Tema 5:

Especificación usando VHDL'08

Diseño automático de sistemas

José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

VHDL
evolución del estándar

- VHDL es un estándar en evolución:
  - VHDL 1076-1987: Primer estándar.
  - VHDL 1076-1993: Algunos cambios, en particular adición de variables compartidas.
  - VHDL 1076-2000: Añade los tipos protegidos para dar sentido a las variables compartidas.
  - VHDL 1076-2002: Relaja las restricciones en el uso de puertos de tipo buffer.
  - VHDL 1076-2008: Incorpora una colección extensa de cambios.

- El subconjunto de síntesis soportado por ISE está basado en VHDL'87/93
  - Los cambios hasta VHDL'02, afectan principalmente al modelado de testbenches.

- Nuevas herramientas como Vivado comienzan a soportar VHDL'08
  - Porque incorpora cambios que afectan al subconjunto sintetizable.

Puertos

- En VHDL'08 es posible leer puertos de salida dentro de la arquitectura
  - En versiones anteriores, era necesario usar una señal intermedia que se conectase a dicho puerto o declararlo de tipo buffer.

VHDL'93

```
entity rShifter is
port (
  ...
  dout : out std_logic_vector(7 downto 0)
);
end shifter;

architecture syn of rShifter is
  signal cs : std_logic_vector(dout'range);
begin
 process (rst_n, clk)
  begin
    if rst_n='0' then
      cs <= (others=>'0');
    elsif rising_edge(clk) then
      if sht='1' then
        cs <= din & cs(7 downto 1);
      end if;
    end if;
  end process;
  dout <= cs;
end syn;
```

VHDL'08

```
entity rShifter is
port (
  ...
  dout : out std_logic_vector(7 downto 0)
);
end shifter;

architecture syn of rShifter is
begin
 process (rst_n, clk)
  begin
    if rst_n='0' then
      cs <= (others=>'0');
    elsif rising_edge(clk) then
      if sht='1' then
        dout <= din & dout(7 downto 1);
      end if;
    end if;
  end process;
end syn;
```

Instanciación

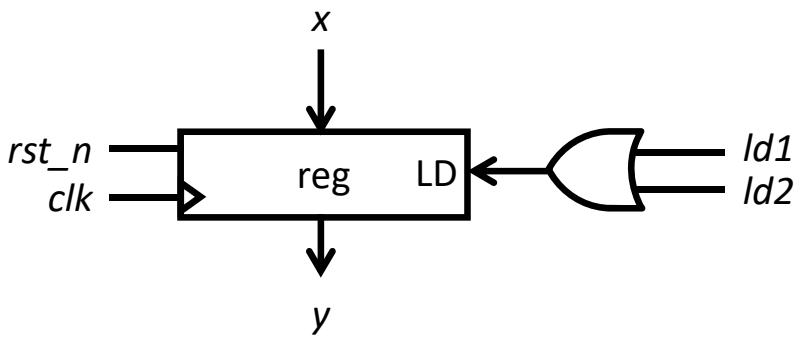
- VHDL'08 admite expresiones en la instanciación de componentes
  - En versiones anteriores era necesario usar una señal auxiliar.
  - Es útil para especificar lógica pegamento o conversión entre tipos

VHDL'93

```
...
reg : register
  port map (
    rst_n   => rst_n,
    clk     => clk,
    dataIn  => x,
    ld      => ldAux,
    dataOut => y
  );

ldAux <= ld1 or ld2;
...
```

VHDL'08

```
...
reg : register
  port map (
    rst_n   => rst_n,
    clk     => clk,
    dataIn  => x,
    ld      => ld1 or ld2,
    dataOut => y
  );
...
```



Comentarios

- En VHDL'08 se permite comentar bloques completos de código
  - En versiones anteriores era necesario comentar línea a línea
  - Usa la misma sintaxis que C /* y */

VHDL'93

```
...

-- process (rst_n, clk)
-- begin
--  if rst_n='0' then
--    q <= '0';
--  elsif rising_edge(clk) then
--    q <= d;
--  end if;
-- end process;

...
```

VHDL'08

```
...

/*
 process (rst_n, clk)
 begin
  if rst_n='0' then
    q <= '0';
  elsif rising_edge(clk) then
    q <= d;
  end if;
 end process;
*/

...
```

J.M. Mendías 2016

Procesos

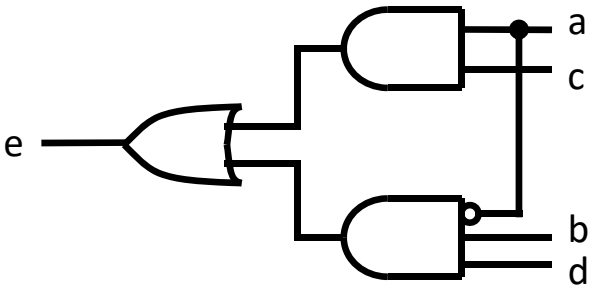
- En VHDL-08 la lista de sensibilidad de un proceso puede abreviarse
  - La palabra reservada **all** representa a todas las señales leídas por el proceso.
  - Útil para procesos combinacionales que debe tener listas de sensibilidad completas.
    - Evita que un olvido provoque que las herramientas infieran latches.

VHDL'93

```
process (a, b, c, d)
begin
  if a='1' then
    e <= c;
  elsif b='1' then
    e <= d;
  else
    e <= '0';
  end if;
end process;
```

VHDL'08

```
process (all)
begin
  if a='1' then
    e <= c;
  elsif b='1' then
    e <= d;
  else
    e <= '0';
  end if;
end process;
```



Expresiones

- En VHDL'08 los literales de tipo cadena de bits tienen capacidades expresivas aumentadas:
  - Soportan base 10.
  - Soportan la indicación explícita de anchura.
  - Soportan la indicación explícita de signo (realizando la extensión que corresponda).
  - Permiten el uso de valores metalógicos en su definición.
  - En versiones anteriores los literales octales/hexadecimales solo podían representar constantes de anchura múltiplo de 3/4.

VHDL'93

literal	equivale a
X"0f"	"0000_1111"
O"7"	"111"

VHDL'08

literal	equivale a
6X"0f"	"00_1111"
6X"xf"	"XX_1111"
6UX"f"	"00_1111"
6SX"f"	"11_1111"
6UO"7"	"000_111"
6D"13"	"00_1101"

Expresiones

- VHDL'08 permite usar vectores en agregados
  - Versiones anteriores solo permitían asignar colecciones de elementos individuales

VHDL'93	equivale a	VHDL'08
('1', '0', others => '0')	"1000_0000"	("10", others => '0')
(3 downto 0=>'0', others =>'1')	"1111_0000"	("0000", others =>'1')

  - También se usa cuando el destino es un vector agregado

VHDL'93

```
signal x, y : unsigned (7 downto 0);
signal s    : std_logic_vector(7 downto 0);
signal cout : std_logic;
signal aux  : unsigned (8 downto 0);
...
aux <= ('0' & x) + ('0' & y);
s <= std_logic_vector( aux(7 downto 0) );
cout <= std_logic( aux(8) );
```

VHDL'08

```
...
(cout, s) <= std_logic_vector( ('0' & x) + ('0' & y) );
```

J.M. Mendías 2016

Expresiones

- VHLD'08 incorpora las funciones **`maximun`** y **`minimum`** para escalares y vectores.
  - No existentes en versiones anteriores.
  - Usa los correspondientes operadores relacionales definidos para el tipo.
  - El paquete **`numeric_std`** los define con argumentos **`unsigned`** y **`signed`**

VHDL'93

```
process (x, y)
begin
  if x > y then
    z <= x;
  else
    z <= y;
  end if;
end process;
```

VHDL'08

```
process (all)
begin
  z <= maximum(x,y);
end process;
```

Expresiones

- VHDL'08 completa los operadores infijos de rotación y desplazamiento de sus paquetes aritméticos
  - En versiones anteriores solo estaba definido para vectores de tipo **`bit`** / **`boolean`** y el paquete **`numeric_std`** no definía desplazamientos aritméticos para los tipos **`unsigned`** y **`signed`**

operador	bit / std_logic
sll	desplazamiento lógico a izquierdas
srl	desplazamiento lógico a derechas
sla	desplazamiento aritmético a izquierdas
sra	desplazamiento aritmético a derechas
rol	rotación a izquierdas
ror	rotación a derechas

Expresiones

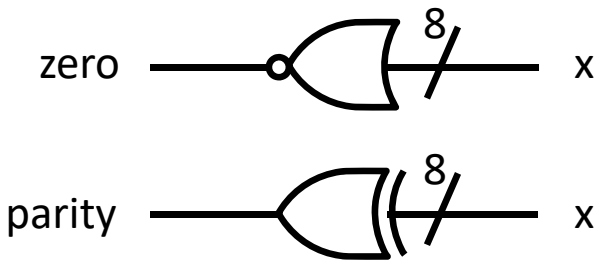
- VHDL'08 define una nueva colección de operadores lógicos unarios para argumentos de tipo vector de **`bit`** / **`boolean`**.
  - En versiones anteriores los operadores eran binarios
  - El paquete **`std_logic_1164`**, define la versión con argumento **`std_logic_vector`**.

VHDL'93

```
...
signal x : std_logic_vector(7 downto 0);
signal zero : std_logic;
signal parity : std_logic;
...
process (data)
  variable aux : std_logic;
begin
 aux := '0';
 for i in data'range loop
   aux := aux nor x(i);
 end loop;
 zero <= aux;
end process;

parity <= x(0) xor x(1) xor x(2) xor x(3)
  xor x(4) xor x(5) xor x(6) xor x(7);
```

VHDL'08

```
...
signal x : std_logic_vector(7 downto 0);
signal zero : std_logic;
...
zero <= nor( x );
parity <= xor( x );
...
```



Expresiones

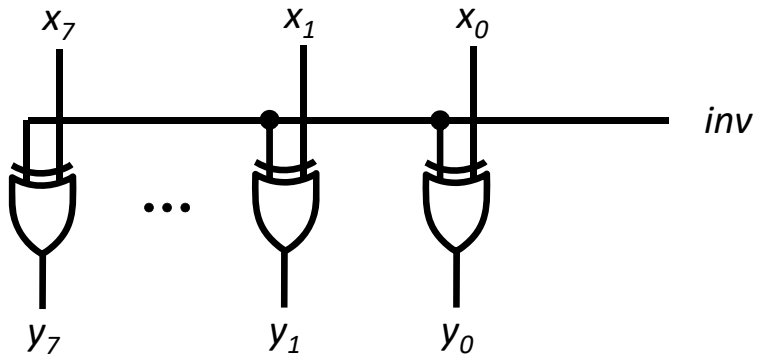
- VHDL'08 permite, además, que los operadores lógicos binarios tomen un argumento escalar y el otro vectorial
  - Versiones anteriores ambos argumentos tenían que ser del mismo tipo y anchura.
  - El estándar los define para argumentos **`bit/boolean`**, y el paquete **`std_logic_1164`** para la versión con argumento **`std_logic`**.

VHDL'93

```
...
signal x : std_logic_vector(7 downto 0);
signal y : std_logic_vector(7 downto 0);
signal inv : std_logic;
...
process (dataIn)
begin
  for i in dataIn'range loop
    y(i):= x(i) xor inv;
  end loop;
end process;
```

VHDL'08

```
...
signal x : std_logic_vector(7 downto 0);
signal y : std_logic_vector(7 downto 0);
signal inv : std_logic;

...
y <= x xor inv;
...
```



Expresiones

- VHDL'08 define el nuevo operador unario ??
  - Devuelve un valor de tipo **`boolean`** tomando como argumento de tipo **`bit`** (asumiendo codificación directa: true='1' / false='0').
  - El paquete **`std_logic_1164`**, define la versión para argumentos **`std_logic`**.
  - En muchas ocasiones el operador puede quedar implícito.

VHDL'93

```
process (a, b)
begin
 if a='1' and b='1' then
   data <= ...;
 else
   data <= ...;
 end if;
end process;
```

VHDL'08

```
process (all)
begin
 if ?? a and b then
   data <= ...;
 else
   data <= ...;
 end if;
end process;
```

```
process (all)
begin
 if a and b then
   data <= ...;
 else
   data <= ...;
 end if;
end process;
```

Expresiones

- VHDL'08 define un nuevo conjunto de operadores relacionales que devuelven valores de los tipos **`bit`** o **`std_logic`**
  - Si comparan metalógicos, asimilan 'H' con '1', 'L' con '0' y '-' con indiferencia.
  - En versiones anteriores los operadores relacionales devuelven **`boolean`**

boolean	bit/std_logic
=	?=
/=	?/=
<	?<
<=	?<=
>	?>
>=	?>=

x	y	x = y	x ?= y
'0'	'0'	true	'1'
'0'	'1'	false	'0'
'0'	'L'	false	'1'
'0'	'H'	false	'0'
'0'	'-'	false	'1'
'1'	'1'	true	'1'
'1'	'L'	false	'0'
'1'	'H'	false	'1'
'1'	'-'	false	'1'

VHDL'93

```
process (x, y)
begin
 if x=y then
   eq <= '1'
 else
   eq <= '0'
 end if;
 ...
end process;
```

VHDL'08

```
process (all)
begin
 eq <= x ?= y;
 ...
end process;
```

Sentencias

- VHDL'08 define un nuevo tipo de sentencia de selección condicional que implícitamente usa el operador ?= para comparar valores

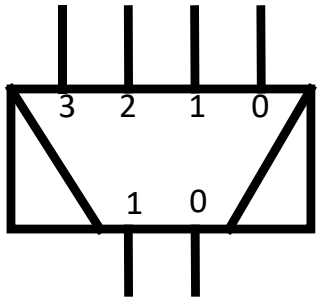
VHDL'93

```
process (x)
begin
  case x is
    when "1000" => y <= "11";
    when "1001" => y <= "11";
    ...
    when "1111" => y <= "11";
    when "0100" => y <= "10";
    ...
    when "0111" => y <= "10";
    when "0010" => y <= "01";
    when "0011" => y <= "01";
    when others => y <= "00";
  end case;
end process;
```

VHDL'08

```
process (x)
begin
  case? x is
    when "1---" => y <= "11";
    when "01--" => y <= "10";
    when "001-" => y <= "01";
    when others => y <= "00";
  end case;
end process;
```

```
with x select?
  y <= "11" when "1---",
       "10" when "01--",
       "01" when "001-",
       "00" when others;
```



Sentencias

- En VHDL-08 se amplian las posibilidades de las sentencia generate
  - En versiones anteriores solo existe **`if`** y **`for`**
  - Se añade case y además los **`if`** pueden tener rama **`else`**/**`elsif`**

VHDL'93

```
outputRegisters:
if REGOUTPUTS generate
  process (rst_n, clk)
  begin
    if rst_n='0' then
      dataOut <= '0';
    elsif rising_edge(clk) then
      dataOut <= count;
    end if;
  end process;
end generate;

outputSignals:
if not REGOUTPUTS generate
  dataOut <= count;
end generate;
```

VHDL'08

```
outputRegisters:
if REGOUTPUTS generate
begin
  process (rst_n, clk)
  begin
    ...
  end process;
else
  dataOut <= count;
end generate;
```

```
outputRegisters:
case REGOUTPUTS generate
  when true =>
    process (rst_n, clk)
    begin
      ...
    end process;
  when false =>
     dataOut <= count;
end generate;
```

Sentencias

- En VHDL'08 es posible usar dentro de un proceso sentencias de asignación condicional o asignación selectiva de señal
  - En versiones anteriores, este tipo de asignaciones solo puede usarse en el ámbito de una arquitectura.

VHDL'93

```
c <= (a and b) when d='1'
     else (a or b);
```

```
process (all)
begin
  if  d='1' then
    c <= a and b;
  else
    c <= a or b;
  end if;
end process;
```

```
process (all)
begin
  c <= (a and b) when d='1'
       else (a or b);
end process;
```

VHDL'08

```
with a+b select
  c <= d when "0000",
       not d when "1111",
       '1' when others;
```

```
process (all)
begin
  case a+b is
    when "0000" => c <= d;
    when "1111" => c <= not d;
    when others => c <= '1';
  end case;
end process;
```

```
process (all)
begin
  with a+b select
    c <= d when "0000",
         not d when "1111",
         '1' when others;
end process;
```

Declaraciones

- VHDL'08 permite la referencia a múltiples paquetes a través de un único nombre usando la construcción **`context`**

VHDL'93

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.common.all;

entity ... is
  ...
end ...;
```

VHDL'08

```
context stdIntegerArith is
  library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
  use work.common.all;
end;
```

```
context work.stdIntegerArith;

entity ... is
  ...
end ...;
```

Declaraciones

- VHDL'08 permite que en la declaración de tipos vectoriales (o registro) el número de elementos no esté definido.
  - En versiones anteriores era ilegal.

VHDL'93

```
type ramType1 is array (0 downto 2**10-1) of std_logic_vector (7 downto 0);
type ramType2 is array (0 downto 2**12-1) of std_logic_vector (15 downto 0);

signal ram1 : ramType1;
signal ram2 : ramType2;
```

VHDL'08

```
type ramType is array (natural range <>) of std_logic_vector;

signal ram1 : ramType (0 downto 2**10-1)(7 downto 0);
signal ram2 : ramType (0 downto 2**12-1)(15 downto 0);
```

Genéricos

- En VHDL-08 es posible que en la declaración de genéricos se referencie a otros genéricos
  - En versiones anteriores era ilegal-

VHDL'08

```
entity foo is
  generic (
    DWIDTH : natural;
    INIT   : std_logic_vector(DWIDTH-1 downto 0)
  );
  port (
    dataIn  : std_logic_vector(DWIDTH-1 downto 0);
    dataOut : std_logic_vector(DWIDTH-1 downto 0)
  );
end foo;
```

Genéricos

- En VHDL-08 los genéricos pueden ser también tipos, subprogramas y paquetes
  - En versiones anteriores solo podían ser constantes.

```
entity incrementer is
  generic (
    type data_type;
    function increment (x: data_type) return data_type
  );
  port (
    dataIn  : in  data_type;
    dataOut : out data_type;
    inc     : in  std_logic
);
end incrementer;

architecture syn of incrementer is
begin
  dataOut <= increment(dataIn) when inc = '1';
end syn;
```

VHDL'08

```
function addOne( x: integer )
  return integer is
begin
  return x + 1;
end;
...
incr_inst : incrementer
  generic map (
    data_type => integer,
    increment => addOne
  )
  port map (
    dataIn  => dataIn,
    dataOut => dataOut,
    inc     => inc
);
```

Genéricos

- En VHDL-08 es posible que los paquetes y subprogramas tengan genéricos.
  - En versiones anteriores solo los tenían las entidades.

VHDL'08

```
package fixed_generic_pkg is
  generic (
    fixed_round_style    : fixed_round_style_type    := fixed_round;
    fixed_overflow_style : fixed_overflow_style_type := fixed_saturate;
    fixed_guard_bits     : natural                   := 3;
    no_warning           : boolean                   := false
);
...
end package fixed_generic_pkg;
```

  - Para que estos paquetes puedan ser usados deben instanciarse:

VHDL'08

```
package fixed_pkg is new IEEE.fixed_generic_pkg
  generic map (
    fixed_round_style    => IEEE.fixed_float_types.fixed_round,
    fixed_overflow_style => IEEE.fixed_float_types.fixed_saturate,
    Fixed_guard_bits     => 3,
    no_warning           => false
  );
```

Paquetes estándar

- VHDL'08 incorpora al estándar los paquetes de uso generalizado.
- Predefinido:
  - `standard`: tipos primitivos del lenguaje (`boolean`, `bit`, `integer`, `real` ...)
- Biblioteca STD:
  - `textio`: E/S de los tipos primitivos del lenguaje.
- Biblioteca IEEE:
  - `math_real`: funciones matemáticas sobre el tipo `real`
  - `math_complex`: tipo complejo (`complex`) y funciones matemáticas sobre él.
  - sintetizables:
    - `numeric_bit` / `numeric_bit_unsigned`: aritmética entera sobre vectores de bits.
    - `std_logic_1164`: tipo multivaluado estándar (`std_logic`) y E/S del mismo.
    - `std_logic_textio`: incluído por compatibilidad hacia atrás (definido como vacío).
    - `numeric_std` / `numeric_std_unsigned`: aritmética entera sobre vectores del tipo multivaluado estándar.
    - `fixed_float_types` / `fixed_generic_pkg` / `fixed_pkg`: aritmética en punto fijo sobre vectores del tipo multivaluado estándar.
  - `float_generic_pkg` / `float_pkg`: aritmética real sobre vectores del tipo multivaluado estándar.

J.M. Mendías
2016

Paquetes estándar

ieee.numeric_std_unsigned

- VHDL'08 define un nuevo paquete que permite la aritmética sin signo con argumentos de tipo std_logic_vector
  - Equivale a trabajar con datos de tipo unsigned del paquete numeric_std.
  - Evita la sobrecarga que suponen los casting y las conversiones de tipo.

VHDL'93

```
library ieee;
use ieee.numeric_std.all;

architecture syn of adder is
  signal x, y, s : std_logic_vector(7 downto 0);
begin
  s <= std_logic_vector(unsigned(x) + unsigned(y));
end syn;
```

VHDL'08

```
library ieee;
use ieee.numeric_std_unsigned.all;

architecture syn2 of adder is
  signal x, y, s : std_logic_vector(7 downto 0);
begin
  s <= x + y;
end syn2;
```

Paquetes estándar

aritmética en punto fijo (i)

- VHDL'08 define un nuevo paquete genérico que permite la aritmética en punto fijo. Se compone de 3 elementos:

- fixed_float_types: define los tipos enumerados necesarios para configurar el paquete de punto fijo.
  - fixed_overflow_style_type: indica qué hacer cuando el resultado de una operación es demasiado grande para ser representado.
    - fixed_saturate: devuelve el máximo valor representable.
    - fixed_wrap: desecha los bits más significativos sobrantes (puede haber cambio de signo).
  - fixed_round_style_type: indica qué hacer cuando el resultado de una operación requiere más bits de los disponibles para representarlo.
    - fixed_truncate: desecha los bits menos significativos sobrantes
    - fixed_round: aplica un algoritmo de redondeo.
  - round_type: indica el algoritmo de redondeo a aplicar cuando sea necesario.
    - round_nearest: redondea al número representable más cercano.
    - round_inf: redondea hacia el número representable más cercano al infinito positivo.
    - round_neginf: redondea hacia el número representable más cercano al infinito negativo.
    - round_zero: redondea el número representable más cercano al cero.

Paquetes estándar

aritmética en punto fijo (ii)

- fixed_generic_pkg: define los el tipo y operadores.

```
package fixed_generic_pkg is
  generic (
    fixed_round_style    : fixed_round_style_type    := fixed_round;
    fixed_overflow_style : fixed_overflow_style_type := fixed_saturate;
    fixed_guard_bits     : NATURAL                   := 3;
    no_warning           : BOOLEAN                   := false
);

...

end package fixed_generic_pkg;
```

- fixed_pkg: Realiza una instanciación concreta del tipo.
  - El diseñador puede realizar otras instanciaciones distintas a su conveniencia.

```
package fixed_pkg is new IEEE.fixed_generic_pkg
  generic map (
    fixed_round_style    => IEEE.fixed_float_types.fixed_round,
    fixed_overflow_style => IEEE.fixed_float_types.fixed_saturate,
    fixed_guard_bits     => 3,
    no_warning           => false
  );
```

Paquetes estándar

ieee.fixed_generic_pkg (i)

- El paquete **`fixed_generic`** define 2 tipos de datos vectoriales que, basados en **`std_logic`**, son interpretables como reales codificados en punto fijo.
  - Asume que el punto se halla siempre entre la posición 0 y -1 del vector.
  - El subrango positivo del vector codifica los bits enteros del número.
  - El subrango negativo del vector codifica los bits decimales del número.
  - El tipo **`ufixed`** especifica un número real sin signo representado en binario puro.

```
type ufixed is array (natural range <>) of std_logic;
```

  - El tipo **`sfixed`** especifica un número real con signo representado en C2.

```
type sfixed is array (natural range <>) of std_logic;
```

ufixed (3 downto -3)	representa a
"0110_100"	+6.5
"1100_101"	+12.625

sfixed (3 downto -3)	representa a
"0110_100"	+6.5
"1100_101"	-3.375

- Además define operadores sobrecargados para dichos tipos:
  - Aritméticos, lógicos, relacionales, funciones de conversión, etc.

J.M. Mendías
2016

Paquetes estándar

ieee.fixed_generic_pkg (ii)

Perfiles de las funciones aritméticas

```
function "+" (l, r : ufixed) return ufixed;
function "+" (l, r : sfixed) return sfixed;
function "+" (l : ufixed; r : REAL) return ufixed;
function "+" (l : REAL; r : ufixed) return ufixed;
function "+" (l : ufixed; r : NATURAL) return ufixed;
function "+" (l : NATURAL; r : ufixed) return ufixed;
function "+" (l : sfixed; r : REAL) return sfixed;
function "+" (l : REAL; r : sfixed) return sfixed;
function "+" (l : sfixed; r : INTEGER) return sfixed;
function "+" (l : INTEGER; r : sfixed) return sfixed;
```

Perfiles de las funciones relacionales

```
function ">" (l, r : ufixed) return BOOLEAN;
function ">" (l, r : sfixed) return BOOLEAN;
function ">" (l : ufixed; r : NATURAL) return BOOLEAN;
function ">" (l : NATURAL; r : ufixed) return BOOLEAN;
function ">" (l : ufixed; r : REAL) return BOOLEAN;
function ">" (l : REAL; r : ufixed) return BOOLEAN;
function ">" (l : sfixed; r : INTEGER) return BOOLEAN;
function ">" (l : INTEGER; r : sfixed) return BOOLEAN;
function ">" (l : sfixed; r : REAL) return BOOLEAN;
function ">" (l : REAL; r : sfixed) return BOOLEAN;
```

Paquetes estándar

ieee.fixed_generic_pkg (iii)

- A diferencia de los operadores de **numeric_std**, este paquete está diseñado para que nunca haya posibilidad de overflow
  - El rango del resultado queda definido según el rango de los operandos .

operación	rango del resultado
A + B, A – B	Max(A'left, B'left) + 1 downto Min(A'right, B'right)
A * B	A'left + B'left + 1 downto A'right + B'right
A rem B, A mod B (con signo)	Min(A'left, B'left) downto Min(A'right, B'right)
A / B (con signo)	A'left - B'right + 1 downto A'right - B'left
reciprocal(A) (con signo)	-A'right downto -A'left - 1
abs(A), -A	A'left + 1 downto A'right
A / B (sin signo)	A'left - B'right downto A'right - B'left – 1
A mod B (sin signo)	B'left downto Min(A'right, B'right)
reciprocal(A) (sin signo)	-A'right + 1 downto -A'left

  - Si uno de los argumentos es **real**/**integer**/**natural**, previamente a operar con él se pasa a representación en punto fijo con el rango del otro argumento.

Paquetes estándar

ieee.fixed_generic_pkg (iv)

- Las funciones: **ufixed_high** / **ufixed_low** / **sfixed_high** / **sfixed_low**
  - Permiten definir el rango del resultado sin memorizar las reglas de dimensionado.

Perfiles de las funciones auxiliares para dimensionado

```
function ufixed_high (
  left_index, right_index   : INTEGER;
  operation                 : CHARACTER := 'X';
  left_index2, right_index2 : INTEGER   := 0
) return INTEGER;

function ufixed_high (
  size_res  : ufixed;
  operation : CHARACTER := 'X';
  size_res2 : ufixed
) return INTEGER;
```

  - Se usan cuando se desea que no haya pérdida de precisión.

```
signal a : ufixed (5 downto -3);
signal b : ufixed (7 downto -9);
signal c : ufixed( ufixed_high(5,-3,'/',7,-9) downto ufixed_low(5,-3,'/',7,-9) );
signal d : ufixed( ufixed_high(a,'*',b) downto ufixed_low(a,'*',b) );
...
c <= a / b;
d <= a * b;
```

Paquetes estándar

ieee.fixed_generic_pkg (v)

- La función **resize** se usa cuando se desea que el resultado tenga un rango determinado
  - Cuando hay pérdida de precisión se aplican reglas de saturación y redondeo.

Perfiles de las funciones de redimensionado

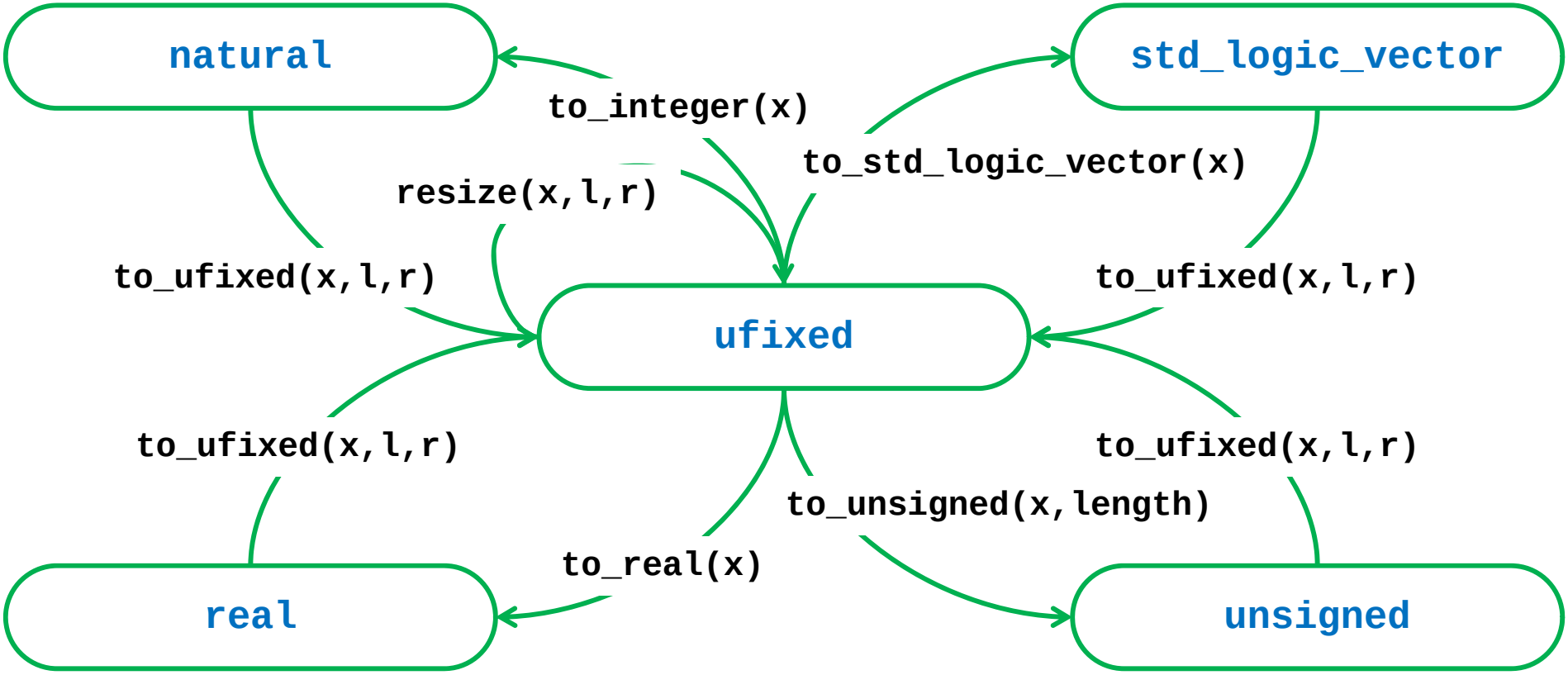
```
function resize (
  arg : ufixed;
  left_index : INTEGER; right_index : INTEGER;
  overflow_style : fixed_overflow_style_type := fixed_overflow_style;
  round_style    : fixed_round_style_type    := fixed_round_style
) return ufixed;

function resize (
  arg : ufixed;
  size_res  : ufixed;
  overflow_style : fixed_overflow_style_type := fixed_overflow_style;
  round_style    : fixed_round_style_type    := fixed_round_style
) return ufixed;
```

```
signal a : ufixed (5 downto -3);
signal b : ufixed (7 downto -9);
signal c, d: ufixed (7 downto -3);
...
c <= resize( a+b, 7, -3 ); -- también c <= resize( a+b, c'high, c'low);
d <= resize( a-b, d );
```

Paquetes estándar

ieee.fixed_generic_pkg (vi)

- Para pasar de un tipo a otro se usan funciones de conversión.
  - Cuando hay pérdida de precisión se aplican reglas de saturación y redondeo.



J.M. Mendías
2016

Paquetes estándar

ieee.fixed_generic_pkg (vii)

Perfiles de las funciones de conversión

```
function to_ufixed (
  arg : REAL;
  left_index : INTEGER; right_index : INTEGER;
  overflow_style : fixed_overflow_style_type := fixed_overflow_style;
  round_style    : fixed_round_style_type    := fixed_round_style;
  guard_bits     : NATURAL                   := fixed_guard_bits
) return ufixed;

function to_ufixed (
  arg : REAL;
  size_res : ufixed;
  overflow_style : fixed_overflow_style_type := fixed_overflow_style;
  round_style    : fixed_round_style_type    := fixed_round_style;
  guard_bits     : NATURAL                   := fixed_guard_bits
) return ufixed;
```

```
signal pi, e : ufixed (7 downto -9);
...
pi <= to_ufixed( 3.14, 7, -9 ); -- también pi <= to_ufixed( 3.14, pi'high, pi'low);
e  <= to_ufixed( 2.72, d);
```

Paquetes estándar

ieee.fixed_generic_pkg (viii)

```
package my_fixed_pkg is new IEEE.fixed_generic_pkg
  generic map (
    fixed_round_style    => IEEE.fixed_float_types.fixed_truncate,
    fixed_overflow_style => IEEE.fixed_float_types.fixed_wrap,
    fixed_guard_bits     => 0,
    no_warning           => true
  );
```

```
library ieee;
use ieee.std_logic_1164.all;
entity adder is
generic(
  m : integer := 8      -- Num. bits enteros
  n : integer := 8 );   -- Num. bits decimales
port(
  x :  in std_logic_vector(m+n-1 downto 0);
  y :  in std_logic_vector(m+n-1 downto 0);
  s : out std_logic_vector(m+n-1 downto 0) );
end adder;

library ieee;
use work.my_fixed_pkg.all;
architecture syn of adder is
begin
  s <= to_std_logic_vector( resize(to_ufixed(x,m-1,-n)+to_ufixed(y,m-1,-n),m-1,-n) );
end syn;
```

Puede instanciarse el paquete
para adaptar la lógicaq a las necesidades

Paquetes estándar

ieee.fixed_generic_pkg (ix)

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

entity adder is
generic(
  m : integer := 8      -- Num. bits enteros
  n : integer := 8 );   -- Num. bits decimales
port(
  x :  in std_logic_vector(m+n-1 downto 0);
  y :  in std_logic_vector(m+n-1 downto 0);
  s : out std_logic_vector(m+n-1 downto 0) );
end adder;

library ieee_proposed;
use ieee_proposed.fixed_pkg.all;

architecture syn of adder is
begin
 s <= to_std_logic_vector( resize(to_ufixed(x,m-1,-n)+to_ufixed(y,m-1,-n),m-1,-n) );
end syn;
```

Xilinx Vivado soporta un el paquete de punto fijo equivalente pero escrito en VHDL'93. Se encuentra en la biblioteca IEEE_proposed

Acerca de *Creative Commons*

- Licencia CC (Creative Commons)
  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

Reconocimiento (*Attribution*):
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No comercial (*Non commercial*):
La explotación de la obra queda limitada a usos no comerciales.

Compartir igual (*Share alike*):
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/