Práctica 3

Redes iterativas y unidad multifunción

toc

Objetivos

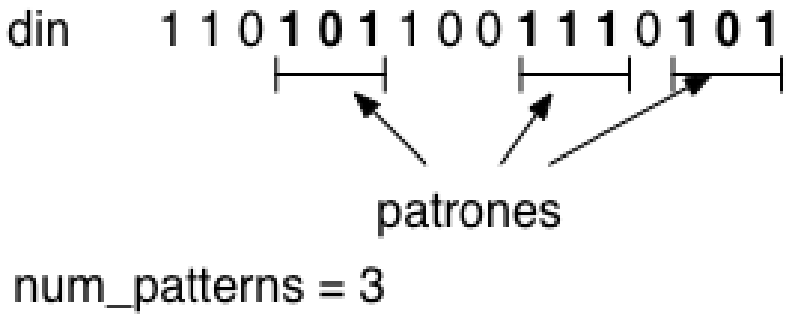
- Aprender a utilizar la sentencia `generate`.
- Aprender a utilizar genéricos para parametrizar el tamaño de los diseño en VHDL
- Aprender a crear paquetes (`package`) en VHDL.
- Aprender a manejar operadores aritméticos de la librería *`numeric_std`*.
- Optimizar la implementación de unidades funcionales.
- Demostradores:
  - Red iterativa
  - Unidad Multifunción

Objetivos

- Práctica 3.a: Red iterativa.
  - Reconocedor del patrón “1X1”.
- Práctica 3.b: Unidad multifunción.
  - Sumador, restador, MAX, MIN, ABS.
- Práctica 3.c: Apartado AVANZADO que haréis en el laboratorio.

Especificación

- **Spec 1**: La red iterativa debe calcular el número de veces que aparece el patrón 1x1 (siendo x cualquier valor binario) sin solapamiento en el vector de entrada del circuito.



- **Spec 2**: La red iterativa es un circuito combinacional. No posee ni señal de reloj ni de reset.
- **Spec 3**: El circuito tiene un puerto de entrada, `din`, de ancho parametrizable definido mediante un genérico.

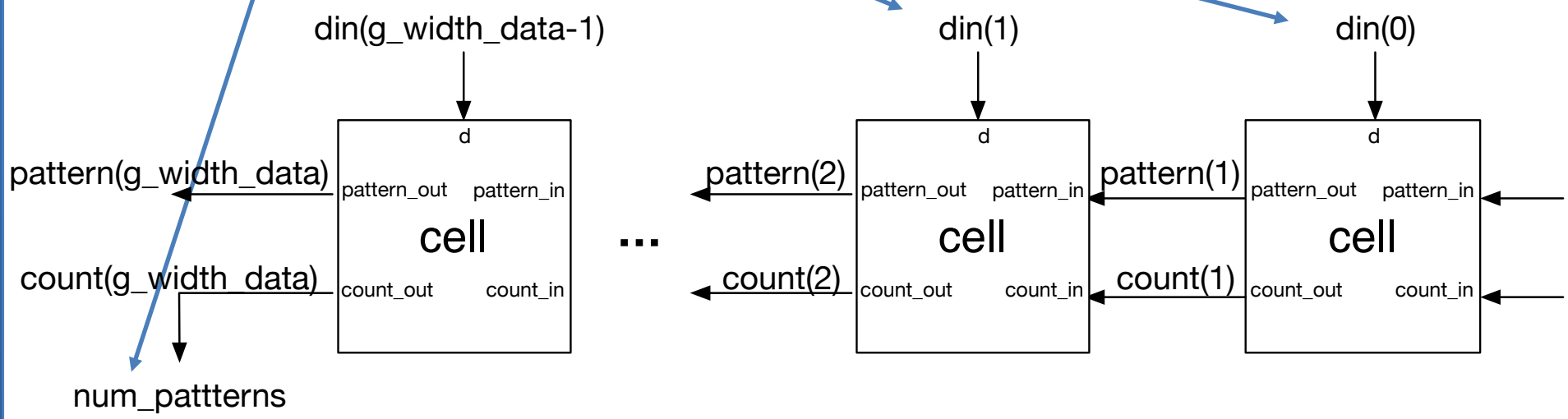
Especificación

- **Spec 4**: El circuito tiene un puerto de salida, num_patterns, de un ancho parametrizable definido mediante un genérico. El puerto num_pattern devuelve, en formato complemento a 2, el número de veces que aparece el patrón en din.
- **Spec 5**:

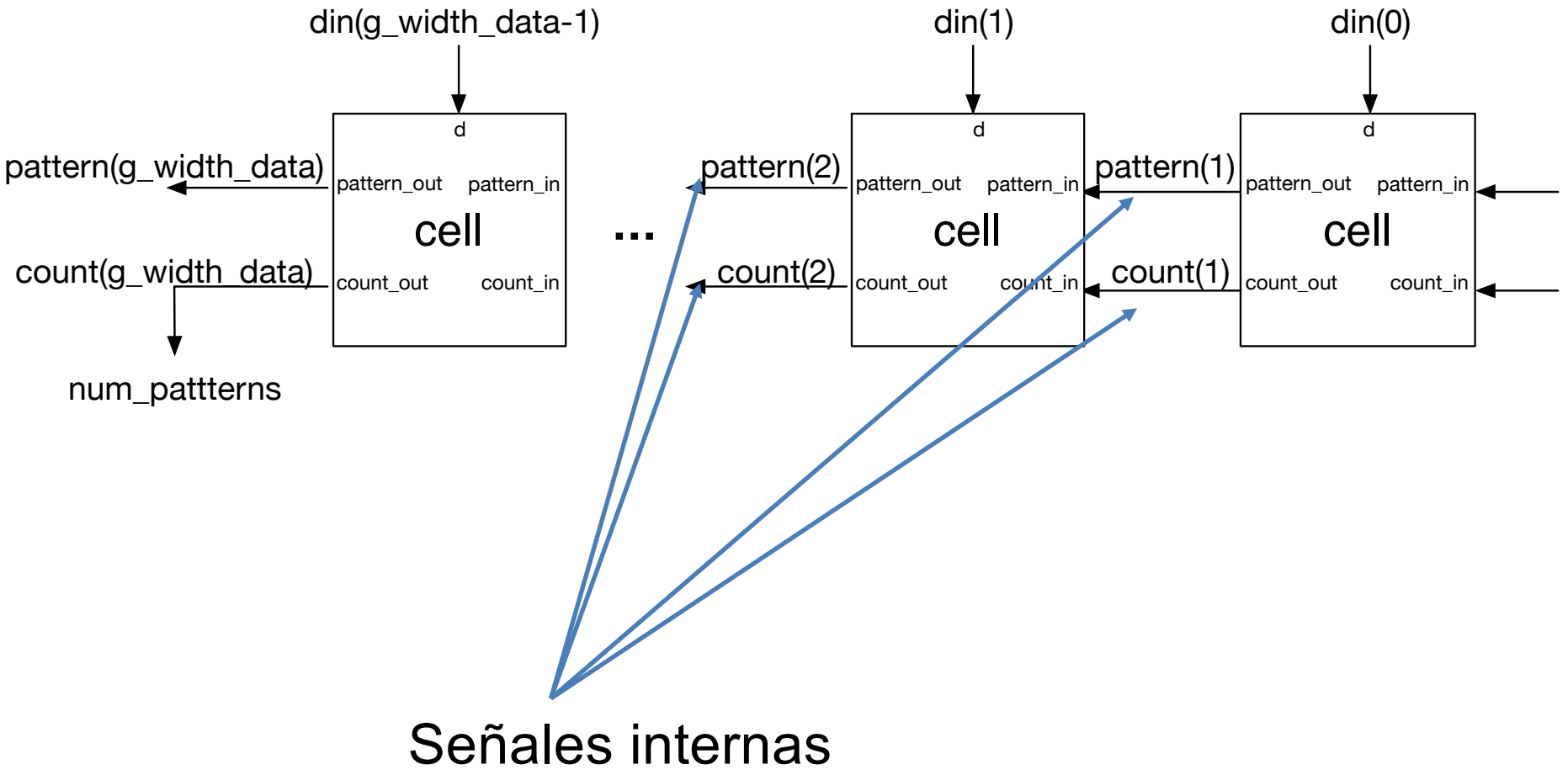
```
entity iterative_1d is
    generic (g_width_data  : natural := 32;
             g_width_count : natural := 5);
    port ( din          : in  std_logic_vector(g_width_data-1 downto 0);
           num_patterns : out std_logic_vector(g_width_count-1 downto 0);
end iterative_1d;
```

Estructura

```
entity iterative_1d is
    generic (g_width_data  : natural := 32;
             g_width_count : natural := 5);
    port ( din          : in  std_logic_vector(g_width_data-1 downto 0);
           num_patterns : out std_logic_vector(g_width_count-1 downto 0);
end iterative_1d;
```



Estructura



toc

Descripción de alto nivel

- Puerto *count_out*. Número de veces que se ha detectado el patrón
- Puerto pattern_out. Reconocimiento del patrón.
  - Cinco valores distintos: no_pattern, first_one, second_one, second_zero y pattern_rec.

	d	
Pattern_in	0	1
no_pattern	no_pattern	first_one
first_one	second_zero	second_one
second_zero	no_pattern	pattern_rec
second_one	second_zero	pattern_rec
pattern_rec	no_pattern	first_one

Descripción VHDL de la celda

- Definir los tipos de las señales internas

```
package definitions is
  type t_pattern is (no_pattern, first_one, second_one, second_zero, pattern_rec);
end package definitions;
```

- Definir las señales externas

```
use work.definitions.all;

entity cell is
  generic ( g_width : natural := 32);
  port (d           : in std_logic;
        pattern_in  : in t_pattern;
        count_in    : in unsigned(g_width-1 downto 0);
        pattern_out : out t_pattern;
        count_out   : out unsigned(g_width-1 downto 0));
end cell;
```

Necesario para incluir este paquete en *cell.vhd*
work es el “espacio de trabajo” por defecto

Descripción VHDL de la celda

- Usaremos dos procesos y una sentencia asignación.

```
p_pattern : process (pattern_in, d) is
  begin
    case pattern_in is
      <completar>
        pattern_cell <= ...
      <\completar>
    end case;
end process p_pattern;
```

Este proceso determina la salida pattern_cell

Este proceso determina la salida count_out

```
p_count: process (count_in, pattern_cell) is
  begin
    <completar>
     count_out <= ...
    <\completar>
end process p_count;
```

```
pattern_out <= pattern_cell;
```

Descripción VHDL de la red

- Nuevos tipos en parte declarativa arquitectura

```
architecture struct of iterative_1d is
  type t_pattern_vector is array (g_width_data downto 0) of t_pattern;
  type t_count_vector is array (g_width_data downto 0) of unsigned(g_width_count-1 downto 0);
  <...>
end package definitions;
```

- Definir las señales de interconexión

```
signal count   : t_count_vector;
signal pattern : t_pattern_vector;
```

- Usar la cláusula generate

```
cell_generation : for idx in 0 to g_width_data-1 generate
  i_cell : cell
    generic map ( g_width  => g_width_count)
    port map ( din         => din(idx),
               pattern_in  => pattern(idx),
               count_in    => count(idx),
               pattern_out => pattern(idx+1),
               count_out   => count(idx+1));
end generate cell_generation;
```

Especificaciones

- La unidad multifunción es un circuito combinacional. No posee ni señal de reloj ni de reset.
- El circuito posee dos entradas de operandos en complemento a 2, op1 y op2, de ancho parametrizable.
- El circuito posee un puerto de control, `sel`, de ancho 3 bits que selecciona el tipo de operación a realizar sobre los dos operandos. Los valores son:
  - 000: suma
  - 001: resta
  - 100: min
  - 101: max
  - 111: abs del dato op1

Especificaciones

- En el caso de que el puerto `sel` tome un valor distinto de los enumerados anteriormente, la unidad realizará la operación de suma.
- El circuito posee una salida en complemento a 2, `res`, de ancho parametrizable.
- La entidad `multifunction` viene definida por el siguiente código VHDL:

```
entity multifunction is
    generic (g_width  : natural := 32);
    port ( op1 : in  signed(g_width-1 downto 0);
           op2 : in  signed(g_width-1 downto 0);
           sel : in  std_logic_vector(2 downto 0);
           res : out signed(g_width-1 downto 0));
end multifunction;
```

Descripción VHDL

- Se definen constantes con los valores de sel para cada operación.

```
constant c_add : std_logic_vector(2 downto 0) := "000";
constant c_sub : std_logic_vector(2 downto 0) := "001";
constant c_min : std_logic_vector(2 downto 0) := "100";
constant c_max : std_logic_vector(2 downto 0) := "101";
constant c_abs : std_logic_vector(2 downto 0) := "110";
```

- Usar la sentencia case para definir la funcionalidad

```
p_fu : process (sel, op1, op2) is
  begin
    case sel is
      <completar>
    end case;
end process p_pattern;
```