CIC, DM, KP

Diseno de sistemas digitales

1 de febrero de 2015

Springer

Capítulo 1
Máquinas de Estado Algorítmicas (ASM)

Resumen La lógica de transferencia de registros es una de las técnicas fundamentales para la descripción del comportamiento de los circuitos secuenciales, los cuales pueden ser implementados con un grupo de registros, una unidad que realiza operaciones arirméticas y/o lógicas y una máquina de control la cual maneja la transferencia o envío de datos de un registro a otro. La secuencia o flujo de datos al realizar ese proceso de transferencia: registros, unidad aritmético-lógica, registros; es lo que permite que se generen los algoritmos. Así, todo sistema secuencial (algoritmo) puede ser visto como la unión de un camino de datos (registros y operaciones) y una unidad de control que determina cuando y cuales operaciones se deben realizar, al igual que las transferencia o movimiento de datos entre registros.

1.1. Introducción

Para el diseño de circuitos digitales existen diferentes metodologías, técnicas y herramientas; las cuales corresponden a los diferentes niveles de abstracción en que se puede concebir un sistema. El nivel mas utilizado y por el cual podemos iniciar a comprender mejor el mundo digital es el nivel conocido como “RTL” (en inglés, Register Transfer Level) o Nivel de Transferencia de Registros, en donde los diseñadores abordan la solución de un problema al nivel de operaciones y de tranferencia de datos entre registros, lo cual ocurre durante muchos ciclos de reloj [REF GAJSKI BUSCAR]. En este nivel de abstracción no se aborda el problema desde la perspectiva de las dimensiones geométricas o conexiones físicas concretas del sistema, sino que se aborda desde el nivel comportamental desarrollado por cada uno de los pasos que permiten realizar esa manipulación de datos almacenados en registros, hasta la ejecución de operaciones aritmetico lógicas; finalizando nuevamente en registros en donde se van guardando los resultados. Es este el punto de partida para abordar el desarrollo estructurado de soluciones en hardware, que para problemas de mediana y baja complejidad hacen que sea una solución acertada. Para problemas

con complejidades altas, se hace necesario ir a un nivel de abstracción más alto, en donde los conceptos adquiridos en este capítulo son fundamentales para poder tener un acercamiento a dicho nivel. A continuación se presenta.....

1.2. Metodologías de diseño RTL

Una primera alternativa para llegar a diseñar un sistema digital nace de las propias necesidades que entregan las especificaciónes del problema, en donde se busca realizar la integración de diferentes bloques, elementos de librería o periféricos (p.e decodificadores, multiplexores, sumadores, restadores, contadores, registros de desplazamiento) que conlleven a tener una arquitectura del sistema. Esta arquitectura puede llegar a tener elementos netamente combinacionales, o secuenciales, o la combinación de ambos.

En el caso más frecuente de sistemas digitales como lo es el sincrónico, se requiere de elementos secuenciales que siempre estan gobernados bajo la misma señal de reloj, la cual tiene una frecuencia de operación muy superior a cualquiera de las frecuencias de operación de las demás señales de entrada del sistema. Adicionalmente uno de los métodos más conocidos para el diseño de circuitos digitales de cierta complejidad se basa en la utilización de máquinas de estado finito "FSM" (en inglés Finite State Machine), a donde llegan todas las señales de control las cuales controlan o manipulan los demás bloques o elementos de librería que componen dicha arquitectura.

Las máquinas de estado finito utilizan una representación por medio de círculos los cuales representan los estados o momentos de operación en determinado instante de tiempo, definido por condiciones de operación que controlan los demás bloques en dicho instante y flechas que interconectan los diferentes estados, las cuales representan los condiciones de cambio entre estados o instantes de tiempo.

GRÁFICA DE PLANTILLA DE PIRB

Para poder tener un acercamiento de la funcionalidad de cualquier sistema hacia su equivalente máquina de estados, es necesario hacer una descripción previa con un algoritmo, es decir una secuencia de operaciones. Estas descripciones previas facilitan la implementación llegando a tener circuitos aritmético o lógicos basados en estados los cuales por medio de secuencias dan solución al algoritmo. Dicho algoritmo permite identificar adicionalmente las operaciones que puedan ejecutarse concurrentemente en el caso de soluciones completamente hardware; o permite identificar las operaciones secuenciales que dan orden al desarrollo de los algoritmos.

En el caso de máquinas de estados complejas con gran cantidad de estados y señales, la representación con círculos y flechas puede llegar a ser confusa y muy larga. Es por esto que se hace necesario acompañar siempre cada descripción y de hecho comenzar la etapa de diseño utilizando los algoritmos que permitan manipular con rigor y consistencia las operaciones matemáticas necesarias para la ejecución de los diversos cálculos implicados. Para esto se suelen emplear grafos que muestren

dependencia entre operandos y flujos que comunican los datos del algoritmo. A este tipo de notación se le conoce como máquinas de estado algorítmicas, en las cuales se tiene una representación gráfica intuitiva de las operaciones del circuito en cada ciclo de reloj.

1.3. 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 (entre registros y la unidad aritmético lógica) y una unidad de control que determina en que momento se deben realizar las operaciones y las transferencias de los datos entre registros. En la Figura 1.1 se puede ver la estructura fundamental de una máquina de estados algorítmica. Nótese que los datos nunca pasan por la unidad de control y que la comunicación entre el bloque de Control y el bloque Camino de datos se lleva a cabo por señales simples que deciden que operación se debe realizar (señal control) y señales que dan información al Bloque de control sobre el estado de la operación, lo que llamaremos banderas. Estas señales dan información al control como por ejemplo si el resultado de una operación da un número negativo, el resultado da cero, hay overflow o se genera un acarreo al realizarse la operación entre otras.

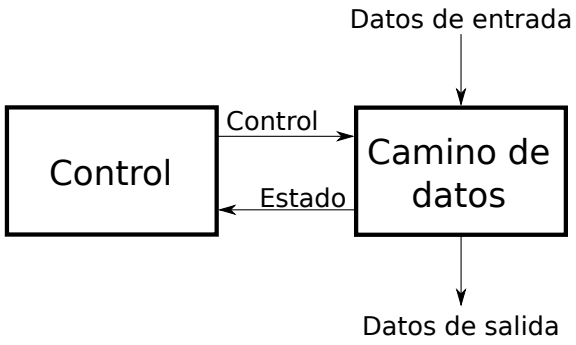


Figura 1.1 Estructura de una máquina de estados algorítmica.

A la unión de una unidad de control y un camino de datos le daremos el nombre de máquina de estados algorítmica (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 ya sea a nivel gráfico o en texto.
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.

1.3.1. Ejemplo de ASM: Multiplicación de números binarios

El algoritmo de multiplicación que se implementará se basa en productos parciales (PP); el primer producto parcial siempre es cero (ver Figura 1.2, 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 todos los bits del multiplicador y el último producto parcial es el resultado final.

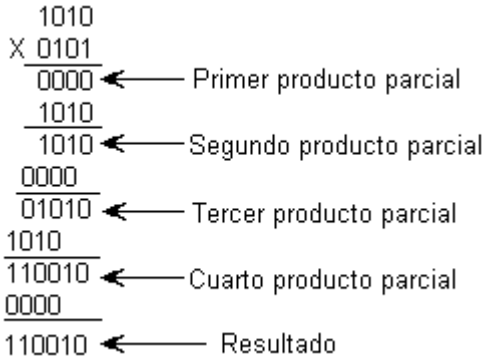


Figura 1.2 Multiplicación de números binarios usando productos parciales.

1.3.1.1. Diagrama de Flujo

En la Figura 1.3 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 parcialmente cero. En este caso se utiliza un corrimiento a la izquierda para obtener el siguiente bit del multiplicador, si por ejemplo al número *1010* se le realiza un corrimiento a la derecha se obtiene el número *0101*, con lo que el bit menos significativo corresponde al segundo bit de *1010*, si se realiza otro corrimiento a la derecha se obtiene *0010* y de nuevo el bit menos significativo corresponde al tercer bit de *1010*, al realizar de nuevo un corrimiento se obtiene *0001*, con lo que tendríamos todas las cifras del multiplicador de forma consecutiva en el bit 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.

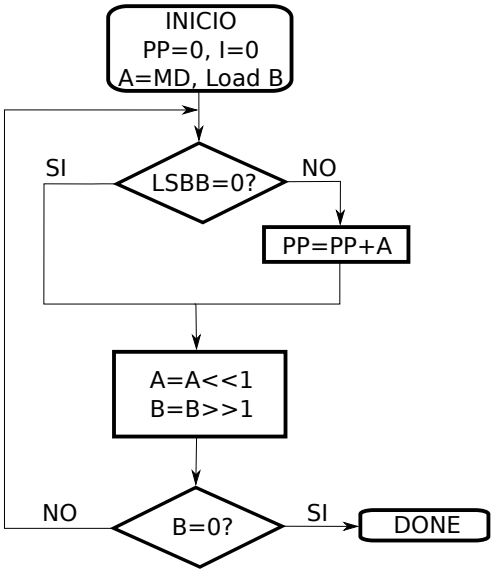


Figura 1.3 Diagrama de flujo para la multiplicación de números binarios.

Una vez conocido el funcionamiento del sistema se procede a realizar el diagrama de caja negra de entradas y salidas. En la Figura 1.5 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.

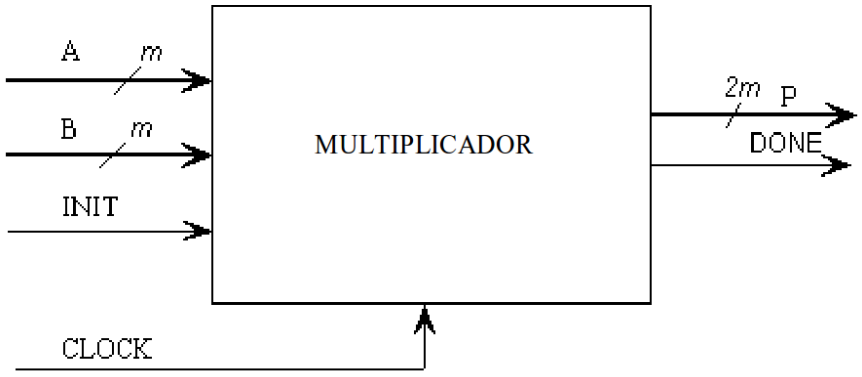


Figura 1.4 Diagrama de caja negra para el multiplicador de números binarios.

1.3.1.2. 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.

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

En la figura 1.6 se muestra la interconexión de los componentes del camino de datos y las señales que lo controlan.

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 mul-

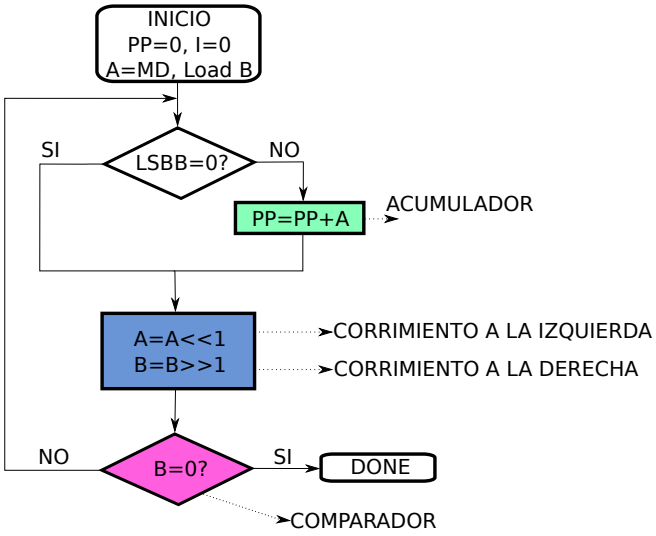


Figura 1.5 Identificación de los componentes del camino de datos para el multiplicador de numeros binarios.

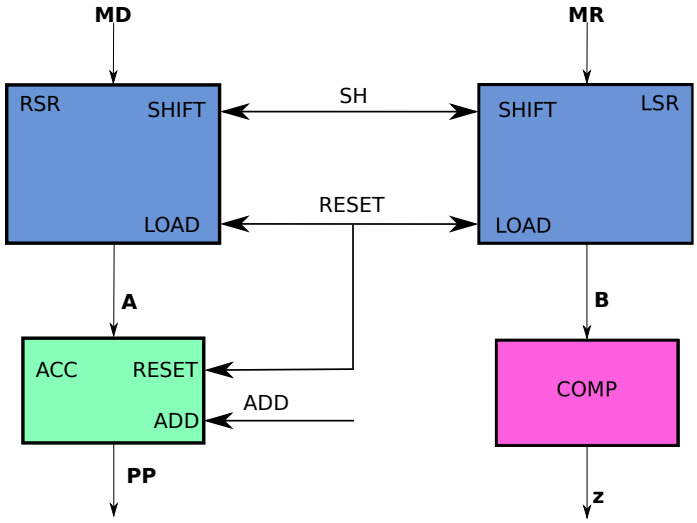


Figura 1.6 Identificación de las señales de control e interconexión del camino de datos.

tiplicando, 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.

1.3.1.4. 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 1.7 se muestra la relación entre el diagrama de flujo y el diagrama de estados.

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

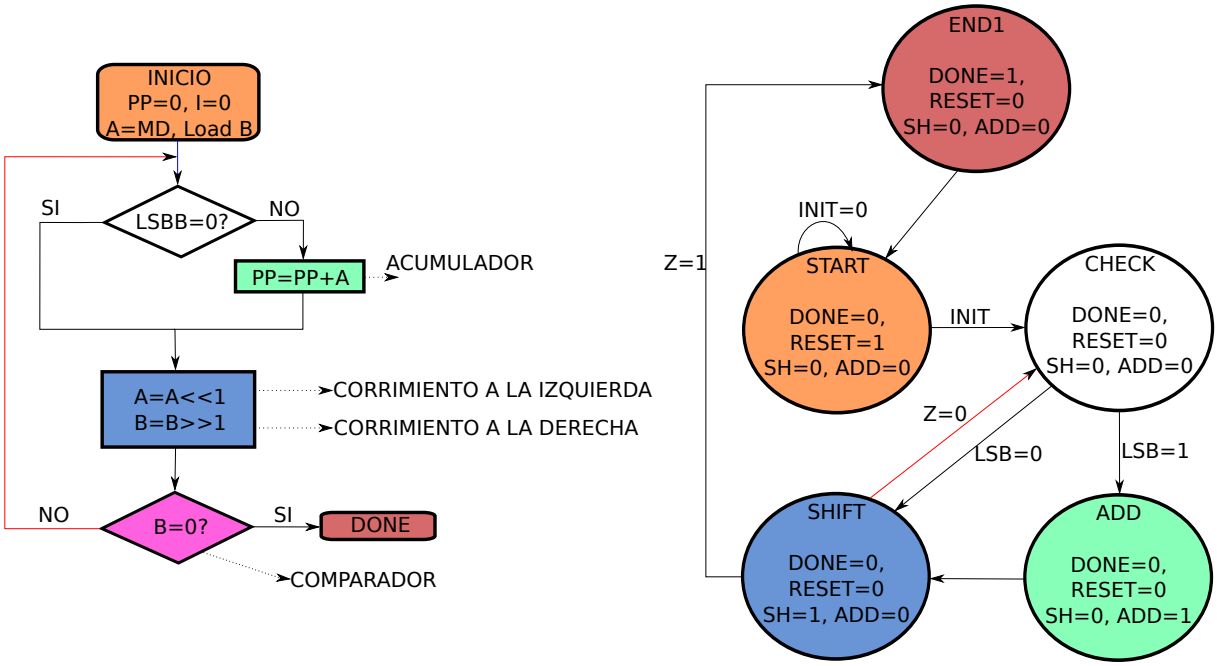


Figura 1.7 Diagrama de estados de la unidad de control del multiplicador binario.

a un estado de la máquina de control y las transiciones entre ellos son idénticas, observe las líneas del mismo color en la figura 1.7

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.

1.3.1.5. 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 estándar del lenguaje; por esto, se recomienda utilizar únicamente entrada de texto en las descripciones.

1.3.1.6. 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 estándar. 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 lenguaje 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*.

1.3.1.7. 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. (OJO)

1.3.2. 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 1.8 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.

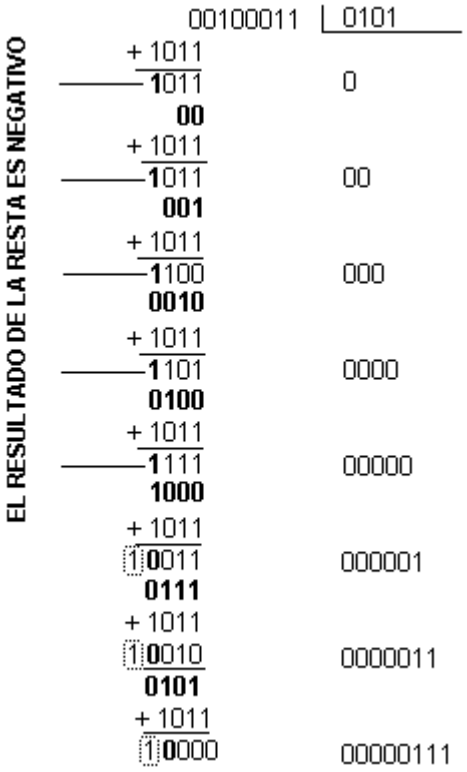


Figura 1.8 División de numeros binarios.

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 1.8), 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

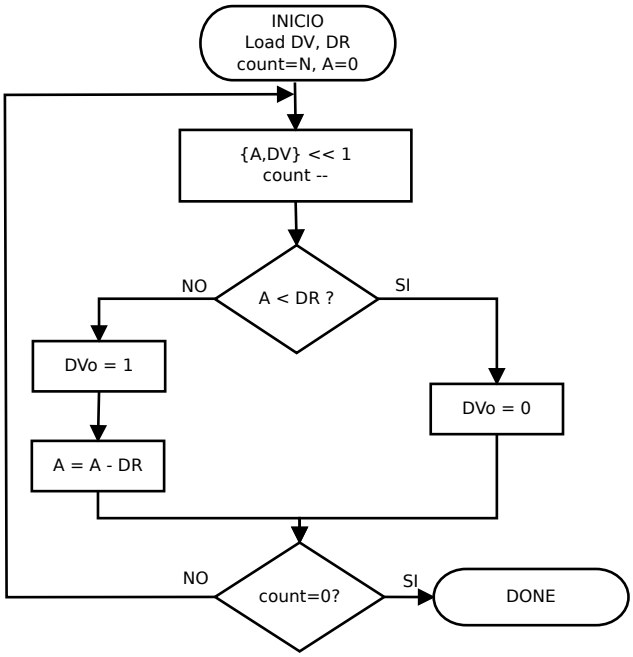


Figura 1.9 Algoritmo para la división de numeros binarios.

1.3.2.1. Identificación de componentes del camino de datos e interconexión

En la figura 1.10 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.

En la figura 1.11 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 reali-

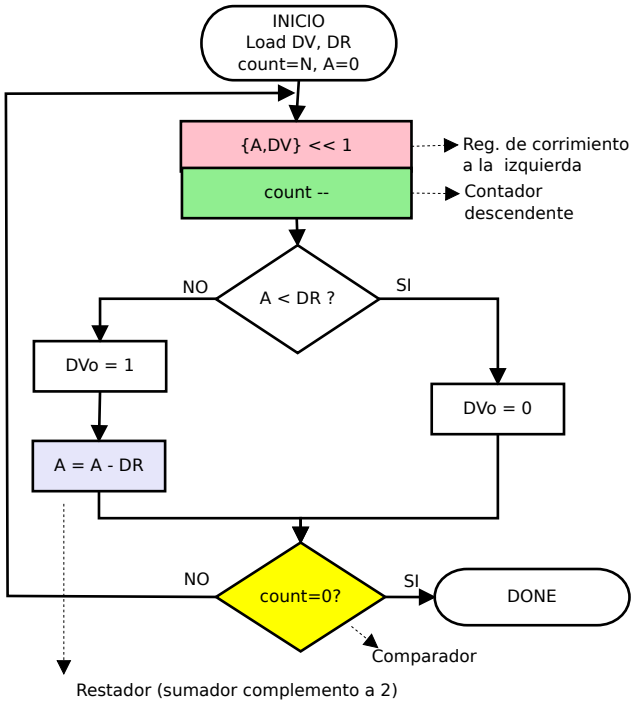


Figura 1.10 Identificación de componentes del camino de datos para la división de numeros binarios.

zará 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.

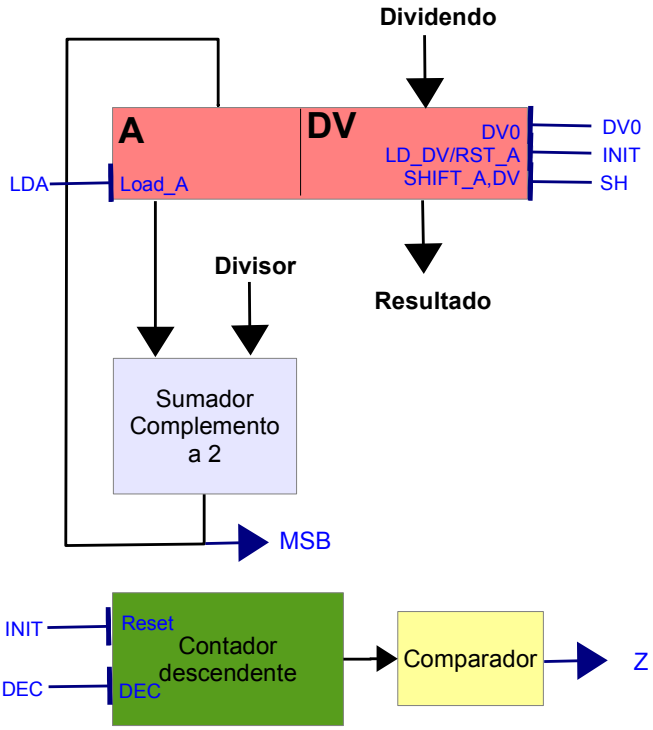


Figura 1.11 Interconexión del camino de datos para la división de numeros binarios.

1.3.2.2. Unidad de control

En la Figura 1.12 se muestra la relación existente entre el diagrama de flujo y el diagrama de estados de la unidad de control del divisor binario.

1.3.3. Contador de número de unos

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

1.3.4. Circuito para determinar la raíz cuadrada de un número binario

ewruewpor upoewrupoew urpoewf

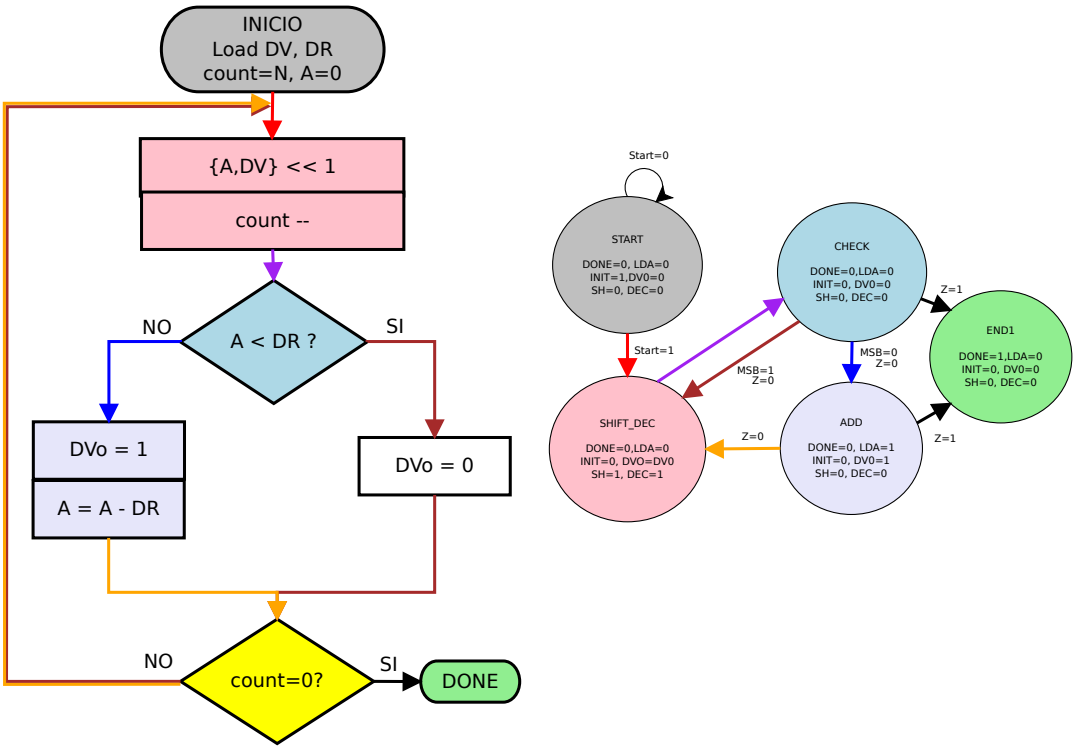


Figura 1.12 Diagrama de estados de la unidad de control para la división de numeros binarios.

1.4. Del hardware al software: datapath

Como se puede ver en los ejemplos anteriores, las arquitecturas encontradas dependen básicamente del problema que se esté trabajando, siendo la incorporación de los elementos hardware algo dependiente de la necesidad de las operaciones del algoritmo. No hay un orden establecido ni una arquitectura única que permita solucionar varios problemas, sino por le contrario la arquitectura seleccionada obedece solo al requerimiento en cuestión. De hecho cabe notar que no sobra ningún bloque del sistema dado que todos estan puestos cuidadosamente para cumplir alguna función. En todos los casos se sigue manteniendo que se utiliza la transferencia de registros y el flujo de los datos para poder llegar acertadamente a una implementación, que a nivel de hardware sea de los mas sencilla. En este caso, la incorporación de periféricos específicos basados en celdas básicas se hace necesaria y de hecho es requisito fundamental para la implementación. Cualquier elemento que permita obtener una respuestas mas eficiente en cuanto a tiempo de ejecución se puede utilizar y solo hace falta optimizar la máquina de estados para que controle cada uno de los periféricos del sistema. Es así como aparece el concepto de concurrencia muy ligado a lo que es una solución en hardware. La concurrencia permite realizar operaciones

al mismo tiempo, siempre y cuando los datos que requieran dichas celdas básicas no sean los mismos, o que por lo menos su funcionalidad no esté relacionada. Básicamente lo que se quiere mostrar es que se pueden llevar a cabo varias operaciones al tiempo.

1.5. Camino de datos o datapath

Dado que algunas veces se requiere tener un solo sistema que resuelva diferentes problemas, surge la necesidad de no tener alternativas netamente concurrentes, en donde la incorporación de periféricos sea de manera indeterminada según la necesidad del diseñador. Para esto se hace necesario plantear estructuras mas genéricas en donde la implementación de los algoritmos dependa claramente de recursos que ya están disponibles y están dados por una arquitectura definida invariante

Dichas arquitecturas constan básicamente de 3 elementos. Una unidad aritmético - lógica la cual se encarga de realizar las operaciones que requiera el algoritmo. Es de notar que si la unidad aritmético lógica no realiza alguna operación del algoritmo, es necesario implementar dicha operación en base a nuevos algoritmo; es decir hacer el algoritmo para realizar dicha operación.

En la medida que las unidades aritmético - lógicas tengan más instrucciones, se hacen mas costosas porque tienen que tener todo el harwdare implementado para dichas funciones. Sin embargo eso representa que no se requieren ciclos de máquina o ciclos de instrucción para realizar dicha operación dado que la implementación de esa operación es netamente hardware.

Lógicamente las arquitecturas no tienen implemenentadas todas las posibles funciones matemáticas o funciones aritméticas, por lo que es necesario realizar algoritmos que suplan dichas funciones, para lo cual se utilizan los recursos disponibles en cuanto a funciones básicas de la unidad aritmético lógica.

El otro elemento importante de la arquitectura básica es el banco de registros en donde se mantiene almacenada la información o los datos. En el caso genérico de esta ASM se tienen registros de propósito general, pero cuando hablemos de arquitecturas de computadores tendremos que hablar adicionalmente de registros de acceso a memoria, registros programados, registro especiales etc.

Finalmente se tiene la unidad de control la cual se encarga de manipular las señales que controlan tanto la unidad aritmético - lógica como los registros. Dicha unidad de control genera señales como por ejemplo seleccionar la operación que quiere hacer la unidad aritmético lógica, generar señales de control como lectura o escritura sobre los registros etc. Esta unidad se encarga de coordinar y realizar acciones necesarias para que se ejecuten las instrucciones. (En el caso de arquitectura de computadores nos referiremos también a instrucciones)

Cada elemento se debe comunicar con los demás para poder realizar su función, lo que se logra por medio de una estructura de interconexión la cual se puede hacer por medio de:

a) Uno de conexión directa: Se realiza la conexión cableda directamente de la salida de un elemento a la entrada del otro. En caso de requerirse lógica de enrutamiento, se emplean elementos digitales adecuados como multiplexores.

b)Uso de buses. Se define un conjunto de líneas para interconectar múltiples elementos entre sí. Se debe implementar una lógica de control que garantice enrutamiento del dispositivo fuente al destino. Debe evitarse que dos dispositivos sean fuente en el bus de la forma simultánea pues esto provocaría cortocircuitos en las líneas cuando estén sometidas a valores lógicos diferentes. En la figura 1.13 se puede ver en esquema basado en buses.

1.6. Tipos de datapaths

Existen diferentes arquitecturas sobre las cuales están organizados los tres elementos básicos: control, unidad artimético - lógica y banco de registros. A continuación se presentan dos esquemas básicos mas cónocidos:

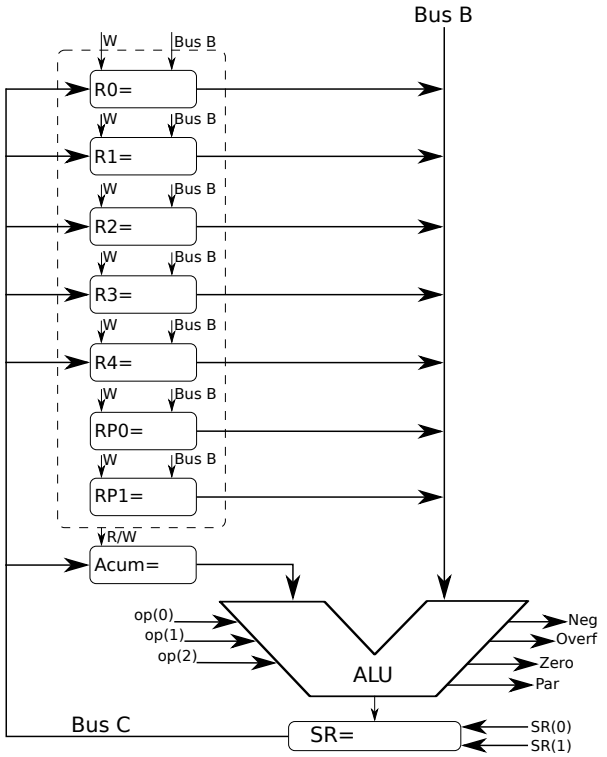
1.6.1. Datapath basado en registro acumulador

En la figura 1.13 se puede ver un esquema en donde se tiene un datapath donde aparece un registro (ACUM) que se denomina acumulador por la función que cumple.. En este tipo de arquitecturas todos los resultados de las operaciones se lleva a cabo en el registro acumulador, es decir los resultados parciales siempre pasan a través de este registro. Para realizar una operación el programador debe tomar el primer operando y llevarlo al acumulador y realizar la operación siguiente siempre con el acumulador. El resultado se guarda en el acumulador nuevamente. Dicho resultado eventualmente se puede transferir a otro registro. Ejemplos de procesadores basados en acumulador son el IAS de Von Neumann, el M6502 y el 6809 de Motorola. Este último cuenta con dos acumuladores A y B

1.6.2. Datapath sin acumulador

En un datapath sin acumulador, los resultados de las operaciones parciales se pueden ir almacenando en cualquiera de los registros de propósito general. Es necesario realizar un algoritmo de forma mas ordenada de manera que se conozca en cada momento que registro tiene los valores necesarios para continuar con las operaciones. En la figura 1.14 se puede ver un esquema de esta arquitectura.

En este tipo de arquitecturas se utilizan muchos registros de propósito general para almacenar temporalmente los operandos. Este tipo de arquitecturas tiene la



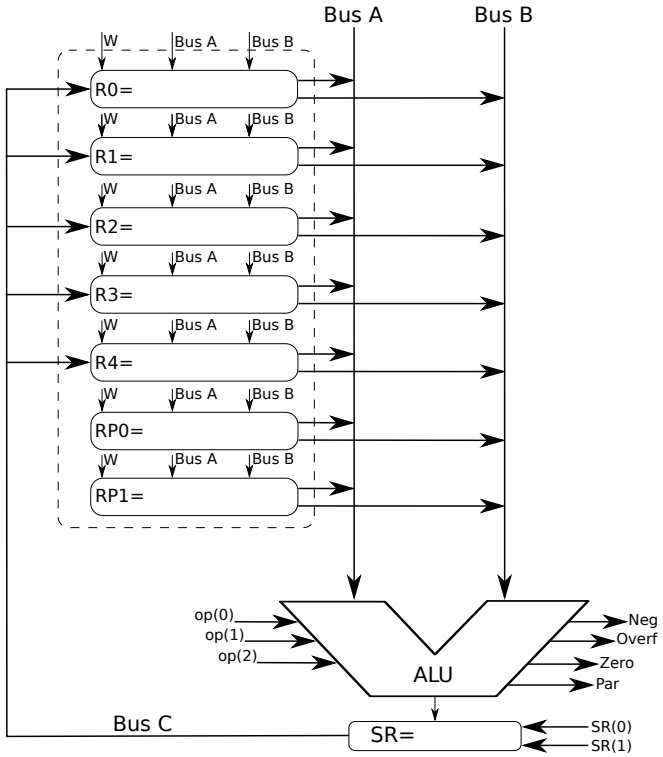
R0		R1		R2		R3		R4		RP0		RP1		Acum	ALU			SR	
W	Bus B	W	Bus B	W	Bus B	W	Bus B	W	Bus B	W	Bus B	W	Bus B	R/W	op(2)	op(1)	op(0)	SR(1)	SR(0)

Figura 1.13 Datapath con acumulador

ventaja que los datos mas frecuentes se cargan solamente una vez desde la memoria principal. En este estilo de diseño respecto al manejo de los operandos, se promueve el uso de muchos registros de propósito general para almacenar los operandos temporalmente. Esto tiene la ventaja que los datos más frecuentes sólo se cargan una vez desde la memoria. Una arquitectura típica consta de un banco de m registros de propósito general (Ro... Rm-1). Son llamados de esta forma porque en cualquiera de ellos se pueden mantener datos, direcciones de memoria o el resultado de alguna operación aritmética o lógica. El diseñador puede emplear estos registros para realizar las diferentes operaciones de un algoritmo.

Algunos procesadores tienen algún destino específico para algunos de ellos; el registro F (Flag) o PSW (Processor status Word) es un registro de banderas y/o códigos de condición que mantiene información importante del estado del procesador o del desarrollo del programa. Por ejemplo, cuando el resultado de una operación fue cero (bandera Z = 1) o cuando ocurrió un acarreo (bandera C = 1).

Con procesadores de registros generales podemos tener instrucciones de dos y tres operandos. En máquinas 2 operandos, el destino o resultado de la operación está implícito en uno de los dos operandos. Este fenómeno se denomina lectura destructiva de operandos, pues el valor del registro que es a la vez operando y destino se pierde por sobre escritura del resultado.



R0			R1			R2			R3			R4			RP0			RP1			ALU			SR	
W	Bus A	Bus B	W	Bus A	Bus B	W	Bus A	Bus B	W	Bus A	Bus B	W	Bus A	Bus B	W	Bus A	Bus B	W	Bus A	Bus B	op(2)	op(1)	op(0)	SR(1)	SR(0)

Figura 1.14 Datapath sin acumulador

1.6.3. Ejemplo: Diseño simple de una operación de suma

1.6.4. Solución hardware

En la figura 1.15 se presenta una solución hardware para la realización de una operación simple de suma.

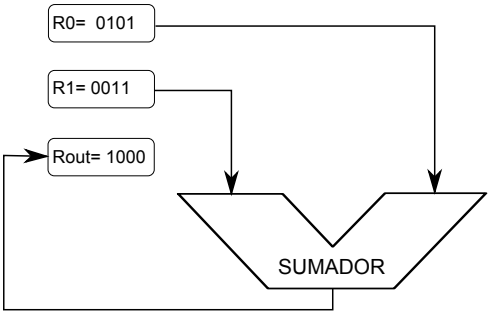


Figura 1.15 Ejemplos simple de una suma. Solución hardware

1.6.5. Solución software en datapath sin acumulador

En la figura 1.16 se presenta una máquina de estados de la una solución en datapath sin acumulador para una operación simple de suma.

Figura 1.16 sumamaquinaestadossinacumulador

1.6.6. Ejemplo: Cálculo de valor absoluto

En la figura 1.17 se presenta una solución hardware para la realización del cálculo de una operación de valor absoluto.

En la figura 1.18 se presenta una máquina de estados de la una solución en datapath sin acumulador para la realización de una operación de valor absoluto.

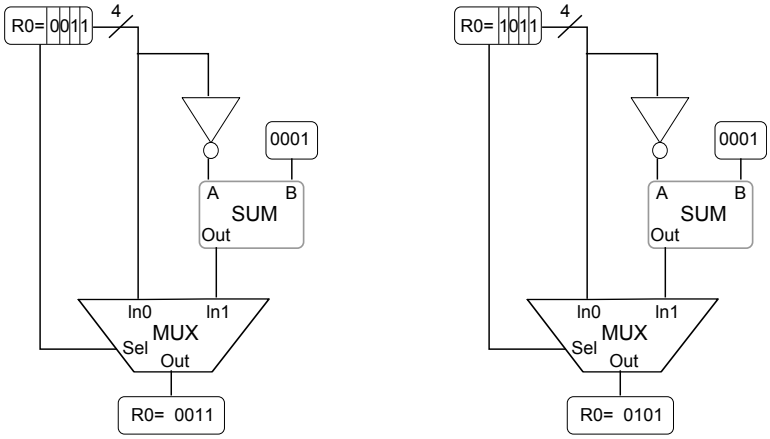


Figura 1.17 absolutoHW

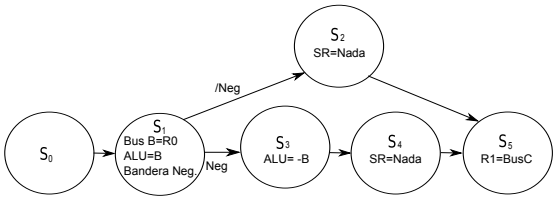


Figura 1.18 absolutoMaquinaEstados

1.7. Sistema en Silicio J1

En la figura 1.19 se muestra el diagrama de bloques de un Sistema en Silicio basado en la unidad de procesamiento central J1. Un Sistema en silicio (SoC) es la unión de una unidad de control (CPU) y un grupo de periféricos (multiplier, divider, uart); la CPU está encargada de leer, decodificar y ejecutar una lista de instrucciones que implementan un determinado algoritmo; a diferencia de las máquinas de estado algorítmicas explicadas anteriormente (cuya función es fija) esta arquitectura permite la ejecución de cualquier algorítmo con solo cambiar las instrucciones que deben ser ejecutadas.

Los periféricos realizan tareas fijas y permiten adicionar funcionalidad a la CPU, en este ejemplo los periféricos *multiplier* y *divider* posibilitan la realización de multiplicaciones y divisiones, mientras que el periférico *uart* permite la transmisión de información utilizando un protocolo serial asíncrono.

La comunicación entre los periféricos y la CPU se reliza a través de los buses de *datos, direcciones* y *control*. Existen dos tipos de comunicación: lectura, la CPU lee información suministrada por el periféricos; escritura: la CPU envía información a los periféricos. En ambos casos es la CPU la que lee o escribe, los periféricos nunca realizarán operaciones de lectura o escritura a la CPU. Este proceso centralizado

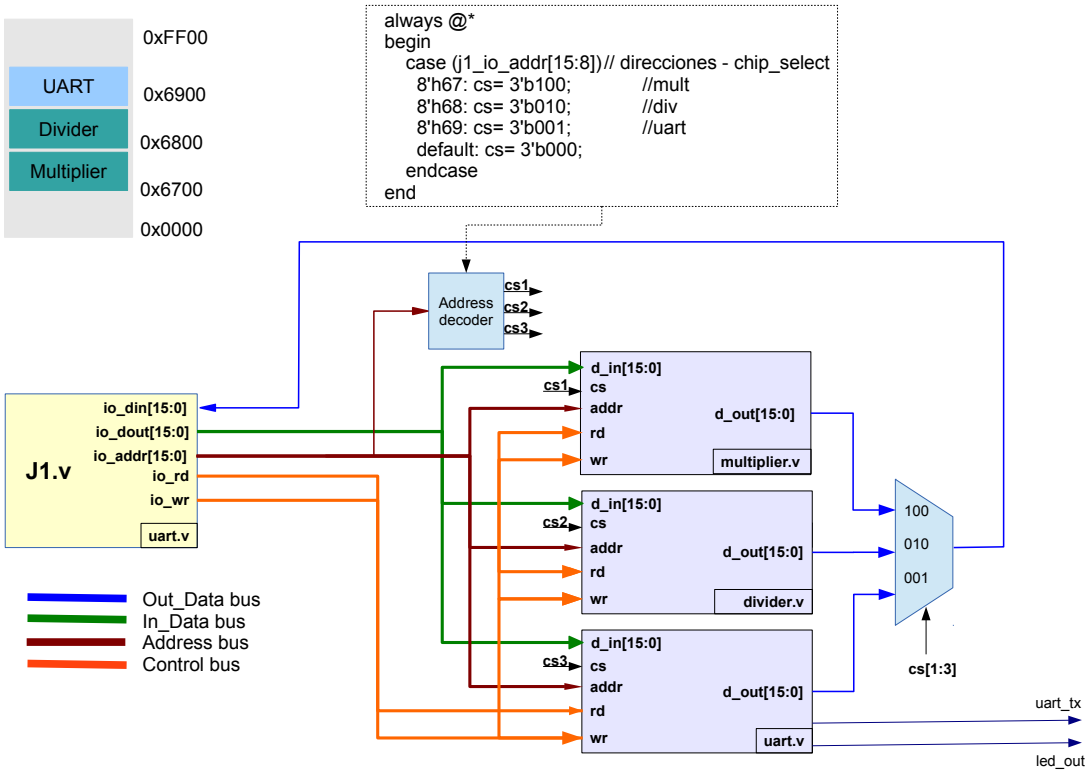


Figura 1.19 Diagrama de bloques del J1_SOC

permite coordinar las acciones de los periféricos y cede el control de la ejecución del algortimo a la CPU.

1.7.1. Ejemplo de comunicación CPU-periférico

Como puede verse en la figura 1.19 cada periférico se conecta con la CPU mediante los buses de datos, dirección y control. El bus de datos está dividido en dos partes uno para la salida de datos de la CPU (escritura) y otro para la entrada de datos (lectura), esto es necesario debido a que internamente en una FPGA no se pueden colocar señales bidireccionales.

Las señales de control *rd*, *wr* indican el tipo de transmisión; cuando el procesador escribe a un periférico activa la señal *wr* y cuando lee información activa la señal *rd*, dicha información será transmitida por el bus de datos de salida y de entrada respectivamente. Las señales *cs1, cs2 y cs3* (chip select) activan al periférico con el que se desea realizar la comunicación, y únicamente el que se encuentre activado (solo se activa uno) participará en el intercambio de información con la CPU.

El bus de datos de salida de la CPU se conecta al bus de datos de entrada de todos los periféricos, y solo el que sea selaccionado (activando su señal *cs* procesará esa información. El bus de datos de salida de los periféricos se conecta a un multiplexor que controla la conexión de estos al bus de datos de entrada del procesador, las señales *cs1, cs2 y cs3* determinan que periférico es conectado. de esta forma cuando se activa la señal de activación del periérico se crea un camino entre el periférico y la CPU.

El bus de direcciones permite diferenciar a los periféricos asignándoles rangos de memoria a cada uno de ellos, esto recibe el nombre de *mapa de memoria*. Un mapa de memoria muestra el rango de direcciones asignado a cada periférico, estas direcciones deben ser únicas (no se puede asignar la misma dirección a mas de ún periférico). En este ejemplo: 0x0000 - 0x66FF para la *BRAM*; 0x6700 - 0x67FF para lel*multiplier*; 0x6800 -0x68FF para el *divider* 0x6900 - 0x69FF para la *uart*; el *decodificador de direcciones* (Address decoder) genera las señales de activación de los periféricos utilizando este mapa de memoria, como lo muestra el código en verilog en la figura 1.19.

Para aclarar este proceso, considere que desea escribir los datos 0x0005 en la dirección 0x6700, 0x0003 en la 0x6702 y 0x0001 en la 0x6704; estas direcciones se encuentran en el rango del multiplicador por lo que la señal *cs1* se activará y las seles *cs2* y *cs3* desactivará los otros periféricos.

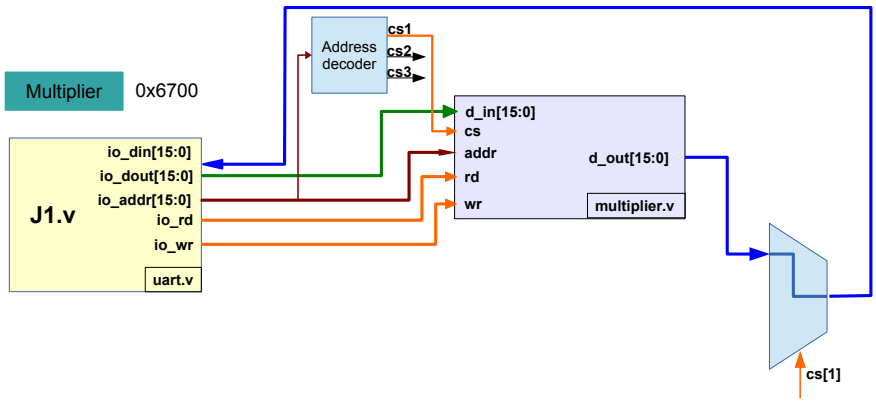


Figura 1.20 SoC J1 - Comunicación con un periférico

En la figura 1.20 se muestra el circuito equivalente al seleccionar el rango de direcciones del multiplicador, lo otros periféricos se desactivan lo que para fines prácticos equivale a su inexistencia, con lo que la CPU se encuentra totalmente conectada al multiplicador.

1.7.1.1. Escritura a un periférico

En la figura 1.21 se muestra las formas de anda de las señales asociadas a un ciclo de escritura desde la CPU a las direcciones 0x6700, 0x6702 y 0x6703.

Cuando se realiza una operación de escritura, la CPU coloca la dirección en el bus de direcciones *io_addr*, activa la señal *j1_io_wr* del bus de control y coloca el dato a escribir en el bus de datos de salida *io_dout*. Para las señales mostradas en la figura escribe 0x0005 en la dirección 0x6700, 0x0003 en la dirección 0x6702 y 0x0001 en la 0x6704.

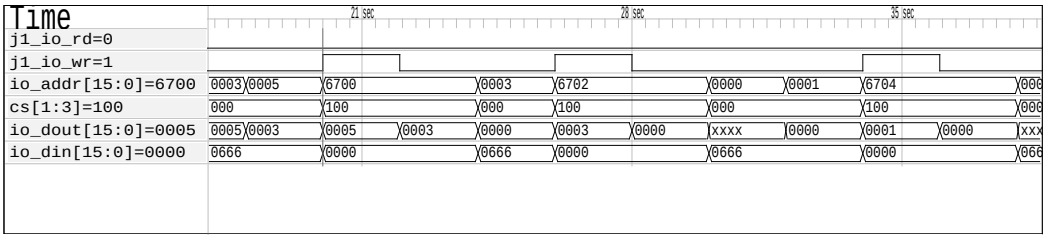


Figura 1.21 SoC J1 - Formas de onda de la escritura un periférico

1.7.1.2. Lectura a un periférico

En la figura 1.22 se muestra las formas de anda de las señales asociadas a un ciclo de lectura desde la CPU a las direcciones 0x6706, 0x6708 y ox670A.

Cuando se realiza una operación de lectura, la CPU coloca la dirección en el bus de direcciones *io_addr*, activa la señal *j1_io_rd* del bus de control y coloca el dato a escribir en el bus de datos de salida *io_din*. Para las señales mostradas en la figura lee 0x0001 en la dirección 0x6706, 0x0000 en la dirección 0x6708 y 0x000F en la 0x670A.

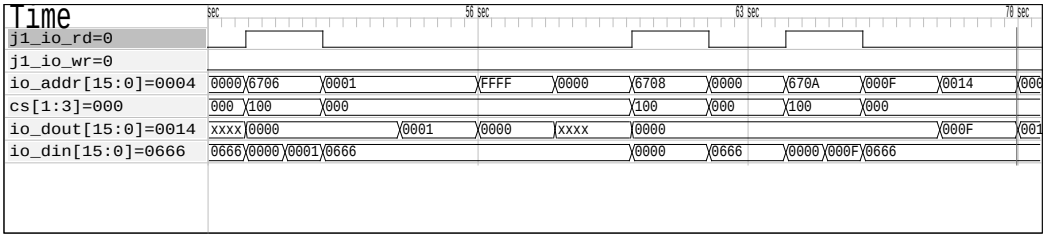


Figura 1.22 SoC J1 - Formas de onda de la lectura un periférico

1.7.1.3. Direccionamiento tipo byte

En los ejemplos de lectura y escritura descritos anteriormente se escribe y se lee 16 bits en las posiciones de memoria 0x6700, 0x6702, 0x6704, 0x6706, 0x6708 y 0x670A, la diferencia del valor entre las direcciones de memoria indica el número de bytes necesarios para almacenar el valor en este procesador. Si el bus de datos del procesador fuera de 32 bits, se tendrá que reservar 4 bytes para cada escritura, por lo que las direcciones serían 0x6700, 0x6704, 0x6708, etc. Al escribr 0x0003 en la posición de memoria 0x6700 el valor 0x00 queda almacenado en la dirección 0x6700 y el valor 0x03 en la dirección 0x6701 (ver figura 1.23).

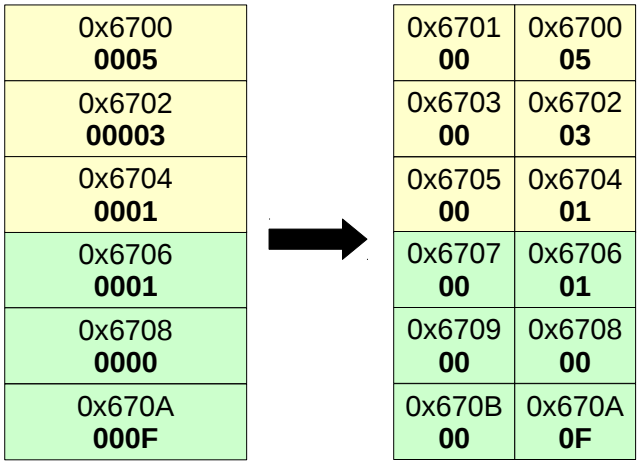


Figura 1.23 Direccionamiento tipo byte para las operaciones de lectura y escritura

1.7.2. Arquitectura de los periféricos

En esta sección mostraremos como debe ser la estructura interna de los periféricos para que puedan comunicarse con la CPU en el SoC J1. En la figura 1.24 se muestra el diagrama de caja negra para el multiplicador; este módulo tiene como entradas los operandos de 16 bits **A** y **B** y la señal **init** y como salida el resultado de 32 bits **PP** y la señal que indica que ya se realizó la operación **done**.

Se deben crear mecanismos que permitan suministrar los operandos y obtener el resultado de la operación a través de los buses de datos conectados a la CPU, para esto, el primer paso, es identificarlos y asignarles una dirección de memoria a cada uno de ellos para que la CPU pueda leerlos o escribirlos. En la figura 1.24 se muestra la asignación (puede variar a criterio del diseñador) dada a estas señales; el primer operando se encuentra asignado a la dirección *BASE + 00*, donde *BASE* es

la dirección de memoria asignada al periférico, no es necesario que este periférico este atado a esta dirección (como se mencionó anteriormente el decodificador de direcciones del SoC activa la señal **cs** correspondiente al multiplicador, por lo que esta dirección base puede ser cualquiera y no afectará el funcionamiento del periférico), por lo que solo se necesitarán las primeras 4 lineas del bus de direcciones (*io_addr[3:0]*) para seleccionar a que registro se quiere acceder.

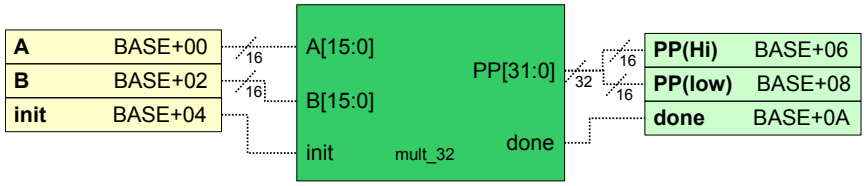


Figura 1.24 Mapa de memoria para el multiplicador

Una vez definida la dirección de memoria para la información de entrada y salida del periferico se debe adaptar el diseño para que permita el intercambio de información con la CPU. En la figura 1.25 se muestra el diagrama de bloques del multiplicador, en el podemos identificar dos multiplexores conectados a los buses de datos (*dout* y *din*) encargados de seleccionar a que señal se escribirá/leerá la señal desde/hacia la CPU.

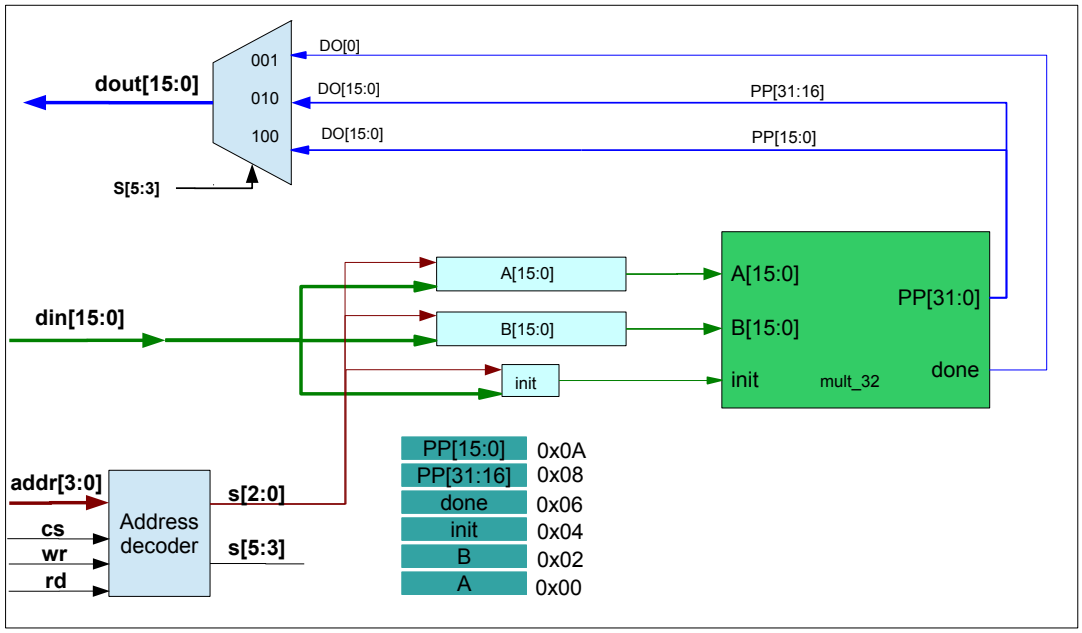


Figura 1.25 Diagrama de bloques del periférico multiplicador

1.7.2.1. Lectura del periférico

En la figura 1.26 se muestra el diagrama de bloques simplificado para la lectura. En ella podemos observar que la señal de control *s[5:3]* conecta el bus de datos de salida *dout* con las señales PP[15:0], PP[31:16] y done, el decodificador de direcciones interno activa una de estas señales de acuerdo con el mapa de memoria del dispositivo tal como se indica en el código en verilog de la figura 1.26.

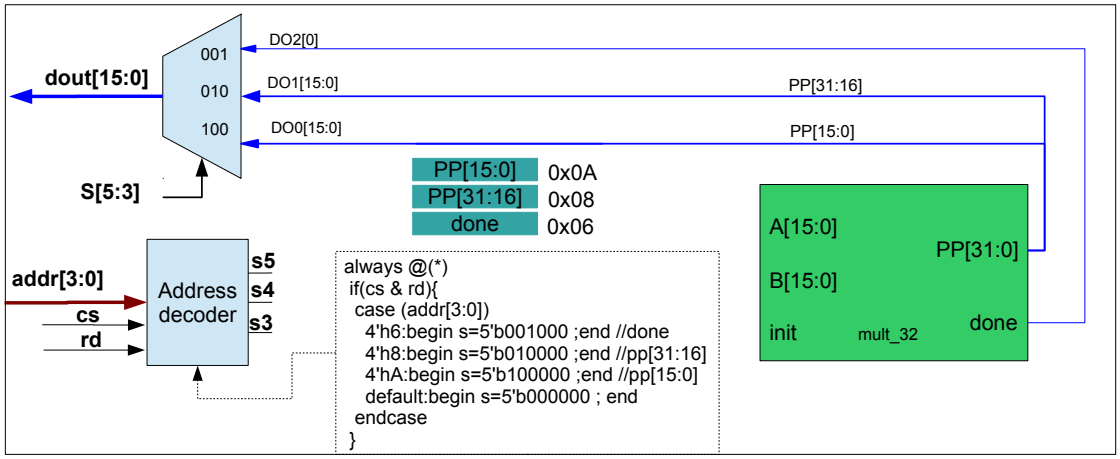


Figura 1.26 Diagrama de bloques para la lectura del multiplicador

1.7.2.2. Escritura del periférico

En la figura 1.27 se muestra el diagrama de bloques simplificado para la escritura; de forma similar a la escritura de datos desde la CPU a los priféricos, el bus de datos de entrada (*din*) está conectado a todos los registros *A*, *B* e *init*, el codificador de direciones controla la activación de dichos registros; en este caso se utilizan registros ya que es necesario mantener los operandos en el valor deseado durante la ejecución del algorítmo de multiplicación.

1.7.2.3. Diagrama de bloques del divisor

En la figura 1.28 se muestra el diagrama de bloques del divisor; su arquitectura es muy similar a la del multiplicador ya que tiene prácticamente las mismas señales, y requieren la misma información para su operación.

Periféricos similares a estos poseen la misma arquitectura, las variables son almacenadas en registros, multiplexores en los buses de entrada y salida de datos

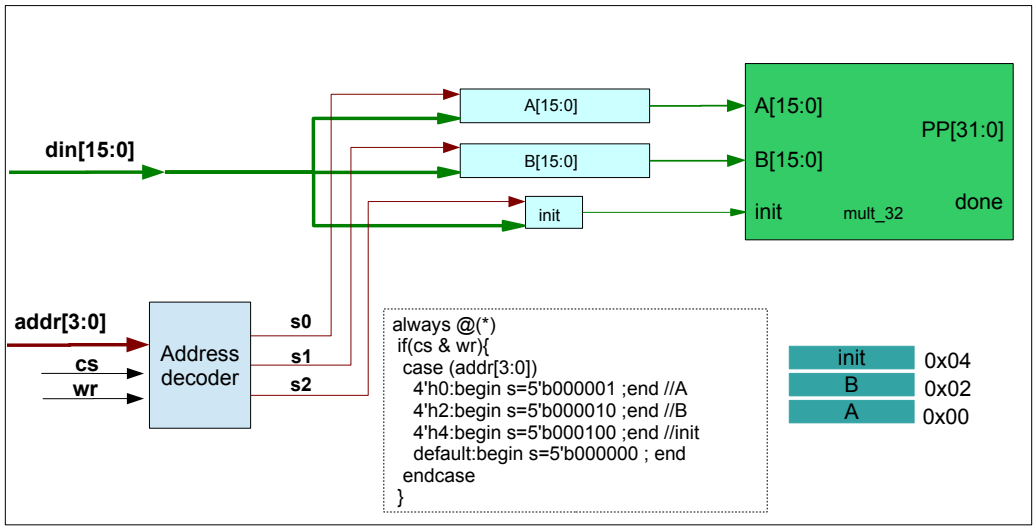


Figura 1.27 Diagrama de bloques para la escritura del multiplicador

controlan en donde se almacena o de donde proviene la información que llega o sale a la CPU. Esta arquitectura posee un decodificador de direcciones interno que controla los multiplexores de entrada y salida.

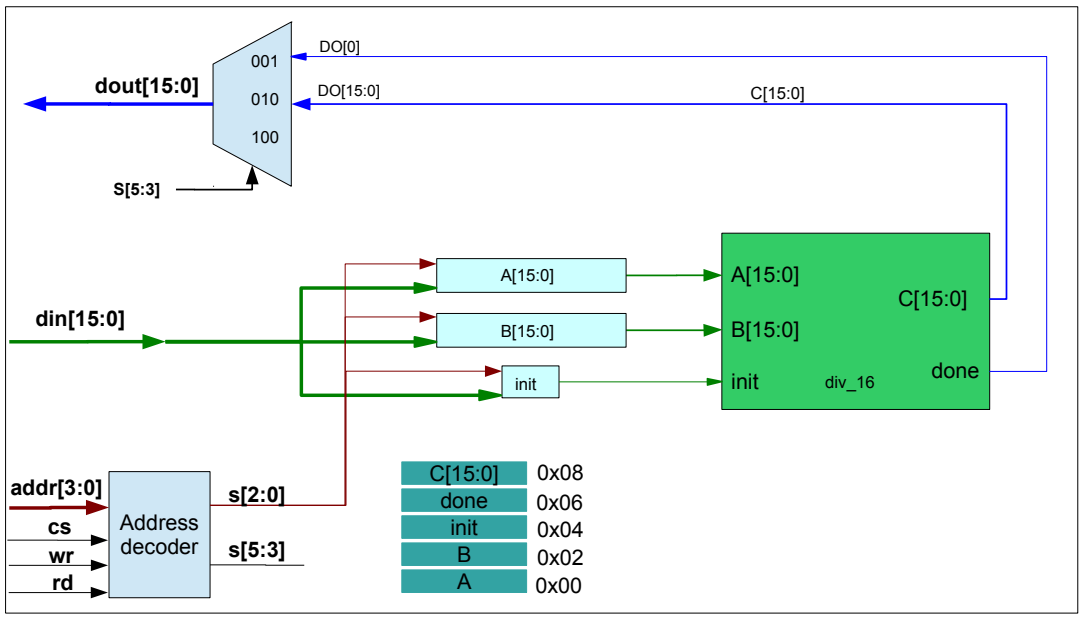


Figura 1.28 Diagrama de bloques del divisor

1.7.2.4. Intercambio de información utilizando memorias RAM de doble puerto

En los ejemplos anteriores se transfería muy poca información, y la arquitectura propuesta funciona bien bajo estas condiciones, pero cuando la cantidad de infomación aumenta es mejor utilizar una interfaz basada en memorias. La figura 1.29 muestra el diagrama de bloques de una interfaz con la CPU basada en una memoria doble puerto.

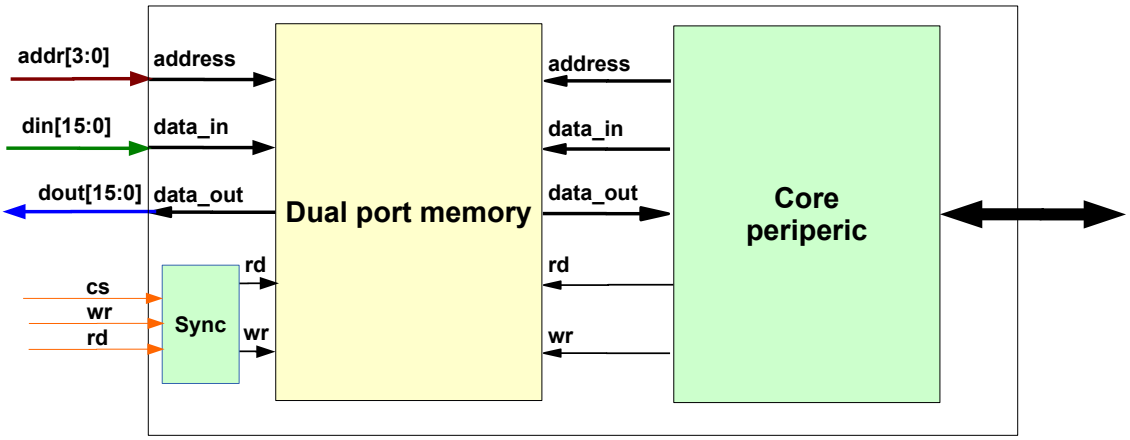


Figura 1.29 Diagrama de bloques de la interfaz basada en memoria

Las memorias RAM de doble puerto duplican las entradas de datos, dirección y control para permitir operaciones simultáneas de lectura y escritura desde dos sitios diferentes. Como puede verse en la figura 1.29 un puerto de la memoria es controlado por la CPU, mientras que el otro es controlado por el periférico; con esta arquitectura la CPU puede escribir en la RAM una serie de datos que son procesados por el periférico y los resultados son escritos en la RAM para que la CPU los utilice. Con esta configuración se reduce de forma considerable la complejidad del periférico al simplificar el decodificador de direcciones interno y reducir el número de registros para alamacenamiento de variables.