Sesión 10- Multiplicador Segmentado

Arquitectura de Computadores I (IS19)

Área de Arquitectura y Tecnología de Computadores
Departamento de Ingeniería y Ciencia de los Computadores

Curso 2002-2003

1. Objetivos

La finalidad de esta práctica es el estudio de la segmentación y sus efectos sobre la implementación del sistema. Para ello se procederá a la definición de un multiplicador en el que se variarán las etapas en las que se realice la multiplicación y se observarán los efectos sobre la superficie de silicio ocupada, sobre los tiempos de latencia y sobre la frecuencia máxima de funcionamiento.

2. Descripción del material a emplear

El entorno de trabajo es el Xilinx ISE 4.1 desde donde podemos introducir, simular, compilar e incluso grabar la implementación en el dispositivo elegido. Dentro del entorno podemos lanzar la síntesis y la simulación del programa.

3. Introducción

Como se ha comentado en los objetivos se utilizará un multiplicador segmentado. Para describirlo utilizamos el lenguaje de descripción de hardware VHDL (Very high speed Hardware Description Logic).

3.1 VHDL

- **¿Qué es VHDL?**

La considerable complejidad de los chips que se pueden fabricar, implica unos riesgos y costes de diseño desmesurados e imposibles de asumir por las empresas. Es entonces, cuando diversos grupos de investigadores empiezan a crear y desarrollar los llamados "lenguajes de descripción de hardware" cada uno con sus peculiaridades. Empresas tales como IBM con su IDL, el TI - HDL de Texas Instruments, ZEUS de General Electric, etc., así como los primeros prototipos empleados en las universidades, empezaron a desarrollarse buscando una solución a los problemas que presentaba el diseño de los sistemas complejos.

En 1983, IBM, Intermetrics y Texas Instruments empezaron a trabajar en el desarrollo de un lenguaje de diseño que permitiera la estandarización, facilitando con ello, el mantenimiento de los diseños y la depuración de los algoritmos, para ello el IEEE propuso su estándar en 1984.

Tras varias versiones llevadas a cabo con la colaboración de la industria y de las universidades, que constituyeron a posteriori etapas intermedias en el desarrollo del lenguaje, el IEEE publicó en diciembre de 1987 el estándar IEEE std 1076-1987 que constituyó el punto firme de partida de lo que después de cinco años sería ratificado como VHDL.

Esta doble influencia, tanto de la empresa como de la universidad, hizo que el estándar asumido fuera un compromiso intermedio entre los lenguajes que ya habían desarrollado previamente los fabricantes, de manera que éste quedó como ensamblado y por consiguiente un tanto limitado en su facilidad de utilización haciendo dificultosa su total comprensión. Este hecho se ha visto incluso ahondado en su revisión de 1993.

Pero esta deficiencia se ve altamente recompensada por la disponibilidad pública, y la seguridad que le otorga el verse revisada y sometida a mantenimiento por el IEEE.

La independencia en la metodología de diseño, su capacidad descriptiva en múltiples dominios y niveles de abstracción, su versatilidad para la descripción de sistemas complejos, su posibilidad de reutilización y en definitiva la independencia de que goza con respecto de los fabricantes, han hecho que VHDL se convierta con el paso del tiempo en el *lenguaje de descripción de hardware* por excelencia

- ## Elementos Básicos.

Las estructuras o dispositivos que queramos generar los implementaremos sobre una serie de dispositivos físicos programables. Un dispositivo lógico programable (PLD) es un circuito integrado, formado por una matriz de puertas lógicas y flip-flops, que proporcionan una solución al diseño de forma análoga a las soluciones de suma de productos, productos de sumas y multiplexores. La estructura básica de una PLD permite realizar cualquier tipo de circuito combinacional basándose en una matriz formada por puertas AND, seguida de una matriz de puertas OR. Tres son los tipos más extendidos de PLD's, la PROM, PLA, y la PAL.

PROM (Programmable Read Only Memory).Este tipo de dispositivo se basa en la utilización de una matriz AND fija, seguida de una matriz OR programable. La matriz programable esta formada por líneas distribuidas en filas y columnas en las cuales los puntos de cruce quedaran fijos por unos diodos en serie con unos fusibles que serán los encargados de aislar las uniones donde no se requiera la función lógica.

La fase de programación se realiza haciendo circular una corriente capaz de fundir el fusible en aquellas uniones donde no se desee continuidad. Por otra parte, para cada combinacion de las señales de entrada, el codificador activa una única fila y a su vez activa aquella columna a las que esta todavía unida a travéz del diodo.

PLA (Programmable Logic Array). Parecido en la dispositivo a la PROM, difiere de esta, en que aquí en la PLD , ambas matrices, la de puertas And, así como la de puertas Or es programable, por lo que nos vemos habilitados a incrementar el número de entradas disponibles, sin aumentar el tamaño de la matriz. Esta estructura permite una mejor utilización de los recursos disponibles en el circuito integrado, de tal forma que se genera el mínimo número de términos necesarios para generar una función lógica.

PAL (Programmable array Logic). Una PAL es diferente de una PROM a causa de que tiene una red Y programable y una red O fija. Con un programador PROM podemos obtener los productos fundamentales deseados quemando los eslabones y luego conseguir la suma lógica de dichos productos mediante las conexiones fijas de salida.

VHDL. El lenguaje de descripción hardware VHDL (Very high speed Hardware Description Logic) es un lenguaje orientado a la descripción de hardware pero con muchos elementos heredados de otros lenguajes como C o Pascal. Una vez realizado un programa en VHDL (con extensión VHD) y haberlo compilado con éxito, tendremos un fichero con el mismo nombre y extensión JED, con el cual podremos grabar una PLD (Dispositivo Lógico Programable) con la misma operatividad que el fichero VHD.

Al describir cualquier dispositivo en VHDL (desde una simple puerta and hasta un sistema completo) se deben definir dos elementos principales:

Entidad o **entity** que es la interfaz del dispositivo con el exterior. Tiene por objeto decir que señales son visibles o accesibles desde el exterior, es decir los puertos o ports del

dispositivo. Es describir las patillas del circuito que serán operativas. Su estructura más general es la siguiente, donde las palabras en negrita son propias del lenguaje, y las restantes elegidas por el usuario:

```
entity entidad is
   genericos
   puertos
   declaraciones
end nombre;
```

Pero debemos fijarnos en la estructura más habitual de las entidades que es la siguiente:

```
entity entidad is
   puertos
end entidad;
```

Arquitectura o **arquitecture** que es la funcionalidad que realiza el dispositivo, es decir, qué transformaciones se realizarán sobre los datos que entren por los puertos de entrada para producir la salida. Dentro de este apartado es donde se dota de operatividad al circuito. Su estructura general es la siguiente, y debe estar incluida en el mismo fichero de la entidad a la que hace referencia:

```
arquitecture nombre of nombre_entidad is
   declaraciones
begin
    sentencias
end nombre;
```

Para acabar esta introducción deberemos tener en cuenta una serie de detalles más de éste lenguaje:

- VHDL no distingue las mayúsculas de las minúsculas, por lo que se debe tener cuidado al asignar nombres a las variables, especialmente si estamos acostumbrados a trabajar con C.
- Las variables deben empezar por una letra, no deben contener ni espacios ni símbolos como &, %, $, #, !, etc. Su longitud no está limitada, no pueden acabar con un carácter de subrayado o tener dos subrayados seguidos.
- Para representar un número de una sola cifra, deberemos situarlo entre apóstrofes; así: '1'
- Para representar un número de más de una cifra, lo representaremos así: "10011"
- Es muy probable que en cada práctica encuentres varias entidades y varias arquitecturas. Tomando como ejemplo al multiplexor, sabemos que no todos tienen el mismo número de bits o de canales, por eso cada uno tiene una entidad distinta.

Para aprender más sobre programación VHDL se puede, por ejemplo, visitar el tutorial: http://det.bp.ehu.es/vhdl/pagina/inicio.htm donde se puede encontrar una introducción al lenguaje en castellano.

3.2 Entorno Xilinx

Se puede acceder a una introducción del manejo del entorno Xilinx-ISE en la dirección http://www.ii.uam.es/~cedeps/practicas2003/tutotialEsquematicos.pdf

La herramienta **Xilinx- ISE** (Integrated Software Environment) es una herramienta de diseño de circuitos profesional que permite, entre otras funciones, la realización de esquemáticos, la generación de dispositivos con VHDL y su posterior simulación.

La herramienta consta de dos partes:

- **Project Navigator**: donde se realizará el diseño del circuito, bien mediante un esquemático o utilizando un lenguaje específico de diseño.

- **ModelSim** : donde podrá realizarse la simulación del funcionamiento del circuito y de este modo comprobar si funciona según las especificaciones establecidas.

CREACIÓN DE UN NUEVO PROYECTO

Un proyecto es un conjunto de ficheros de diseño, tales como esquemáticos, líneas de código de programas (si se ha realizado el diseño utilizando un lenguaje de programación específico de diseño de circuitos HDL), listas de conexionado, librerías de componentes, vectores de test para la simulación, etc., seleccionados para un diseño específico.

Nada mas acceder al programa, aparecerá por pantalla una ventana que da acceso al programa de diseño.

Para crear un nuevo proyecto:

• Seleccionar File → New Project

• En la ventana de diálogo de New Project indicar el directorio de ubicación del proyecto en Project Location

• Añadir el nombre ("multiplicador", por ejemplo) en Project Name

• Automáticamente se crea un subdirectorio en la ruta indicada en Project Location con el nombre del proyecto, en este caso "multiplicador", y donde se almacenará todo lo relacionado a este proyecto

• Usar las flechas de desplazamiento para añadir el valor adecuado en los campos correspondientes a Project Device Options, por ejemplo:

  - Device name: Virtex
  - Device: XCV50- 6bg256
  - Design Flow: XST VHDL

Al presionar el botón "OK" ISE crea y muestra el nuevo proyecto en el Project Navigator. Se observarán cambios con respecto al aspecto inicial de la ventana en la parte izquierda, en Sources in Proyect, donde aparece el proyecto creado.

Seguidamente incluiremos los archivos de fuente VHDL mult.vhd y slotmulcc.vhd. Para ello se debe seleccionar Project → Add Source…finalmente, y tras seleccionados ambos archivos, se nos solicitará si se trata de módulos, paquetes o testbench VHDL. Seleccionaremos Módulos VHDL.

Una vez el proyecto ya tiene sus fuentes introducidas podremos simular, sintetizar o implementar el diseño (obsérvese la ventana de procesos posibles sobre el proyecto "Process View").

SIMULACIÓN

Primeros debemos crear un procedimiento de prueba. Esto lo haremos gráficamente con la herramienta adecuada para ello. Otra posibilidad es diseñar un archivo nuevo vhdl en el que se detalle el procedimiento de prueba.

Primero añadiremos un archivo nuevo al proyecto (Project → New Source...), seguidamente aparece una ventana en la que se nos pregunta el tipo y el nombre del nuevo archivo. Seleccionamos "TestBench Waveform" como tipo de archivo y le damos un nombre (por ejemplo tb_mult). Además mantendremos activada la opción "Add to project". Al proceder al paso siguiente se nos hace asociar este archivo con uno de los archivos fuente previos. Elegimos el que contenga la entidad principal (mult.vhd, en nuestro caso). El siguiente paso será elegir los valores de las señales de entrada.

Tras los pasos anteriores se abre HDL Bencher automáticamente. Será necesario actualizar los puertos antes de variar ninguno de los operandos (Options → Update Ports)

Para fijar las señales de entrada podemos asignar un patrón de variación en cada una. Para eso pulsamos en el primer valor de la señal que vamos a cambiar, por ejemplo, 0 de OperandoA y a continuación sobre Pattern. Aparece una nueva ventana donde podemos

asignar los valores inicial y final, el salto entre valores, etc. Así, por ejemplo, podríamos fijar los valores en: Count Up, Do for # cycles: 20, Initial Value=0, Terminal Value=50, Increment By=2 y Count Every=1.

Procederemos de un modo similar con el resto de señales (por ejemplo en OperandoB podríamos poner Count Down, Do for # cycles: 20, Initial Value=1000, Terminal Value=20, Increment By=20 y Count Every=1)

También está la posibilidad de añadir un valor a mano para cada ciclo. Si se trata de un bit, como por ejemplo la señal comienzo en mult.vhd, se puede cambiar el valor actual pulsando con el ratón en el valor que queremos cambiar. En nuestro caso deberemos subir la señal de comienzo a 1 y dejarla ahí para que en todos los ciclos comience una multiplicación.

Finalmente guardamos el testbech (Save Waveform). Si nos pregunta podemos elegir que el test finalice tras un número fijo de asignamientos, por ejemplo 20.

Tras generar el archivo de prueba ya podemos proceder a la simulación. En la ventana del proyecto seleccionamos el archivo de pruebas (tb_mult.tbw en nuestro ejemplo) pulsando sobre el 1 vez (si pulsamos 2 se abrirá el editor que hemos usado para construirlo, pero ahora no es eso lo que buscamos). A continuación vamos al visor de procesos donde podemos ver la representación vhdl de nuestro testbench en "View Behavioral Testbench". También podemos lanzar ModelSim con diferentes opciones.

La diferencia, a la hora de presentar los resultados, entre "Simulate Behavioral VHDL Model" y "Generate Expected Simulation Results" está en la presentación. Mientras que el primero nos presenta los resultados directamente con el visor de ModelSim el segundo devuelve los resultados en el editor HDL Bencher, donde tiene sentido el número de ciclos que habíamos fijado previamente.

SÍNTESIS

Es importante la opción Design Flow del origen del proyecto pues los informes serán diferentes, así como la compilación final. Las opciones que se utilizarán son XST VHDL (de libre utilización) o FPGA Express VHDL (que requiere licencia).

Para ejecutar el proceso deberemos hacer doble clic sobre la opción "Synthesize" del visor de procesos, cuando en la ventana del proyecto esté seleccionado el archivo principal (vhdl).

La descripción del procedimiento así como los resultados podrán consultarse a través de los informes ("View Synthesize Report"). Entre estos resultados podremos encontrar la máxima frecuencia de funcionamiento, los recursos necesarios (LUTs, CLBs, BELs, Slices son elementos básicos de la implementación, cuantificarán la cantidad de recursos necesarios y son dependientes del dispositivo)

IMPLEMENTACIÓN

En este proceso se adecua el diseño al dispositivo físico del que disponemos. Entre los informes disponibles podremos identificar como se ajusta el diseño sobre el dispositivo elegido. Por ejemplo, podemos ver si quedan muchos recursos disponibles porque el dispositivo elegido esté sobredimensionado para las necesidades del diseño o si, por otro lado, es demasiado pequeño.

PROGRAMACIÓN

El último paso para la implementación real del dispositivo es la creación del archivo de programación que será transferido al dispositivo con el programador adecuado. También dispondremos de un informe del proceso.

4. Desarrollo de la sesión de prácticas

Se recomienda crear un archivo de seguimiento con los procedimientos requeridos posteriormente, así como los informes y las conclusiones. Éste será utilizado para la evaluación.

4.1 Análisis del Diseño

Dado el archivo `mult.vhd` identificar los distintos módulos, analizarlos y compararlos con el algoritmo de multiplicación de la clase de teoría. (Reacuérdese que el multiplicador trabaja sobre vectores de NBits=32bits y que OP es el número de bits que se operan en cada ciclo, es decir que NBits módulo OP será el número de etapas de segmentación)

Dibujar un esquema cualitativo del dispositivo en el que se vean las etapas de segmentación para OP=8bits.

4.2 Creación del TestBench

Crear un archivo de pruebas para el multiplicador siguiendo las instrucciones de la sección anterior. Éste será utilizado para la simulación del apartado siguiente.

4.3 Simulación

Proceder a realizar la simulación del multiplicador variando el número de bits operados cada vez, y por tanto la segmentación, para los valores OP=32, 16, 8 y 4. Verificad el correcto funcionamiento del sistema y estudiad la latencia de aparición de los resultados y su relación con las etapas de segmentación.

Realizad una tabla expresando la latencia en función de los bits operados.

4.4 Síntesis e Implementación

Realizad los procesos de síntesis e implementación para los 4 multiplicadores (OP=32, 16, 8 y 4). De los informes obtenidos extraed la información referente a frecuencia de funcionamiento en cada caso así como la de recursos utilizados (pines y recursos internos principalmente)

Realizad una tabla resumen de los recursos utilizados para cada multiplicador.

4.5 Conclusiones.

Resumir los efectos de la segmentación sobre la latencia, la frecuencia máxima y los recursos utilizados. Responder a las cuestiones siguientes:

- ¿Aumenta o disminuye la frecuencia máxima posible al aumentar la segmentación?
- ¿Aumenta o disminuye la latencia al aumentar la segmentación?
- ¿Aumentan o disminuyen los recursos utilizados al aumentar la segmentación?
- ¿Cuáles son las ventajas de la segmentación en este sistema?
- ¿Cuál es el dispositivo más pequeño, de la serie virtex, en la que se puede implementar el multiplicador con OP=8?

5. Anexo: Listado de `mult.vhd` y `slotmultcc.vhd`

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mult is
  generic ( NBITS:NATURAL:=32; OP:NATURAL:=8);
  Port ( OperandoA : in std_logic_vector(NBITS-1 downto 0);
         OperandoB : in std_logic_vector(NBITS-1 downto 0);
```

```
          Resultado : out  std_logic_vector(2*NBITS-1 downto 0);
          comienzo,clk: in std_logic);
end mult;

architecture Behavioral of mult is
        type matrizlineas is array ( NBITS downto 0) of std_logic_vector (2*NBITS-1 downto 0);
        type matrizhalf is array ( NBITS downto 0) of std_logic_vector (NBITS-1 downto 0);
        signal fblineas,lineas:matrizlineas;
        signal multiplicando:matrizhalf;

        COMPONENT slotmultcc
           generic (N:natural:=32);
           PORT(
                anterior : IN std_logic_vector(2*N-1 downto 0);
                multiplicando : IN std_logic_vector(N-1 downto 0);
                posterior : OUT std_logic_vector(2*N-1 downto 0)
                );
        END COMPONENT;

begin
        regi: process (OperandoA,OperandoB, comienzo,clk)
        begin
                if (clk'event and clk='1') then
                  if (comienzo='1') then
                        multiplicando (0)<= OperandoA ;
                        --multiplicando(NBITS-1 downto 1) <= multiplicando (NBITS-2 downto 0);
                        lineas(0)<=  (others=>'0');
                        lineas(0)(NBITS-1 downto 0) <= OperandoB;
                  end if;
                end if;
        end process regi;

        todo:for i in 0 to NBITS-1 generate
                Inst_slotmultcc: slotmultcc generic map (NBITS) PORT MAP(
                        anterior => lineas(i),
                        posterior => fblineas(i+1),
                        multiplicando => multiplicando(i)
                );
        end generate;
        allregs:for i in 1 TO NBITS generate
                siregs: if ((i mod OP)=0) generate
                        regs: process (fblineas(i) ,clk)
                        begin
                                if (clk'event and clk='1') then lineas (i)<= fblineas(i);
                                        multiplicando(i) <= multiplicando(i-1);
                                end if;
                        end process regs;
                end generate;

                noregs: if ((i mod OP)/=0) generate
                                lineas(i) <= fblineas(i);
                                multiplicando(i)<=multiplicando(i-1);
                end generate;
        end generate ;
        Resultado <=lineas(NBITS);
end Behavioral;

***** slotmultcc.vhd *****
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity slotmultcc is
   generic(N:natural:=32);
   Port ( anterior : in std_logic_vector(2*N-1 downto 0);
          posterior : out std_logic_vector(2*N-1 downto 0);
```

```
            multiplicando : in std_logic_vector(N-1 downto 0));
end slotmultcc;

architecture Behavioral of slotmultcc is
        signal suma:std_logic_vector(N downto 0);-- n+1
begin
        suma <= ('0' & multiplicando)+('0'&anterior(2*N-1 downto N));
        posterior <=  suma & anterior(N-1 downto 1) when (anterior(0)='1')
              else '0' & anterior (2*N-1 downto 1);
end Behavioral;
```