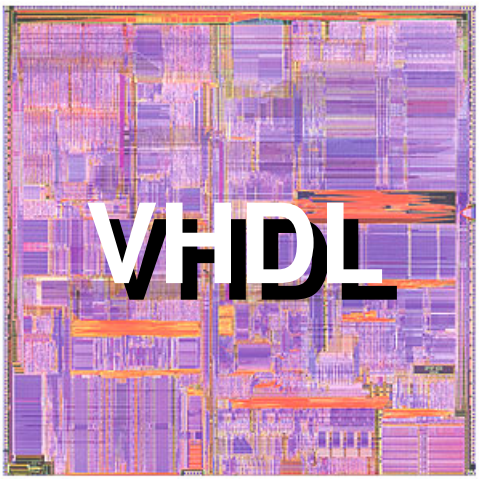
```
library ieee;
use ieee.std_logic_1164.all;

package count8_comp is
type count8_in_type is record
     load : std_logic;
     count : std_logic;
     din : std_logic_vector(7 downto 0);
end;

type count8_out_type is record
     dout : std_logic_vector(7 downto 0);
     zero : std_logic;
end;
end package;
```


Records en las interfaces

- Una modificación a la interfaz de la entidad implica una modificación en el record, en un solo sitio.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.count8_comp.all;

entity count8 is
port (
     clk : in std_logic;
     d : in count8_in_type;
     q : out count8_out_type);
end;
```

VHDL Avanzado



- Entidades parametrizadas
- Instanciaciones Iterativas y Condicionales
  - Generate
  - Generic + Generate
- Atributos
- Alias
- Funciones y procedimientos
- Records
- **Mixed Language Design**

VHDL instanciando Verilog

- Una vez declarado un componente en un modulo Verilog, se puede instanciar como cualquier otro componente VHDL.
- Las únicas restricciones para instanciar un módulo Verilog dentro de VHDL son:
  - No se permiten UDPs (user-defined primitive).
  - Los puertos deben tener nombre (Verilog permite definir módulos con puertos sin nombre).
  - Los puertos no se conectan a *pass switches* bidireccionales.
- Los identificadores VHDL para nombre de componente, puertos y parámetros genéricos son los mismos que en el módulo Verilog, excepto:
  - El identificador en Verilog no es un identificador válido VHDL 1076-1987.
  - El identificador en Verilog no es único cuando se ignoran mayúsculas y minúsculas.