METODOLOGÍA DE DISEÑO E IMPLEMENTACIÓN DE SISTEMAS EMBEBIDOS

—Utilizando Herramientas Abiertas—

CARLOS IVÁN CAMARGO BAREÑO

ÍNDICE GENERAL

[a4paper]article [latin1]inputenc listings [T1]fontenc [spanish,es-noshorthands]babel
amsmath amssymb,amsfonts,textcomp color [top=2.501cm,bottom=2.501cm,left=3cm,right=3cm,nohead,nofoot]ge
array supertabular hhline hyperref [pdftex]graphicx

0.1. Máquinas de Estados Algorítmicas (ASM)

La lógica de transferencia de registros es una técnica ampliamente utilizada para la descripción a alto nivel del comportamiento de circuitos secuenciales; los cuales pueden ser vistos como un grupo de registros y operaciones aritméticas y/o lógicas que transfieren datos de un registro a otro. Todo sistema secuencial puede ser visto como la unión de un camino de datos (registros y operaciones) y una unidad de control que determina cuando realizar las operaciones y las transferencias (ver Figura **??**).

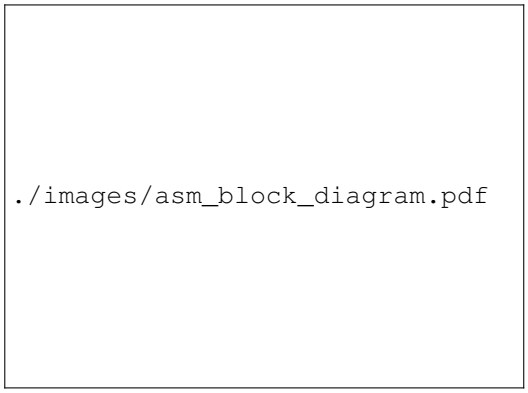

Figura 0.1: Estructura de una máquina de estados algorítmica.

En este capítulo la unión de una unidad de control y un camino de datos recibirá el nombre de máquina de estados algorimica (ASM), para diferenciarla de la máquina de estados finitos y para indicar la posibilidad de implementación de cualquier tipo de algorítmo. Los pasos que se realizan para el diseño e implementación de una máquina de estados algorítmica son los siguientes:

1. Elaboración de un diagrama de flujo que describa la funcionalidad deseada.

2. Identificación de los componentes del camino de datos.

3. Identificación de las señales necesarias para controlar el camino de datos e interconexión.

4. Especificación de la unidad de control utilizando diagramas de estado.

5. Implementación de los componentes del camino de datos y de la unidad de control utilizando lenguajes de descripción de hardware.

6. Simulación y pruebas

Ejemplo de ASM: Multiplicación de números binarios

El algoritmo de multiplicación que se implementará se basa en productos parciales; el primer producto parcial siempre es cero (ver Figura **??**, a continuación se realiza la multiplicación iniciando con el bit menos significativo del multiplicador, el resultado de la multiplicación se suma al primer producto parcial y se obtiene el segundo producto parcial; si el bit del multiplicador es '0' no se afecta el contenido de PP, por lo que no se realiza la suma. A continuación se realiza la multiplicación del siguiente bit (a la izquierda del LSB) y el resultado se suma al producto parcial pero corrido un bit a la izquierda, esto para indicar que la potencia del siguiente bit tiene un grado más; este corrimiento se debe realizar ya que si un número binario se multiplica por 2 el resultado es el mismo número corrido a la izquierda, por ejemplo:

15 (1111) X 2 = 11110 = (30); 15 (1111) X 4 = 111100 = (60)

Este proceso continúa hasta completar los bits del multiplicador y el último producto parcial es el resultado.

Figura 0.2: Multiplicación de numeros binarios usando productos parciales.

Diagrama de Flujo

En la Figura **??** se muestra un diagrama de flujo para la implementación de este algoritmo. El primer paso para realizar la multiplicación es hacer el producto parcial (PP) sea igual a cero, a continuación se realiza una verificación del bit menos significativo del multiplicador, esto se hace para sumar únicamente los resultados que no son cero. En este caso se utiliza un corrimiento a la izquierda para obtener el siguiente bit del multiplicador, si por ejemplo

al número *101**0*** se le realiza un corrimiento a la derecha se obtiene el número *010**1***, con lo que el bit menos significativo corresponde al segundo bit de *1010*, si se realiza otro corrimiento a la derecha se obtiene *001**0*** y de nuevo el bit menos significativo corresponde al tercer bit de *1010*, al realizar de nuevo un corrimiento se obtiene *000**1***, con lo que tendríamos todas las cifras del multiplicador de forma consecutiva en el mit menos significativo. Cuando se realiza un nuevo corrimiento el resultado es *0000* lo que indica que el producto parcial no puede cambiar y podemos terminar el algortimo. Este método para finalizar el algoritmo produce que el número de iteraciones depende del valor del multiplicador; otra forma de terminar el algoritmo sin que dependa del valor del multiplicador se obtiene al contar el número de bits del multiplicador y realizar el corrimiento n veces, donde n es el número de bits del multiplicador.

Para indicar que cada vez que se toma un bit del multiplicador, este tiene una potencia mayor que el bit anterior, debemos multiplicar el resultado por la base, la cual es 2 en este caso; como se mencionó anteriormente, multiplicar por 2 equivale a realizar un corrimiento a la izquierda, por lo que siempre que se tome un nuevo bit del multiplicador debemos correr a la izquierda el multiplicando.

Figura 0.3: Diagrama de flujo para la multiplicación de numeros binarios.

Una vez conocido el funcionamiento del sistema se procede a realizar el diagrama de caja negra de entradas y salidas. En la Figura **??** se muestra el multiplicando y el multiplicador (A y B), señales de m bits cada una, el resultado de la multiplicación PP (Bus de 2m Bits), la señal de reloj (CLOCK). Las señales INIT y DONE se utilizan para iniciar el proceso de multiplicación e indicar que el resultado está disponible respectivamente; es importante que todo sistema digital posea la forma de interactuar con el exterior, ya que sin ello el sistema carecería de utilidad.

Identificación de los componentes del camino de datos

A continuación se identifican los componentes del camino de datos, esto se realiza recorriendo el diagrama de flujo para encontrar las operaciones que se realizan.

En la figura **??** se resaltan las operaciones que se deben realizar para la correcta operación del algorítmo; la primera es una operación de acumulación correspondiente a $PP = PP + A$; la segunda operación que encontramos son los dos corrimientos a la izquierda y derecha del multiplicando (A) y el multiplicador (B) respectivamente, estas operaciones se realizan al mismo tiempo pero en módulos diferentes; el último módulo es un comparador que indica que el multiplicador es igual a cero, indicando que el algoritmo puede finalizar.

Figura 0.4: Diagrama de caja negra para el multiplicador de numeros binarios.

Figura 0.5: Identificación de los componentes del camino de datos para el multiplicador de numeros binarios.

Identificación de las señales de control e interconexión del camino de datos

En la figura **??** se muestra la interconexión de los componentes del camino de datos y las señales que lo controlan.

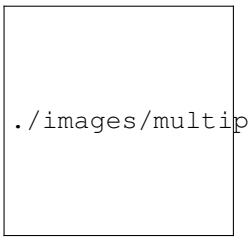

Figura 0.6: Identificación de las señales de control e interconexión del camino de datos.

La primera operación que aparece en el diagrama de flujo es la del acumulador, el cual acumula el valor de la salida del registro de corrimiento que almacena el multiplicando, de aquí obtenemos la conexión entre el registro de corrimiento (LSR) a la izquierda y el acumulador (ACC). La segunda operación que aparece es la de los registros de corrimiento, por lo que los valores del multiplicando y multiplicador deben cargarse para su posterior corrimiento a las unidades de corrimiento a la izquierda y derecha respectivamente. La salida del corrimiento a la derecha del multiplicador es comparada en cada

ciclo para determinar si se llegó al final del algoritmo, por lo que la entrada del comparador es la salida del registro de corrimiento del multiplicador.

Para determinar las señales de control de cada componente del camino de datos, se debe identificar su función y las operaciones que debe realizar; los registros de corrimiento deben permitir la carga de un valor inicial y el corrimiento de las mismas, esto se realiza con las señales *LOAD* y *SHIFT* respectivamente; el acumulador debe tener la posibilidad de inicializarse en cero y una señal para que sume el valor de la entrada al que tiene almacenado, esto se hace con las señales *RESET* y *ADD*; por último el comparador debe proporcionar una señal que indique que el valor de su entrada es igual a cero, *Z* en este caso.

Aunque es posible que la máquina de control maneje todas las señales de control del camino de datos, es mejor aguparlas de acuerdo a su activación; esto es, si una señal se activa al mismo tiempo que otra, se puede utilizar una señal que las controle a ambas. Para esto se utiliza el diagrama de flujo y se observa en que momento se realizan las operaciones: Se observa que se cargan los valores de los registros de corrimiento y se inicializa en cero el acumulador únicamenet al comenzar el algoritmo y durante la ejecución del mismo no se vuelve a relizar esta operación, por este motivo utilizaremos la misma señal (*RESET*) para cargar los registros de desplazamiento e inicializar en cero el acumulador; la señal que controla el momento en que el acumulador se incrementa es única, ya que no se realiza ninguna operación en ese punto del algoritmo y en este caso recibe el nombre de *ADD*; las operaciones de corrimiento se realizan en el mismo lugar, por lo que se puede utilizar una señal común, que en este caso llamaremos *SH*; por último la salida del comparador *Z* y el bit menos significativo de B *LSB* son señales de salida del camino de datos que le darán a la unidad de control la información necesaria para tomar la acción adecuada en los bloques de decisión.

Especificación de la unidad de control utilizando diagramas de estado

Una vez que se conoce el camino de datos, las señales que lo controlan y las señales que ayudarán a la unidad de control a tomar decisiones, se procede con la especificación de la unidad de control, la cual, es una máquina de estados finitos, por lo que la mejor forma de especificarla es utilizando un diagrama de estados; en la figura **??** se muestra la relación entre el diagrama de flujo y el diagrama de estados.

Figura 0.7: Diagrama de estados de la unidad de control del multiplicador binario.

Como se puede observar, existe una relación muy estrecha entre el diagrama de estados y el diagrama de flujo, cada operación del diagrama de flujo

corresponde a un estado de la máquina de control y las transiciones entre ellos son idénticas, observe las líneas del mismo color en la figura **??**

La máquina de estados debe iniciar en START y se queda en este estado siempre que la señal INIT tenga un valor de '0', En el estado INIT la señal RESET = '1', con lo que el valor del acumulador se hace cero y se cargan los registros A y B. Cuando INIT = '1', entramos al estado CHECK el cual evalúa la señal LSB, sí LSB = '0', no se debe realizar la suma de MD, pero si se debe realizar un corrimiento para obtener el siguiente bit del multiplicador y realizar el corrimiento necesario en MD. Si LSB = '1' se debe sumar el valor de las salidas de LSR al valor del acumulador, y después se debe realizar un corrimiento. En el estado ADD se hace la salida ADD = '1' para que el valor a la entrada del acumulador se sume al valor almacenado en él. En el estado SHIFT se realiza el corrimiento de RSR y LSR haciendo el valor de la señal SH = 1.

Para verificar el buen funcionamiento del diagrama de estado debemos realizar una prueba de escritorio: Supongamos que tenemos A = 7 y B = 5 y que INIT = 1:

—m1.855cm—m0.791cm—m1.765cm—m1.038cm—m0.566cm—m0.95699996cm—m1.1719999cm—m1

ESTADO	SH	LSR	RSR	Z	LSB	ADD	DONE	ACC
CHECK	0	00000111	0101	0	1	0	0	00000000
ADD	0	00000111	0101	0	1	1	0	00000111
SHIFT	1	00001110		0010	0	0	0	00000111
CHECK	0	00001110	0010	0	0	0	0	00000111
SHIFT	1	00011100		0001	0	1	0	00000111
CHECK	0	00011100	0001	0	1	0	0	00000111
ADD	0	00011100	0001	0	1	1	0	00100011
SHIFT	1	00111000		0000	1	0	0	00100011
CHECK	0	00111000	0000	1	0	0	0	00100011
END1	0	00111000	0000	1	0	0	1	00100011
START	0	00000111	0101	0	1	0	0	00000000

Como puede observarse el resultado es correcto (35), en la tabla las casillas sombreadas corresponden a las señales que cambian de un estado a otro.

Implementación de los componentes del camino de datos y de la unidad de control

Existe abundante literatura sobre el uso de lenguajes de descripción de hardware para la implementación de sistemas digitales; por este motivo, en este libro no se presentará el código que implementa los diferentes módulos que hacen parte de las máquinas de estado algorítmicas estudiadas.

Es muy importante anotar la importancia de la portabilidad del código, como es bien sabido existen varias empresas que suministran entornos de desarrollo que permiten la entrada de diseño utilizando diferentes medios; las herramientas gráficas utilizados por ellos no son compatibles entre sí, lo que hace imposible el paso de un diseño implementado en una herramienta gráfica a otra de otro fabricante; sin embargo, todas las herramientas aceptan texto con el estándard del lenguaje utilizad; por esto, se recomienda utilizar únicamente entrada de texto en las descripciones.

Simulación

Como se mencionó anteriormente, es posible realizar las simulaciones utilizando las herramientas gráficas de cada uno de los entornos de desarrollo que proporcionan los fabricantes de dispositivos lógicos programables, sin embargo, su uso dificulta la portabilidad del diseño. Por este motivo, se recomienda el uso de *testbench* escritos con el lenguaje estándard. Como parte del proceso de diseño, cada módulo debe ser simulado antes de ser integrado en la descripción de más alto nivel.

Es bueno tener en cuenta los diferentes niveles de simulación que se pueden realizar a un sistema bajo prueba; la simulación más rápida es la que tiene en cuenta únicamente el lengiaje de descripción de hardware utilizado, sin embargo, no es posible garantizar que los resultados del circuito sintetizado sean los mismos que la simulación del lenguaje; por esto, existe la simulación post-síntesis, en la que se simula el RTL (lógica de transferencia de registros) o las compuertas lógicas básicas obtenidas del proceso de síntesis, esta simulación garantiza que el circuito obtenido del proceso de síntesis se comporta como lo deseamos; el tercer nivel de simulación se obtiene cuando se adiciona un modelo de tiempo al diagrama de compuertas del nivel anterior, en este nivel, se tienen en cuenta las capacitancias de carga y la capacitancia de los caminos de interconexión para obtener el retardo de cada elemento del circuito, esta simulación es la más precisa y permite conocer la velocidad máxima a la que puede operar el sistema, esta simulación en algunos entornos de desarrollo recibe el nombre de *simulación post place & route*.

Pruebas

Aunque la simulación es una buena herramienta para la detección de errores, es necesario realizar una prueba sobre el circuito configurado en el dispositivo lógico programable, para esto existen dos opciones: realizar el montaje de la aplicación y probar la funcionalidad del dispositivo configurado, dependiendo de la complejidad del sistema esta puede ser una tarea tediosa; la segunda opción es utilizar el puerto JTAG para aplicar los vectores de prueba y capturar los resultados, este proceso se describirá en la siguiente sección.

Implementación de un divisor de n bits sin signo

El proceso de división de números binarios es similar al de números decimales: Inicialmente se separan dígitos del Dividendo de izquierda a derecha hasta que la cifra así formada sea mayor o igual que el divisor. En la figura **??** separamos el primer dígito de la derecha (0) y le restamos el divisor (la operación de resta se realizó en complemento a dos), el resultado de esta operación es un número negativo (los números negativos en representación complemento a dos comienzan por 1). Esto indica que el número es menor que el divisor, por lo tanto, colocamos un cero en el resultado parcial de la división (este dígito será el más significativo) y separamos los dos primeros dígitos (00) y repetimos el proceso.

Figura 0.8: División de numeros binarios.

Sólo hasta el sexto resultado parcial obtenemos un cero en la primera cifra de la resta (recuerde que en complemento a dos los números tienen una longitud fija en nuestro caso 4 bits, sí una operación provoca un bit adicional este se descarta, los bits descartados se encerraron en líneas punteadas en la Figura **??**), lo que indica que el número es mayor o igual que el divisor, en este caso, se coloca un '1' en el resultado parcial y se conserva el valor de la operación de resta, el cual se convierte en el nuevo residuo parcial, este proceso se repite hasta haber "bajado " todos los dígitos del dividendo.

En la figura

Figura 0.9: Algoritmo para la división de numeros binarios.

Identificación de componentes del camino de datos e interconexión

En la figura **??** podemos observar como se obtienen los componentes del camino de datos a partir del diagrama de flujo del divisor; se necesita un registro de corrimiento a la izquierda donde se almacena el Dividendo (DV) y las cifras que se van separando (A), un contador que cuente el número de bits que se han "bajado", un restador (sumador en complemento a 2) para determinar si el número separado del dividendo "cabe" en el divisor (observando el bit más significativo MSB), y un comparador que indique que el valor del contador llegó a cero.

Figura 0.10: Identificación de componentes del camino de datos para la división de numeros binarios.

En la figura **??** se muestra la interconexión de los elementos del camino de datos y se muestran las señales de control. De nuevo, las señales que se activan en el mismo punto del diagrama de flujo pueden agruparse, por esto, la señal de inicialización del registro A, la carga de DV y la inicialización del contador se realizará con la señal *INIT*; el registro de desplazamiento a la izquierda va almacenando el resultado de la división a medida que se van utilizando los bits más significativos del dividendo, con esto se reduce el número de componentes, la señal *DV0* ayuda a formar el resultado; la señal *SH* realiza el corrimiento a la izquierda del registro $\{A,DV\}$ con lo que en A queda el número que se va separando del dividendo y en DV el resultado de la división; la señal *LDA* carga el resultado de la resta entre A y el divisor únicamente cuando el resultado de la resta es positivo, esto es cuando la señal *MSB* es igual a 1; la señal *DEC* hace que el valor del contador disminuya en 1, y la salida Z se hace 1 cuando el valor de este contador llega a cero indicando que el algoritmo terminó.

De lo anterior tenemos que la unidad de control tiene como entradas las señales: *Reset*, *Start*, *MSB* y Z; y como salidas: *INIT*, *DV0*, *SH*, *DEC* y *LDA*; de nuevo los bloques de decisión del diagrama de flujo del algorítmo hacen referencia a las entradas de la unidad de control.

Unidad de control

En la Figura **??** se muestra la relación existente entre el diagrama de flujo y el diagrama de estados de la unidad de control del divisor binario.

Contador de número de unos

En este ejemplo realizaremos un circuito que cuente el número de bits en una cadena de N bits:

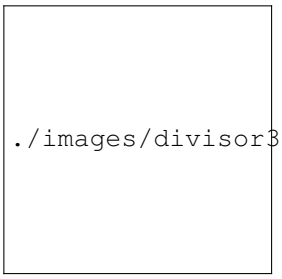

Figura 0.11: Interconexión del camino de datos para la división de numeros binarios.

Figura 0.12: Diagrama de estados de la unidad de control para la división de numeros binarios.

Circuito para determinar la raíz cuadrada de un número binario