



Práctica 3

- Redes iterativas y unidad multifunción

Última actualización: 16/11/2017

Objetivos



- Aprender a utilizar la sentencia `generate`.
- Aprender a utilizar genéricos para parametrizar el tamaño de los diseños 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



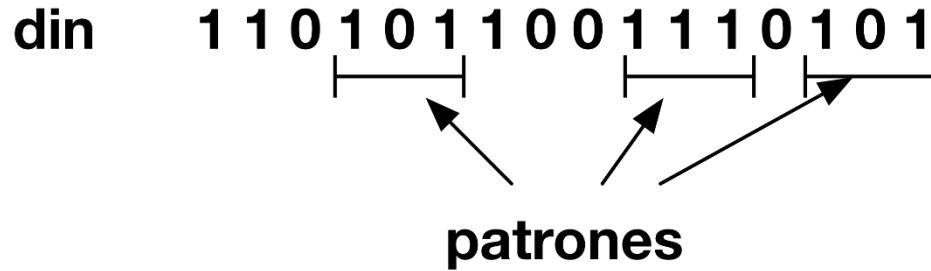
- Práctica 3.a: Red iterativa.
 - Reconocedor del patrón “1X1”.
- Práctica 3.b: Unidad multifunción.
 - Sumador, restador, MAX, MIN.
- Práctica 3.c: Apartado AVANZADO que haréis en el laboratorio.



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

Especificación

- 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.



num_patterns = 3

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



Especificación

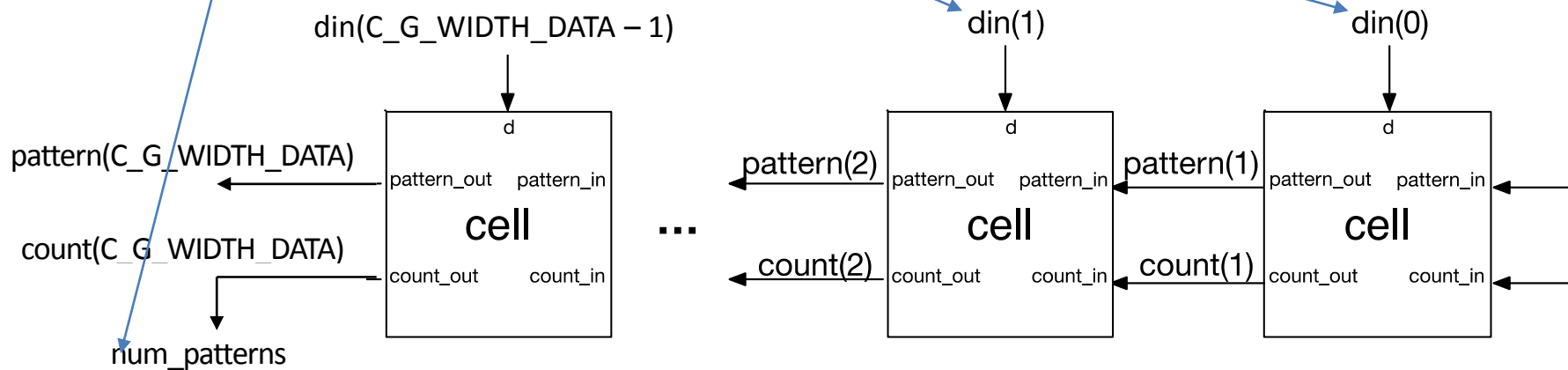
- El circuito tiene un puerto de salida, num_patterns, de un ancho parametrizable. El puerto num_patterns devuelve, en formato binario puro, el número de veces que aparece el patrón en din.
- La definición de la entity sería la siguiente:

Definiremos más adelante las 2 constantes
C_G_WIDTH_DATA y C_G_WIDTH_COUNT

```
entity iterative_1D is
    port (din: in std_logic_vector (C_G_WIDTH_DATA-1 downto 0);
          num_patterns: out std_logic_vector (C_G_WIDTH_COUNT-1 downto 0));
end iterative_1D;
```

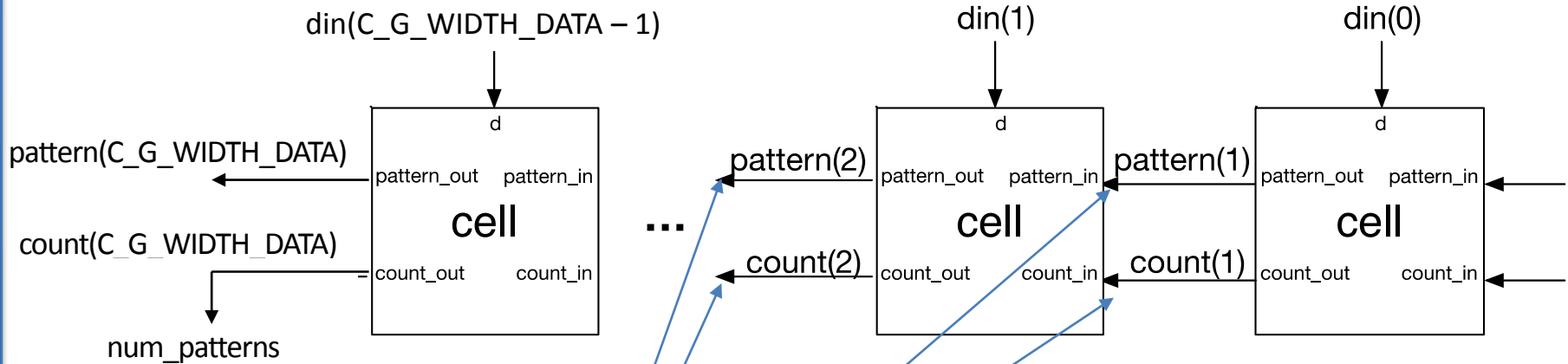
Estructura

```
entity iterative_1D is
  port (din: in std_logic_vector (C_G_WIDTH_DATA-1 downto 0);
        num_patterns: out std_logic_vector (C_G_WIDTH_COUNT-1 downto 0);
end iterative_1D;
```





Estructura



Señales internas



Descripción a alto nivel de la celda básica

- Entrada `din`
 - Cada celda recibe una componente del vector de entrada (1 bit).
- Señales `pattern`
 - Sirven para ir reconociendo el patrón. Indican el valor de los 3 bits situados a la derecha de la celda.
 - Pueden tomar 5 valores distintos: `no_pattern`, `first_one`, `one_zero`, `one_one` y `pattern_rec`.
 - Los posibles valores de `pattern_out`, en función de la entrada (1 bit), y `pattern_in` (`g_width_data` bits) son los siguientes:
 - `no_pattern`: todavía no se ha reconocido el primer 1 del patrón.
 - `first_one`: se ha reconocido el primer 1 del patrón.
 - `one_zero`: se ha reconocido la subsecuencia “01”.
 - `one_one`: se ha reconocido la subsecuencia “11”.
 - `pattern_rec`: se ha reconocido el patrón “101” o “111”.
- Señales `count`
 - Cuentan el número de veces que se ha detectado el patrón hasta la celda actual.

Descripción a alto nivel de la celda básica



- Señal pattern
 - Las transiciones entre los valores que pueden tomar las señales intermedias pattern son las siguientes:

	Entrada	
Estado	0	1
no_pattern	no_pattern	first_one
first_one	one_zero	one_one
one_zero	no_pattern	pattern_rec
one_one	one_zero	pattern_rec
pattern_rec	no_pattern	first_one

- OJO: No son “estados” de una máquina de estados (FSM) secuencial, sino “valores intermedios” que van generando las señales pattern de la red iterativa a medida que la señal de entrada din va siendo analizada por las celdas de la red.



Descripción VHDL de la celda básica

- Los tipos de las señales internas se definirán en un paquete. Este paquete se utilizará tanto en la celda básica como en la red completa:

```
package definitions is
```

```
    constant C_G_WIDTH_DATA: integer := 32;
```

```
    constant C_G_WIDTH_COUNT: integer := 5;
```

```
    type t_pattern is (no_pattern, first_one, one_zero, one_one,  
                      pattern_rec);
```

```
    subtype t_count is unsigned (C_G_WIDTH_COUNT-1 downto 0);
```

```
end package definitions;
```

Nuevo tipo totalmente personalizado

Subtipo t_count. Los valores de las señales de tipo t_count lo son también de tipo unsigned con un rango de mayor número de elementos



Descripción VHDL de la celda básica

- La celda básica (`cell.vhd`) tendrá la siguientes características:

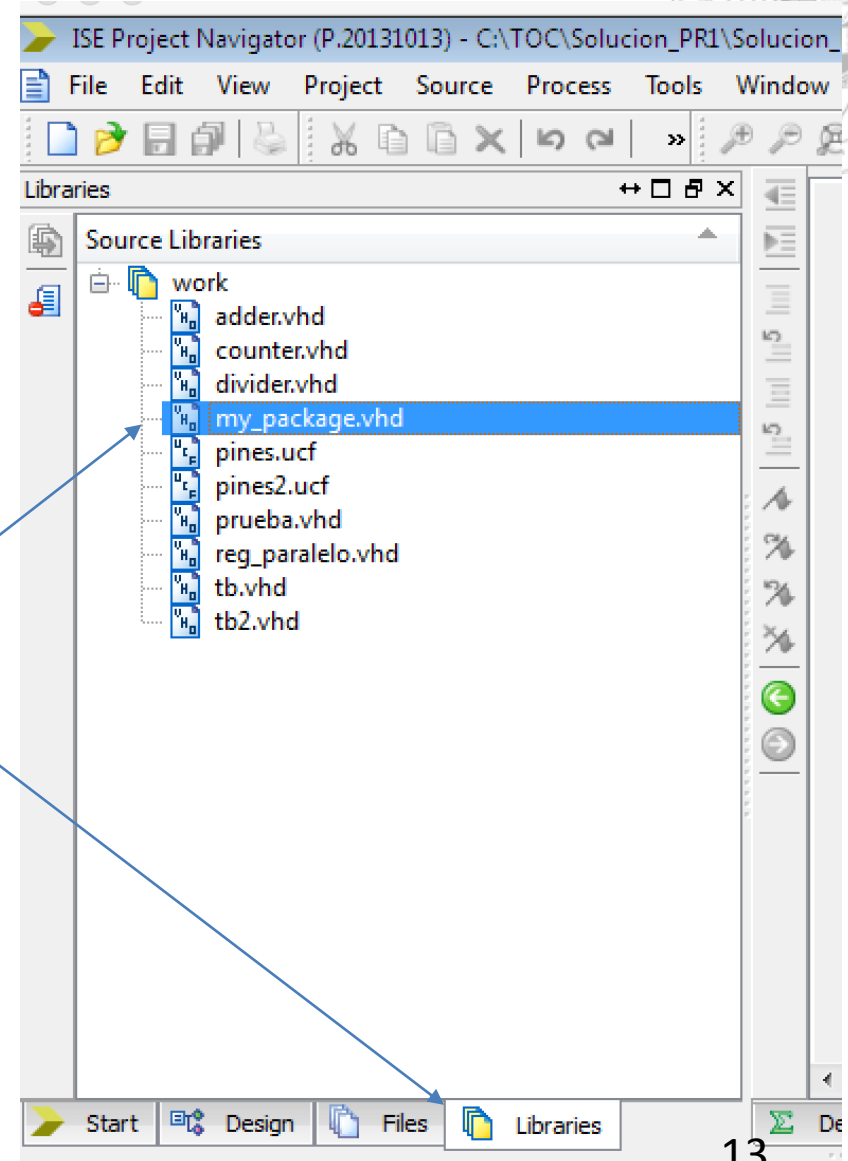
Necesario para incluir este paquete en *cell.vhd*
work es el “espacio de trabajo” por defecto

```
use work.definitions.all;
```

```
entity cell is
    port (d           : in std_logic;
          pattern_in  : in t_pattern;
          pattern_out  : out t_pattern;
          count_in     : in t_count;
          count_out    : out t_count;
end cell;
```

Descripción VHDL de la celda básica

- **ATENCIÓN:** El paquete definitions, una vez añadido al proyecto e invocado desde cualquier fichero fuente, desaparecerá de la lista de fuentes de ISE.
- Para encontrarlo, ir a la pestaña “Libraries”, situada en la sub-ventana izquierda de la interfaz de ISE.





Descripción VHDL de la celda básica

- Dentro de la celda básica, utilizaremos dos procesos para describir su comportamiento:

```
p_pattern_out: process (pattern_in, d) is
begin
    case pattern_in is
        --completar
    end case;
end process;
pattern_out <= pattern_i;
```

Este proceso asigna un valor a la señal intermedia pattern_i.
¡También hay que definirla en el código de la celda!

```
p_count_out: process (count_in, pattern_i) is
begin
    if pattern_i = pattern_rec then
        count_out <= count_in + 1;
    else
        count_out <= count_in;
    end if;
end process;
```

Este proceso determina la salida count_out



Descripción VHDL de la red iterativa

- Para implementar la red completa, tendremos que ampliar el paquete `definitions` con dos nuevos tipos:

```
package definitions is
  constant C_G_WIDTH_DATA: integer := 32;
  constant C_G_WIDTH_COUNT: integer := 5;
  type t_pattern is (no_pattern, first_one, one_zero, one_one,
                    pattern_rec);
  subtype t_count is unsigned (C_G_WIDTH_COUNT - 1 downto 0);
  type t_pattern_vector is array (C_G_WIDTH_DATA downto 0) of t_pattern;
  type t_count_vector is array (C_G_WIDTH_DATA downto 0) of t_count;
end package definitions;
```

- Dicha red tendrá dos señales intermedias de interconexión entre celdas básicas:

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



Descripción VHDL de la red iterativa

- Dentro de su architecture, utilizaremos la sentencia generate para instanciar g_width_data celdas básicas:

```
iterative_network: for i in 0 to C_G_WIDTH_DATA - 1 generate
    i_cell: cell
        -- completar (mirar transparencias 24-26 del Tema 3)
end generate iterative_network;
```




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



Práctica 3.b: Especificación

- **Implementaremos una unidad multifunción, la cual será un circuito combinacional.** Es decir, no posee ni señal de reloj ni de reset.
- El circuito tiene como entradas dos operandos en complemento a 2 (op1 y op2), de ancho parametrizable.
- El circuito también tiene como entrada un puerto de control (sel), de ancho 3 bits que selecciona el tipo de operación a realizar sobre los dos op1 y op2. Los valores son:
 - 000: Suma: $op1 + op2$
 - 001: Resta: $op1 - op2$
 - 100: Mínimo entre op1 y op2
 - 101: Máximo entre op1 y op2

Unidad multifunción: especificación



- La salida del circuito es una señal en binario puro (unsigned) de ancho parametrizable (res).
- La entidad (a quien llamaremos UM) viene definida por el siguiente código VHDL:

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

Unidad multifunción: implementación



- Utilizaremos la sentencia CASE para definir la funcionalidad de la unidad multifunción:

```
process (sel, op1, op2) is
begin
    case sel is
        -- completar
    end case;
end process;
```



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

Xilinx ISE: Estadísticas

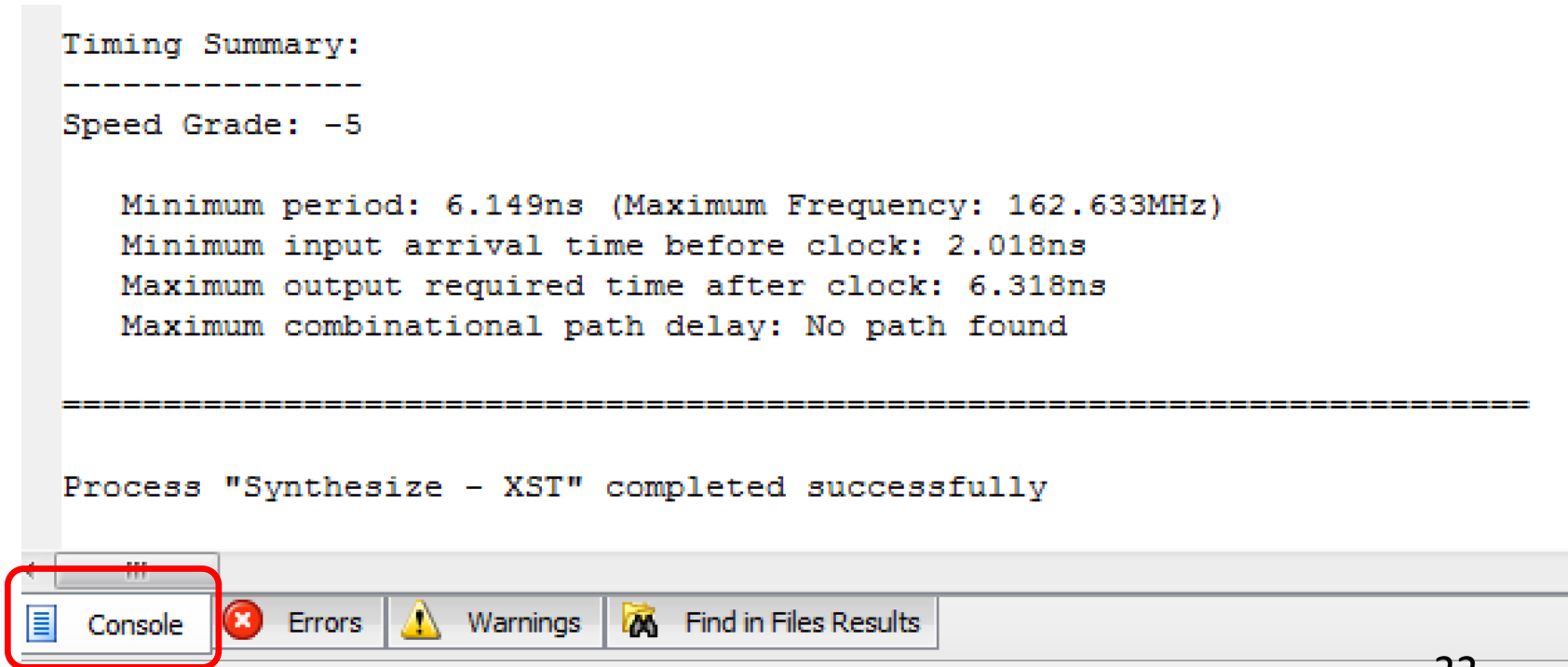
- Se pueden obtener estadísticas interesantes acerca del diseño sintetizado estudiando su “synthesis report”, visible después de haber realizado la síntesis (“Synthesize - XST”), y haciendo click en la pestaña “Console”, situada en la parte inferior de la ventana de ISE:

```
Timing Summary:
-----
Speed Grade: -5

Minimum period: 6.149ns (Maximum Frequency: 162.633MHz)
Minimum input arrival time before clock: 2.018ns
Maximum output required time after clock: 6.318ns
Maximum combinational path delay: No path found

-----

Process "Synthesize - XST" completed successfully
```





Xilinx ISE: Estadísticas

- A lo largo del report, se pueden ir encontrando cómo Xilinx ISE infiere estructuras hardware en nuestro diseño, y cómo las traduce a elementos que existen en la FPGA (sumadores/restadores, contadores, FFs...):

```
Synthesizing Unit <adder>.
  Related source file is "C:/TOC/Solucion_PR1/adder.vhd".
  Found 4-bit adder for signal <C>.
  Summary:
    inferred    1 Adder/Subtractor(s) .
Unit <adder> synthesized.

Synthesizing Unit <reg_paralelo>.
  Related source file is "C:/TOC/Solucion_PR1/reg_paralelo.vhd".
  Found 4-bit register for signal <S>.
  Summary:
    inferred    4 D-type flip-flop(s) .
Unit <reg_paralelo> synthesized.
```



Xilinx ISE: Estadísticas

- Finalmente, también se puede averiguar cuántos recursos de la FPGA se utilizaron para implementar nuestro diseño, programado con VHDL.

Advanced HDL Synthesis Report

Macro Statistics

# Adders/Subtractors	: 1
4-bit adder	: 1
# Counters	: 1
26-bit up counter	: 1
# Registers	: 5
Flip-Flops	: 5

I



Calificación

- Debéis acudir al laboratorio con estos dos apartados estudiados e IMPLEMENTADOS en FPGA desde casa.
 - Si funcionan ambos, +0.2 puntos.
 - Para su simulación, utilizaremos los valores genéricos por defecto dados en las transparencias anteriores.
 - Para su implementación en FPGA, haremos una red iterativa con `C_G_WIDTH_DATA = 8` (utilizar 8 switches para el vector de entrada), `C_G_WIDTH_COUNT = 3` (utilizar 3 LEDs para devolver la salida); y una unidad multifunción con `g_width = 4` (utilizar 8 switches para los 2 operandos, más 3 switches adicionales para op).
- La práctica 3 presenta una parte avanzada (+0.2 puntos).
- La práctica 3 no se recupera.