9 Práctica 3. Redes iterativas y Unidad Multi-Función

9.1 Introducción

Esta práctica tiene los siguientes objetivos:

1. Aprender a parametrizar la descripción de una entidad usando los genéricos de VHDL.
2. Aprender a diseñar sistemas combinacionales iterativos mediante el uso de la cláusula generate de VHDL.
3. Aprender a crear paquetes (package) en VHDL.
4. Estudiar cómo distintas implementaciones de un mismo diseño poseen distintas características físicas.
5. Estudiar la escalabilidad de las distintas implementaciones en función del tamaño de los operandos.
6. Aplicar la segmentación para aumentar el rendimiento en los circuitos digitales.

Vamos a lograr estos objetivos de aprendizaje diseñando dos circuitos: (1) una red iterativa 1-D y (2) una unidad multi-función. En las siguientes secciones presentaremos las especificaciones, el diseño y la descripción en VHDL de cada uno de los circuitos.

9.2 Red iterativa 1-D

9.2.1 Especificaciones

El primero de los circuitos es una red iterativa 1-D con las siguientes especificaciones:

Spec 1. La red iterativa debe calcular el número de veces que aparece el patrón 1x1 sin solapamiento en el vector de entrada del circuito. El símbolo x puede tomar cualquiera de los dos valores binarios. La figura 9.1 presenta un ejemplo de vector de entrada, los patrones que son detectados y el valor de la salida de la red.

Spec 2. La red iterativa es un circuito combinacional. No posee ni señal de reloj ni de reset.

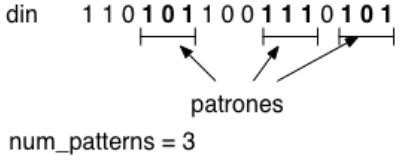

Figura 9.1: Ejemplo de funcionamiento de la red iterativa.

Spec 3. El circuito tiene un puerto de entrada, `din`, de ancho definido mediante el genérico `g_width_data`.

Spec 4. El circuito tiene un puerto de salida, `num_patterns`, de ancho parametrizable mediante el genérico `g_width_count`. El puerto devuelve, en formato complemente a 2, el número de veces que el patrón aparece en el vector de entrada.

Spec 5. La entidad `iterative_1d` viene definida por el siguiente código VHDL:

```
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 unsigned(g_width_count-1 downto 0);
end iterative_1d;
```

9.2.2 Diseño

La figura 9.2 ilustra la estructura típica de una red iterativa. Recuérdese que una red iterativa 1-D está formada por una colección de componentes todos ellos idénticos que únicamente se conectan a los componentes adyacentes. De igual forma, las señales internas que conectan los componentes también son idénticas y su dimensión varia para cada circuito en particular.

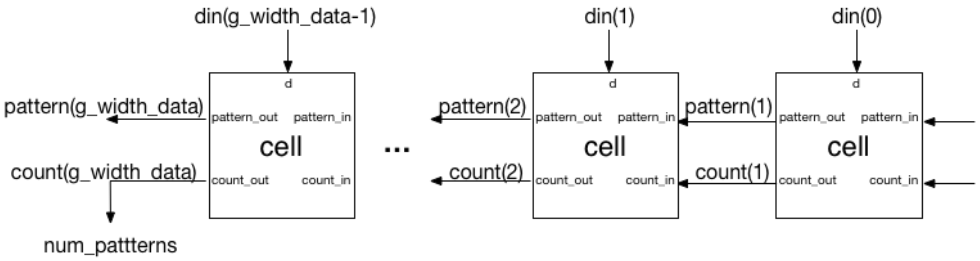

Figura 9.2: Diagrama de bloques de la red iterativa 1 D de la práctica.

El diseño de una red iterativa comienza con la definición de sus celdas y del tipo de información que se intercambia entre ellas. La figura 9.2 ilustra la definición de puertos para cada celda de la red iterativa:

1. Un puerto, `d`, de ancho 1 bit, que recibe una componente del vector de entradas externas de la red iterativa, `din`.
2. Un puerto de entrada, `pattern_in`, de ancho por determinar cuyo valor indica el valor de los tres bits situados a la derecha de la celda. Este valor, junto con el valor de la entrada externa, `d`, sirve para detectar si se ha producido un patrón.
3. Un puerto de entrada, `count_in`, de ancho por determinar, con el valor del número de patrones detectados hasta la celda inmediatamente a la derecha.
4. Un puerto de salida, `pattern_out`, de ancho por determinar cuyo valor indica el valor de los tres bits situados a la derecha de la celda. Incluye el bit de entrada de la celda `d`.
5. Un puerto de salida, `count_in`, de ancho por determinar, con el valor del número de patrones detectados hasta esta celda.

Para el circuito que nos ocupa, cada celda tiene que realizar dos tareas:

1. Detectar si se ha producido un patrón en los bits de entrada situados a su derecha.

2. Si se detecta el patrón entonces sumar 1 a la cuenta de patrones detectados.

Los puertos `pattern_in` y `pattern_out` indican el estado del proceso de reconocimiento del patrón. Los posibles estados en los que se encuentra el proceso de reconocimiento del patrón son:

1. No se ha reconocido el primer 1 de un patrón. A este estado le llamaremos `no_pattern`.
2. Se ha reconocido el primer 1 del patrón. A este estado le llamaremos `first_one`.
3. Se ha reconocido cualquiera de los patrones 01 o 11. A este estado le llamaremos `second_bit`
4. Se ha reconocido el patrón 1x1. A este estado le llamaremos `pattern_rec`.

La tabla 9.1 muestra los valores que tomar el puerto `pattern_out` en función de los valores de los puertos `d` y `pattern_in`.

	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

Tabla 9.1: Descripción de alto nivel de la señal interna `pattern_out`.

Por otro lado, los puertos `count_in` y `count_out` llevan la cuenta del número de patrones detectado a la entrada y la salida de la celda, respectivamente. La ecuación 4.2 describe la funcionalidad del puerto `count_out`.

$$\texttt{count\_out} = \begin{cases} \texttt{count\_in} + 1 & \text{si se ha detectado patrón} \\ \texttt{count\_in} & \text{c.o.c} \end{cases} \tag{9.1}$$

9.2.3 Descripción VHDL

Para definir la funcionalidad de la red iterativa vamos a utilizar una descripción estructural. Primero se va a definir la funcionalidad de la celda básica y a continuación se hará uso de la sentencia generate para definir la funcionalidad de la red iterativa.

Celda básica

La definición de entidad de la celda básica será la siguiente:

```
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;
```

En la definición se utiliza un genérico para definir el ancho de los puertos `count_in` y `count_out`, y el tipo de los puertos `pattern_in` y `pattern_out` es un tipo enumerado que definimos en un archivo de definiciones, definitions.vhd. Este archivo contiene las definiciones de aquellos tipos y constantes que son necesarios en más de un archivo. En nuestro caso contiene la definicion del tipo `t_pattern` que se usa tanto en la celda básica cono en la red iterativa.

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

Para poder utilizar estos nuevos tipos es necesario declarar que se va a utilizar dicho paquete en esta entidad. La declaración debe incluirse en dos sitios: antes de la definicion de la entidad y de la definición de la arquitectura.

```
  <...>
  use work.definitions.all;

  entity cell is
  <...>

  use work.definitions.all;
  architecture struct of iterative_1d is
  <...>
```

Tal y como se indicó en la práctica 2, al utilizar esta estrategia la herramienta de síntesis será quien decida la forma de codificar los valores que pueden tomar las señales que llegan a estos puertos.

A la hora de definir la arquitectura vamos a utilizar dos procesos. El primero para calcular el valor de la señal `pattern_cell` a partir de los valores de `pattern_in` y `d`.

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

La señal `pattern_cell` es una señal interna de la celda que usaremos en dos asignaciones. En primer lugar la usaremos en el segundo de los procesos que calcula el valor `count_out` a partir de `count_in` y de la señal interna `pattern_cell`.

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

La segunda asignación de la señal `pattern_cell` es para asignar su valor al puerto de salida `pattern_out`.

```
pattern_out <= pattern_cell;
```

9.2.4 Red iterativa

La definición de entidad de la red iterativa es la siguiente:

```
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 unsigned(g_width_count-1 downto 0));
end iterative_1D;
```

La especificación del sistema (Spec 2) establece que el tamaño del vector de entrada es parametrizable. Por esa razón se han definido dos generic:

1. g_width_data que establece el tamaño del vector de entrada.

2. g_width_count que establece el tamaño del vector que lleva la cuenta del número de veces que aparece el patrón en el vector de entrada.

Las redes iterativas son el ejemplo arquetípico de sistemas que pueden diseñarse con el bucle generate. Recuérdese que la sentencia generate de VHDL puede generar estructuras parametrizables de tamaño variable. El código siguiente ilustra cómo podemos usar esta sentencia para construir la estructura de la red iterativa.

```
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;
```

En el código anterior se hace uso de las señales count y pattern. Por lo tanto, estas señales deben ser definidas en la parte declarativa de la arquitectura.

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

La definición de las dos señales hace uso de dos nuevos tipos que deben definirse en la sección declarativa de la 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;
```

9.2.5 Análisis de la red iterativa

La red iterativa tiene las siguientes características:

1. Camino critico largo, comienza en el puerto de arrastre de entrada, *c_in* y finaliza en el bit más significativo de la suma.

2. Área reducida.

3. La longitud del camino crítico crece de forma lineal con la longitud de los operandos.

9.2.6 Cuestiones y resultados experimentales

La documentación a presentar en la memoria es:

1. Describir la implementación realizada por XST para la celda básica.

2. Indicar la codificación para cada uno de los valores de la señal interna pattern.

3. Describir el camino crítico e indicar su retardo.

9.3 Unidad Multifunción

9.3.1 Especificaciones

El segundo de los circuitos es una unidad multifuncion con las siguientes especificaciones:

Spec 1. La unidad multifunción es un circuito combinacional. No posee ni señal de reloj ni de reset.

Spec 2. El circuito posee dos entradas de operandos en complemento a 2, op1 y op2, de ancho parametrizable. El nombre del genérico es `g_width`.

Spec 3. El circuito posee una entrada 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 sobre el dato en op1.

En el caso de que el puerto `sel` tome un valor distinto de los enumerados anteriormente, la unidad realizará la opeción de suma.

Spec 4. El circuito posee una salida en complemento a 2, `res`, de ancho parametrizable.

Spec 5. 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;
```

9.3.2 Descripción VHDL

Para definir la funcionalidad de la unidad multfuncion vamos a utilizar cinco constantes para describir la codificacion de cada una de las operaciones. Así:

```
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";
```

La descripción de la funcionalidad se puede hacer mediante una sentencia case.

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

9.3.3 Cuestiones y resultados experimentales

La documentación a presentar en la memoria es:

1. Describir la implementación realizada por XST para la unidad funcional: tipos de operadores (suma, resta, ...) y cómo están conectados.
2. Describir el camino crítico y el valor del retardo para el camino crítico.