Práctica 3

Diseño de alta velocidad. Temporización

Objetivo

El objetivo de esta práctica es el familiarizarse con la problemática del diseño de alta velocidad y utilizar diferentes estrategias de optimización. Como ejemplo, se usará un multiplicador, que es una estructura aritmética básica pero de gran complejidad de implementación.

Descripción del diseño del multiplicador

Utilizamos como referencia un multiplicador basado en sumas y desplazamientos (*shift & add*). Este multiplicador es muy subóptimo en términos de área, velocidad o consumo de potencia, pero por su simplicidad conceptual resulta útil para describir y utilizar diferentes técnicas y opciones de diseño.

Descripción del circuito:

El circuito utiliza la celda de cálculo "*mult_by_1*" (fichero `mult_by_1.vhd`) construido a partir de un sumador y un multiplexor (figura 1). Este circuito ofrece a su salida (SO) la suma del vector de entrada (SI) con el producto del vector A por el bit Bbit. Obviamente, el resultado tiene una anchura de un bit más que la entrada.

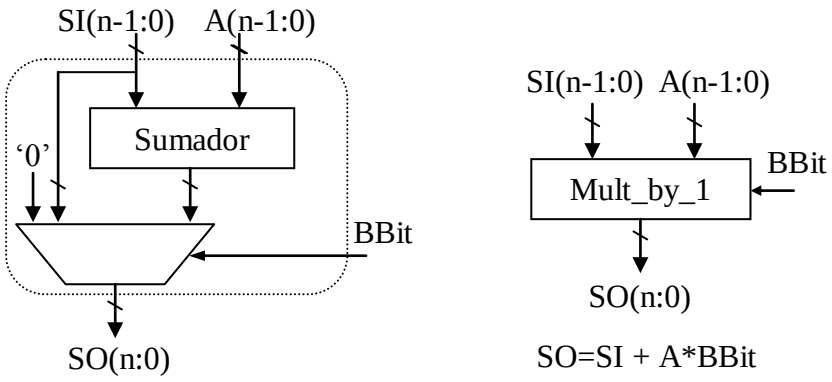

Figura 1. Celda de cálculo *mult_by_1*

El circuito del multiplicador completo se muestra en la figura 2, donde hay que tener en cuenta que la célula básica tiene en esta figura las entradas dispuestas de forma simétrica a como están en la figura 1, *A* está a la izquierda y *SI* a la derecha. El código VHDL utiliza una estructura *generate* para instanciar los diferentes módulos *mult_by_1*.

El algoritmo de *shift & add* es similar al que se utiliza al multiplicar a mano. En esta implementación se suma en cada etapa los bits del resultado anterior. Ver el ejemplo para operandos de 5 bits de la figura 3.

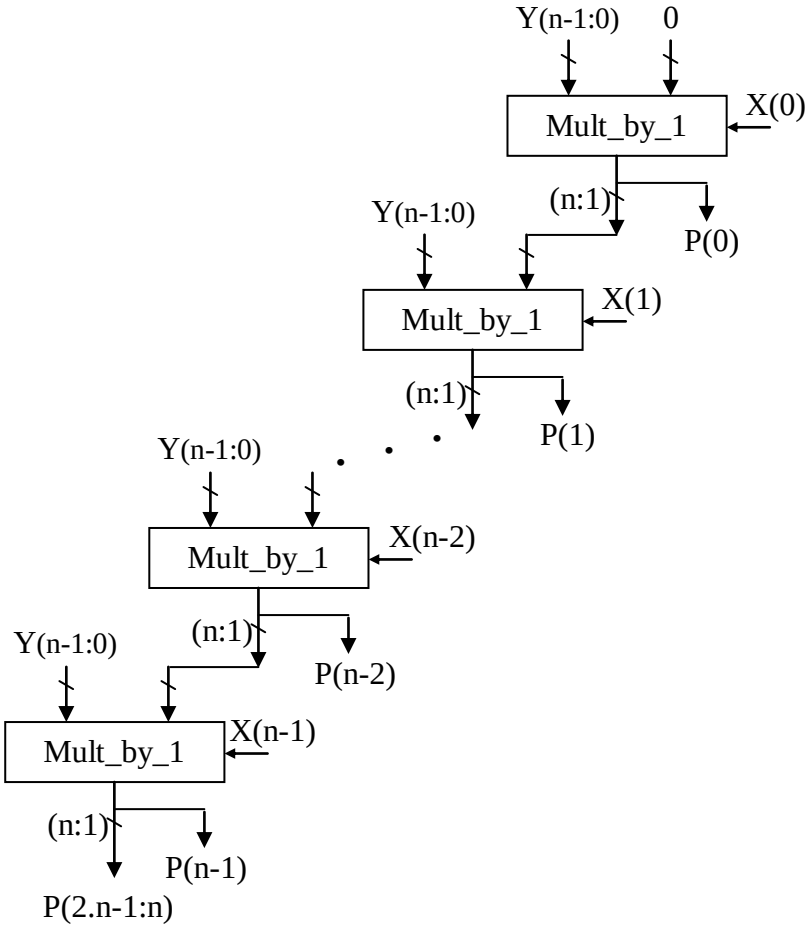

Figura 2. Multiplicador completo. X(n-1:0) e Y(n-1:0) operandos; P(2·n-1:0) resultado.

					1	0	1	0	1
				X	0	1	1	0	1
					1	0	1	0	1
				0	0	0	0	0	
			1	0	1	0	1		
		1	0	1	0	1			
	0	0	0	0	0				
0	1	0	0	0	1	0	0	0	1

Figura 3. Ejemplo Multiplicación algoritmo *shift & add*.

En la figura 4 se muestra un desglose de la operación siguiendo la implementación propuesta. En los recuadros negros están indicados SI y el producto A·BBit en el bloque *mult_by_1* y en fuente azul negrita la correspondiente salida.

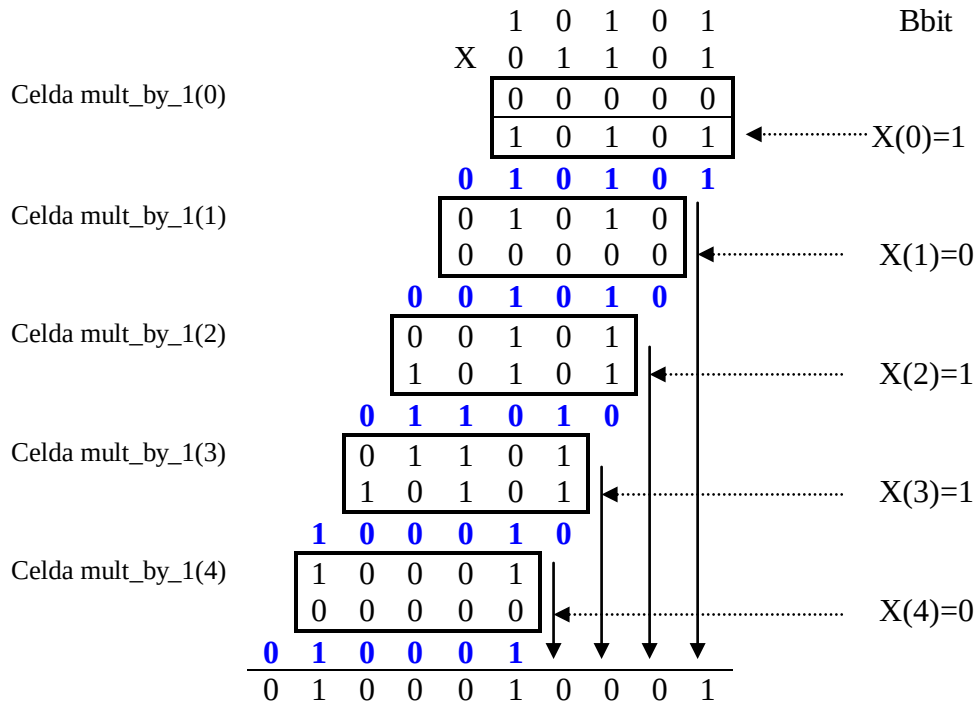

Figura 4. Ejemplo multiplicación algoritmo *shift&add* usando la arquitectura de la figura 2.

Realización de la práctica

La práctica consta de 4 ejercicios en los que se exponen distintas estrategias para mejorar la capacidad del circuito para trabajar a altas frecuencias.

Preparación del diseño

Con objeto de favorecer la limpieza y comprensión del diseño se entrega una base de datos con la estructura siguiente, que habrá de mantenerse:

- `rtl/` - Código fuente VHDL para el diseño
- `sim/` - Directorio de simulación, que contiene los ficheros del banco de pruebas (*test-bench*) y otros ficheros relacionados con la herramienta de simulación.
- `ise/` - Directorio que contendrá el proyecto ISE, que se entrega vacío.

Ejercicio 1: Puesta en Marcha

Se crea el proyecto y se hacen comprobaciones básicas. Las tareas son:

1. Entender el funcionamiento del multiplicador *shift & add*.
2. Modificar el diseño del multiplicador para registrar mediante flip-flops las entradas y salidas de datos, registro que se realizará dentro del mismo fichero `mult_sum.vhd`. En esta práctica, usaremos flip-flops sin reset.
3. Crear un nuevo proyecto "mult_sum" en el directorio `ise/` utilizando un dispositivo Spartan 3 XC3S50 con package PQ208 y speed grade 5. Se trata del dispositivo más pequeño de la familia con una capacidad equivalente de 50k puertas. Añadir los ficheros *.vhd*. Comprobar que el circuito se sintetiza correctamente.
4. Abrir el editor de *constraints*: teniendo seleccionado en el árbol del diseño (solapa *Design*, panel *Hierarchy*) "mult_sum.vhd", ejecutar el proceso *User Contraints* | *Create Timing Constraints*, aceptando a continuación el añadir el fichero UCF al proyecto. Hacer doble clic sobre el reloj *Clk*, en el apartado *Unconstrained Clocks* y,

en la casilla "*Time:*" de "*Specify Time*", definir un requisito para la red `Clk` de 200 ns de periodo (un reloj muy lento con el que seguro que no tendremos problemas). Guardar los cambios y cerrar la solapa *Timing Constraints* de la zona derecha de la herramienta. Comprobar cómo ha quedado "mult_sum.ucf": 1) En la solapa *Design* seleccionar arriba "mult_sum.ucf"; 2) abajo abrir el apartado *User Constraints* y 3) hacer doble clic en *Edit Constraints (Text)*.

5. Volver a seleccionar "mult_sum" en el árbol del diseño y comprobar las opciones del proceso *Implement Design | Place & Route | Generate Post-Place & Route Static Timing*: pulsar el botón derecho del ratón y seleccionar *Process Properties*. Comprobar que *Report Type* está configurado como *Verbose Report*. Esto hará que no sólo se reporten los peores caminos sino también otros que se han acercado más a los requisitos.

6. Implementar el diseño. Observar los distintos reportes de implementación y construir una tabla con los datos que se muestran a continuación, rellenando la primera columna. **Es obligatorio rellenar esta tabla (será pedida por el profesor en clase) y es parte esencial de la práctica comprender su significado**. Para los parámetros indicados en rojo se recomienda seleccionar en la solapa *Design Summary* el apartado *Design Overview | Summary*, consultando los datos del *Device Utilization Summary* que recopila información de los distintos reportes. Los parámetros indicados en azul corresponden al análisis de tiempos. Para obtener estos datos, hacer doble clic sobre el proceso *Analyze Post-Place & Route Static Timing* (bajo *Generate Post-Place & Route Static Timing*). Aparece un panel *Report Navigation* donde, bajo *Timing Constraints*, figuran las diferentes *constraints* especificadas, apareciendo en rojo aquellas que no se hayan podido satisfacer. Pinchando en cada una de las *constraints* aparecerá en la ventana de la derecha (solapa <nombre_diseño>.twx) el reporte correspondiente. Podemos ver el periodo mínimo en el apartado *Timing Summary* y el path más crítico nos aparece el primero de la sección *Setup paths* de la *constraint* de periodo de reloj. Pinchando en los nombres de los *Delay type* ISE mostrará el datasheet del dispositivo, donde normalmente podremos comprobar el significado de ese tiempo. Echar también un vistazo a alguna de las figuras relativas a estos *delay types* que se pueden encontrar bajo el directorio de instalación de Xilinx, concretamente en el directorio: <Xilinx>\ISE_DS\ISE\doc\usenglish\help\delay_types\html\web_ds_sp3.

	Ej. 1	Ej. 3a	Ej. 3b	Ej. 4a	Ej. 4b
Retardo entrada a salida (ciclos de latencia)					
Number of occupied Slices					
Total Number of 4 input LUTs					
Number used as logic					
Number used as route-thru					
Number of Slice Flip-Flops					
Number of IOB Flip-Flops					
Minimum (clock) period					
Levels of logic in data path [1]					
Time used for logic in data path (%) [1]					
Time used for routing in data path (%) [1]					

[1] En la ruta crítica

NOTA: Téngase en cuenta que cuando no se usan flip-flops dentro de las slices o los IOB esta información no aparece en el reporte (no aparece una línea listando 0 FFs). Por otra parte el reporte no diferencia slices con y sin flip-flops. Si se hace un cambio en el diseño que implique solamente añadir flip-flops el número de slices utilizadas puede no cambiar, pero la ocupación de la FPGA habrá aumentado, haciéndose visible en el dato de número de flip-flops.

7. Ejecutar el banco de pruebas entregado para verificar su funcionamiento. La estrategia de comprobar todas las posibles combinaciones de datos de entrada se muestra inviable en cuanto a tiempo de simulación. Se ha simplificado la verificación del diseño probando las combinaciones de datos extremas más algún dato aleatorio, con lo que se obtiene una verificación casi instantánea. Se recomienda invocar directamente el *script* `mult_tb.do` desde la ventana de comandos de ModelSim. Examinar las ondas, entendiendo la temporización entre entradas y salida.

Nota: Se recomienda al terminar cada ejercicio apuntar los datos necesarios para la tabla antes de pasar al siguiente ejercicio, y hacer una copia o archivo comprimido del directorio P3_mult para poder volver a consultar en cualquier momento los resultados de un ejercicio determinado sin tener que volver a hacer cambios al diseño, opciones, etc.

Ejercicio 2: Tiempos en entradas y salidas

(En este ejercicio no es necesario rellenar datos en la tabla).

Una de las formas más eficientes de mejorar las prestaciones de tiempo en entradas y salidas es la de registrar las señales en los bloques de entrada y salida. Además el usar estos flip-flops en los IOBs en vez de otros en los CLBs permite dejar más recursos disponibles para la lógica interna.

1. Asignar los siguientes objetivos para las entradas y salidas del diseño. Esta vez editaremos las siguientes constraints directamente en el fichero UCF, que se aplican a todas las entradas y salidas respectivamente (siempre respecto al pin de la FPGA por donde entra el reloj):

   Tiempo de *setup* de entradas = 4 ns:
   OFFSET = IN 4 ns BEFORE "Clk" RISING;

   Tiempo de establecimiento de datos de salida tras reloj = máximo 7 ns.

   OFFSET = OUT 7 ns AFTER "Clk" RISING;

2. Reimplementar. Analizar el cumplimiento de constraints. Observar en el camino crítico la fórmula del "slack".

   ⇨ ¿Se logran los requerimientos? ¿Cuál es el peor "slack"?

   Mirar uno de los IOB de entrada y otro de salida con el *FPGA Editor* (proceso *Place & Route* | *View/Edit Routed Design (FPGA Editor)*): Seleccionar p.ej. "P<4>", hacer *Zoom Selection* y doble click en el IOB. Capturar la imagen y guardarla. Hacer lo mismo con una entrada. Observar el diseño general de la FPGA.

3. En las propiedades del *Map*per, modificar la denominada *Pack I/O Registers/Latches into IOBs* del valor de *Off* que tiene por defecto al *For Inputs and Outputs*.

4. Reimplementar.

   ⇨ ¿Se logran los requerimientos? ¿Cuál es el peor "slack" ahora?

   Mirar uno de los IOB de entrada y uno de salida con el *FPGA Editor*.

   ⇨ ¿Qué cambios se observan respecto al punto 2?

5. Razonar los datos obtenidos.

   ⇨ Explicar qué ha pasado.

Al principio de esta práctica registramos entradas y salidas del multiplicador,

⇨ ¿Por qué crees que es conveniente registrar las entradas y salidas en este circuito?

Ejercicio 3: Frecuencia de reloj

En este ejercicio se introduce un requisito de velocidad de reloj más exigente en el diseño. Los procesos de *Mapping* y *Place and Route* tratarán de cumplir los requisitos impuestos, tras lo cual realizaremos un análisis del cumplimiento de las especificaciones.

1. Asignar un objetivo de periodo de reloj para el diseño de 60 ns. Para ello, usar de nuevo el *Constraint Editor* o cambiar a mano el fichero UCF.
2. Reimplementar.
   ⇨ ¿Se logran los requerimientos?
   ⇨ Anotar los resultados en la tabla anterior ("3a").
3. Modificar el requisito de tiempo a 55 ns.
   ⇨ Comparar los resultados y anotar los datos en la tabla ("3b").

Ejercicio 4: Pipeline

En este ejercicio se realiza una segmentación del multiplicador aplicando una cadena de flip-flops en mitad de la ruta de datos (ver Figura 5 más abajo). Esto aumenta la latencia (retardo entrada/salida) pero produce mejoras muy significativas en la velocidad de reloj soportada.

1. Segmentar el multiplicador anterior con una etapa de pipeline, tal como se muestra en la Figura 5, e implementarlo. Se recomienda utilizar en el VHDL la misma nomenclatura que en la Figura 5.
   ⇨ Reportar los resultados en la tabla. ¿Qué impacto tiene la técnica en aréa/*timing*?
2. OPCIONAL: Segmentar el multiplicador anterior con dos etapas de pipeline (esto es, partiendo en tres partes en vez de sólo en 2 el camino combinacional). Analizar área y velocidad.
3. OPCIONAL: Alternativamente al punto opcional anterior, realizar la generalización del pipeline de manera que sea configurable la granularidad (mediante el uso de *generic*s VHDL). En este caso reportar la relación máxima frecuencia respecto de la cantidad de etapas de pipeline (tabla y gráfico)

En todos los casos es **imprescindible simular el circuito comprobando la corrección de los resultados**. Como la latencia del multiplicador se ve afectada, este dato debe ser tenido en cuenta por el banco de pruebas, sobre el que habrán de efectuarse las modificaciones oportunas.

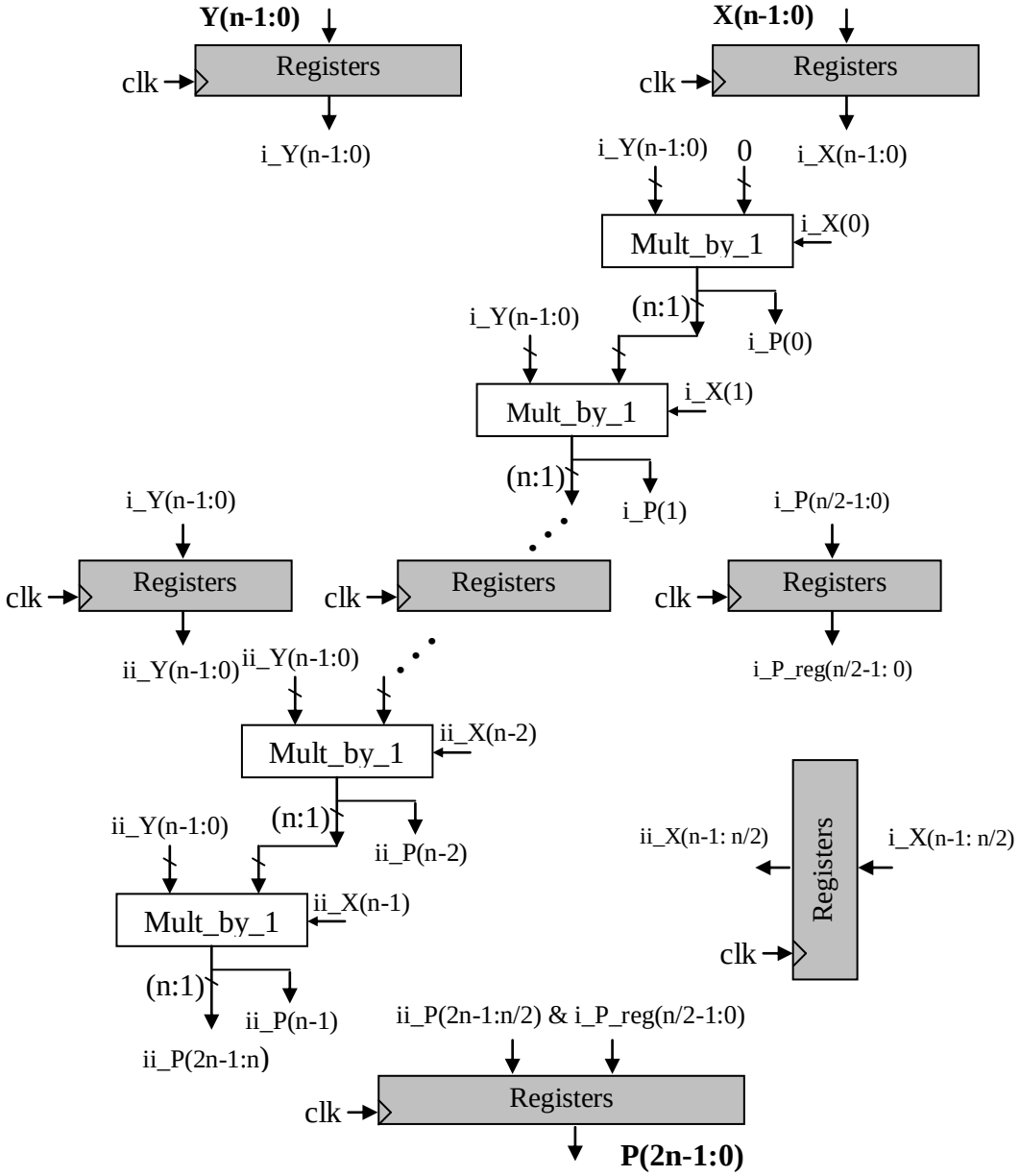

Figura 5. Multiplicador segmentado en una etapa.

Entrega

Antes de la fecha límite especificada en la web del laboratorio, **los alumnos deberán demostrar al profesor el haber ejecutado todos los apartados de esta práctica (**enseñando código creado, simulaciones, resultados obtenidos, etc.**), respondiendo a las preguntas que aparecen en este enunciado y presentando la tabla que se solicita.**

Por otra parte, **deberá entregarse por web un archivo zip con el diseño realizado**, antes de la fecha límite especificada en la web del laboratorio. Entregar el directorio "P3_mult" completo. Basta con entregar el proyecto tal como queda al finalizar la práctica, con los ficheros de rtl/ y sim/ implementando el pipeline (habrá penalización si el testbench no corre satisfactoriamente).

Si se ha hecho algún apartado opcional:

- Basta presentar el proyecto con la implementación más compleja, pero:
- Deberán incluirse una copia de los directorios rtl/ y sim/ por cada tipo de pipeline implementado.
- Deberá incluirse un fichero OPCIONAL.txt/doc, en el directorio P3_mult/ detallando qué mejora opcional se ha implementado.
- Si la mejora (apartado opcional) se entrega tras haber sido mostrada la práctica al profesor, incluir en el fichero OPCIONAL.txt/doc las tablas o gráficos oportunos y entregar el archivo zip con la nomenclatura comentada abajo, sustituyendo "P3" por "P3Mej".

Todos los ficheros del código fuente deberán entregarse con una cabecera adecuada, incluyendo siempre el nombre de los autores (modificar con este fin la cabecera de los ficheros entregados).

Todo el código fuente deberá llevar comentarios apropiados, tanto en contenido como en cantidad.

El archivo zip deberá tener la siguiente nomenclatura para los alumnos de DIE:

```
<{DIE_L | DIE_X | DIE_J}><numero_pareja_2digitos>_P3.zip
```

y la siguiente para los alumnos de DCSE:

```
<{DCSE_L | DCSE_XA | DCSE_XB}><numero_pareja_2digitos>_P3.zip
```

(Ejs.: `DCSE_XB01_P3.zip` para pareja 1 de DCSE-b miércoles, `DIE_J02_P3.zip` para pareja 2 de DIE jueves)

No será necesario presentar ninguna memoria.

Se recuerda que los alumnos deberán comprender el diseño completo y familiarizarse con él.