1. Ejemplos de Aplicaciones del VHDL.

1.1. INTRODUCCION

En los capítulos anteriores se describió el proceso de diseño para sistemas digitales y las herramientas disponibles (tanto software como hardware) para su implementación. En este capítulo se trataran ejemplos prácticos de diseño de sistemas digitales, iniciando con su descripción hasta llegar a su implementación final utilizando en algunos casos componentes discretos y VHDL.

1.2. ¿QUE ES UNA FSM?

Una máquina de estados es un sistema secuencial sincrónico que posee un número fijo de posibles estados. El valor de sus salidas y la transición entre los estados depende del valor de sus entradas y del estado en que se encuentra actualmente. **Todos los cambios de estado ocurren ante un determinado flanco de la señal de reloj** (ya sea de subida o bajada).

Para entender mejor este concepto imaginemos que nuestra máquina de estados es un televisor, que sólo puede sintonizar cuatro canales y que se puede controlar por un control remoto que tiene solo dos teclas para aumentar o disminuir el canal. Los estados de nuestro sistema están representados por el número de canales que se pueden sintonizar, así pues, solo tendremos cuatro posibles estados (Número fijo de estados). En la Figura 1 se muestra un diagrama de nuestro sistema:

Figura 1. Ejemplo de FSM.

Ahora hagámonos una pregunta: ¿Qué nos produce un cambio en el estado de nuestro sistema?, Lo único que nos puede producir un cambio de estado (Canal sintonizado) es un cambio en las teclas de nuestro control remoto (Entradas del sistema). Observemos como actúa el sistema ante cambios de las entradas:

Si oprimimos la tecla de aumentar el canal, el televisor aumentará en uno la cadena sintonizada, por ejemplo, si estaba en el canal 1 pasará al 2, si estaba en el 2 al 3, del 3 al 4 y del 4 al 1 nuevamente.

Si oprimimos la tecla menos es canal diminuirá pasando del 4 al 3, del 3 al 2, del 2 al 1 y del 1 al 4 nuevamente. Si no oprimimos una tecla, el televisor debe permanecer en la cadena sintonizada actualmente y quedarse ahí hasta que se le ordene un cambio por medio del control remoto.

Note que el estado (canal sintonizado) al que pasará el sistema depende del estado actual (canal actual) y de las entradas (tecla del control remoto oprimida). Si las entradas no cambian el sistema no cambia su posición (esto no es necesariamente cierto para todas las máquinas de estado).

1.3. TABLAS Y DIAGRAMAS DE ESTADO

Existen varias formas de representar el comportamiento de las máquinas de estado, siendo los más utilizados las tablas y los diagramas de estado. Tomemos nuevamente el ejemplo del televisor y representemos en una tabla los estados actual y siguiente (Estado al que pasa el sistema ante un cambio de las entradas).

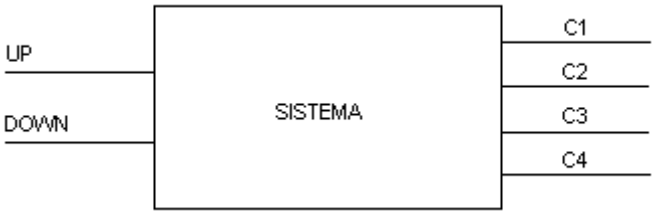

Figura 2. Entradas y salidas del Sistema.

La Figura 2 muestra el sistema como una caja negra en la que sólo se indican las entradas y las salidas. Supongamos que nuestro sistema tiene dos entradas que son las correspondientes a Adelantar (UP) y disminuir (DN) canal, y que tiene cuatro salidas C1, C2, C3, C4 que corresponden a los cuatro canales y que me indican cual canal se está sintonizando actualmente. En este punto debemos tomar varias decisiones:

¿Cuando se considera que una entrada esta activa?, Es decir, con que valor lógico de la entrada se produce un cambio y con cual no. En nuestro caso un valor lógico alto en las entradas producirá un cambio de estado, es decir, si UP = '1' el canal sintonizado aumentará o si DN = '1' el canal disminuirá. Otra decisión que debemos tomar y que se deriva de esta es: que sucede si por error las dos entradas son '1' simultáneamente, lo más conveniente es que el sistema conserve su estado ante esta posible situación.

El valor de las salidas para cada estado, en este ejemplo, un uno lógico en una salida indica que el canal ha sido seleccionado (por ejemplo, un uno en C1 indicara que actualmente el canal seleccionado es el 1), por lo tanto sólo una salida puede tener un valor lógico alto.

Una vez definido completamente el funcionamiento de nuestro sistema se procede a representarlo mediante una tabla de estados:

1.3.1. Tabla de Estados

Todas las condiciones posibles de las entradas

	Estado Siguiente (S*)			
Estado Actual (S)	UP =0, DN=0	UP=0,DN=1	UP=1,DN=0	UP=1,DN=1
Canal 1	Canal 1	Canal 4	Canal 2	Canal 1
Canal 2	Canal 2	Canal 1	Canal 3	Canal 2
Canal 3	Canal 3	Canal 2	Canal 4	Canal 3
Canal 4	Canal 4	Canal 3	Canal 1	Canal 4

Figura 3. Tabla de estados del sistema Televisor.

La Figura 3. Muestra una tabla de estados típica en la cual se resume el comportamiento del sistema. La tabla tiene tres secciones: El estado actual: Lista de todos lo posibles estados.

Posibles combinaciones de las entradas: El número de entradas del sistema determina el número de columnas de la tabla de estados. Así, si la máquina tiene n entradas, la tabla tendrá 2^n +1 Columnas.

El estado siguiente: Indica a que estado pasará la FSM cuando se presente una determinada entrada, Por ejemplo, Si UP=0 y DOWN = 1 y el estado actual es el canal 4 la máquina de estados irá al estado Canal 3.

Otra forma de representar el estado de las entradas es utilizando una convención en la que si la variable aparece negada entonces toma un valor de cero lógico, pero si no lo esta tiene un valor de uno lógico. Esto se hace para simplificar las tablas. Por ejemplo:

A: A = '1'

!A: A = 0;

Con lo que la tabla de estados se convierte en :

	Estado Siguiente				Salidas
Estado Actual	!UP.!DN	!UP.DN	UP.!DN	UP.DN	C1,C2,C3,C4
Canal 1	Canal 1	Canal 4	Canal 2	Canal 1	1,0,0,0
Canal 2	Canal 2	Canal 1	Canal 3	Canal 2	0,1,0,0
Canal 3	Canal 3	Canal 2	Canal 4	Canal 3	0,0,1,0
Canal 4	Canal 4	Canal 3	Canal 1	Canal 4	0,0,0,1

Tabla 1. Tabla de Estados del sistema Televisor.

1.3.2. Diagrama de Estados

Otra forma de representar el comportamiento de una máquina de estados es el diagrama de estados, este diagrama es una representación gráfica del funcionamiento del sistema.

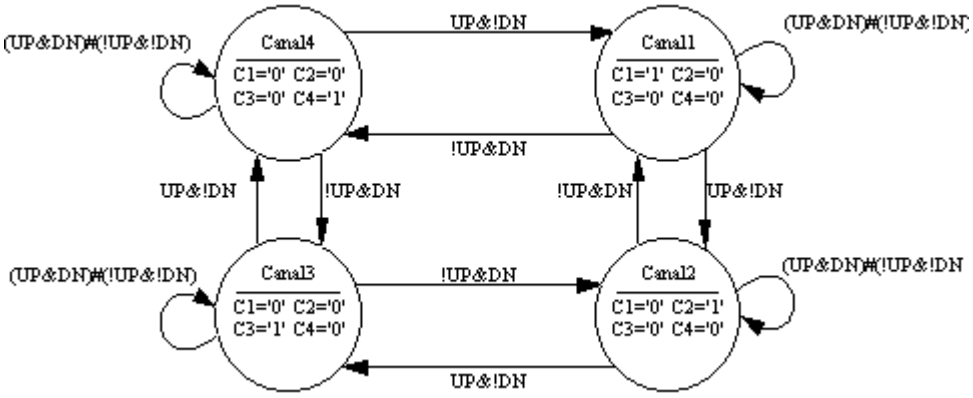

Figura 4. Diagrama de estados del sistema Televisor.

La Figura 4 muestra el diagrama de bloques del sistema televisor. Observamos que cada estado está representado por un círculo en el que se indica el nombre del mismo y el valor de las salidas. Las líneas que unen los estados representan el cambio que sufre la máquina cuando se cumple una determinada regla de transición. (La regla de transición normalmente se indican en las líneas que unen los estados). En esta figura se introduce una nueva nomenclatura para representar las funciones lógicas, el operador ***not*** se representa con el signo ***!***, la operación ***AND*** con el signo ***&*** y la ***OR*** con #.

Podemos observar que del estado Canal1, salen dos líneas: Una hacia el estado Canal2, lo que indica que la máquina pasará de Canal1 a Canal2 si UP = '1' y DN = '0' y se presenta un flanco adecuado en la señal de reloj; Otra hacia el estado Canal4, lo que indica que la máquina de estados pasará de Canal1 a Canal4 si UP = '0', DN = '1' y se presenta un flanco adecuado en la señal de reloj.

Así mismo tenemos dos líneas que llegan al Estado Canal1: Una proviene del estado Canal2, con lo que el sistema pasará de Canal2 a Canal 1 si UP = '0', DN = '1' y se presenta un flanco adecuado en la señal de reloj; y otra desde el estado Canal4. lo que hace que el sistema pase de Canal4 a Canal 1 si UP = '1' y DN = '0' y se presenta un flanco adecuado en la señal de reloj.

Por último existe una curva que sale del Canal1 y vuelve a entrar a Canal1, esto indica que la máquina mantendrá su estado si: (UP = '0' Y DN = '0') O (UP = '1' Y DN = '1').

1.4. SINTESIS DE FSM

En esta sección analizaremos la arquitectura de la FSM y el proceso de síntesis. Como vimos en el capítulo anterior la síntesis parte de una descripción funcional y llega a una descripción a nivel de compuertas.

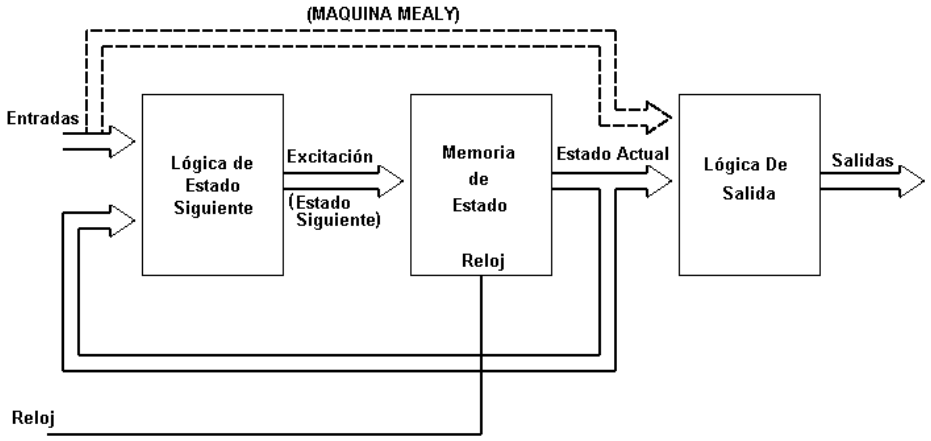

Figura 5. Estructura de una FSM.

1.4.1. Estructura de una FSM

La estructura general de una máquina de estados se muestra en la Figura 5. Como puede observarse existen tres bloques bien definidos:

Lógica de estado siguiente: Está encargada de generar las señales necesarias para producir un cambio de estado en la FSM (Estado Siguiente). Puede observarse que el estado siguiente depende del estado actual y de las entradas

Memoria de Estado: Normalmente esta formada por Flip-Flops tipo D o JK, los cuales tienen la misma señal de reloj. Las salidas de los Flip-Flops determinan el estado actual de la FSM, cada salida del Flip-Flop puede tomar dos valores distintos '1' o '0', por lo tanto se pueden tener dos estados con un Flip-Flop. Si tenemos N Flip-Flops tendremos 2^N estados.

Lógica de Salida: La lógica de salida esta encargada de producir los valores necesarios a la salida de la FSM, su la arquitectura esta basada en decodificadores.

Existen dos tipos de máquinas de estados: Moore: El estado siguiente depende del valor de las entradas y del estado actual; Y la salida depende únicamente del estado actual; y Mealy: Al igual que la máquina de Moore el estado siguiente depende del valor de las entradas y del estado actual, pero se diferencian en que la salida depende del estado actual y del valor de las entradas.

1.4.2. Proceso de Síntesis

El primer paso en el proceso de síntesis de una FSM y en general de cualquier sistema digital es la descripción del sistema ya sea mediante un algoritmo o de un diagrama de tiempos. El siguiente paso consiste en pasar la descripción funcional a un diagrama de estados para su posterior representación en tablas de estado y de salida. A continuación debemos reducir el número de estados (si es posible) utilizando algoritmos de minimización. Después debemos realizar la codificación de estados, es decir, asignarle a los estados un grupo único de valores que corresponden a los valores que tomarán los Flip-Flops. A continuación se debemos obtener las ecuaciones de estado siguiente y de salidas. El siguiente paso consiste en la elección del tipo de Flip-Flop que se va a utilizar para la implementación, recuerde que todos los Flip_Flops tienen diferentes ecuaciones de estado siguiente:

Tipo de F-F	Estado Siguiente (Q^*)
D	Q* = D
JK	Q* = J,!Q + !K.Q
T	Q* = !Q
SR	Q* = S + !R.Q

Tabla 2. Ecuaciones de Estado siguiente de los Flip-Flop.

Una vez elegido el Flip-Flop se procede a obtener las ecuaciones de las señales de excitación de los FFs. Después se de be realizar un proceso de minimización con las ecuaciones obtenidas anteriormente para realizar un diagrama de la implementación lógica de las ecuaciones. Finalmente debemos verificar el correcto funcionamiento del circuito, esto se logra simulando el circuito obtenido y si cumple con lo requerimientos se llevará al plano físico para realizar pruebas de tiempos. La figura 6 resume los pasos a seguir en el proceso de síntesis.

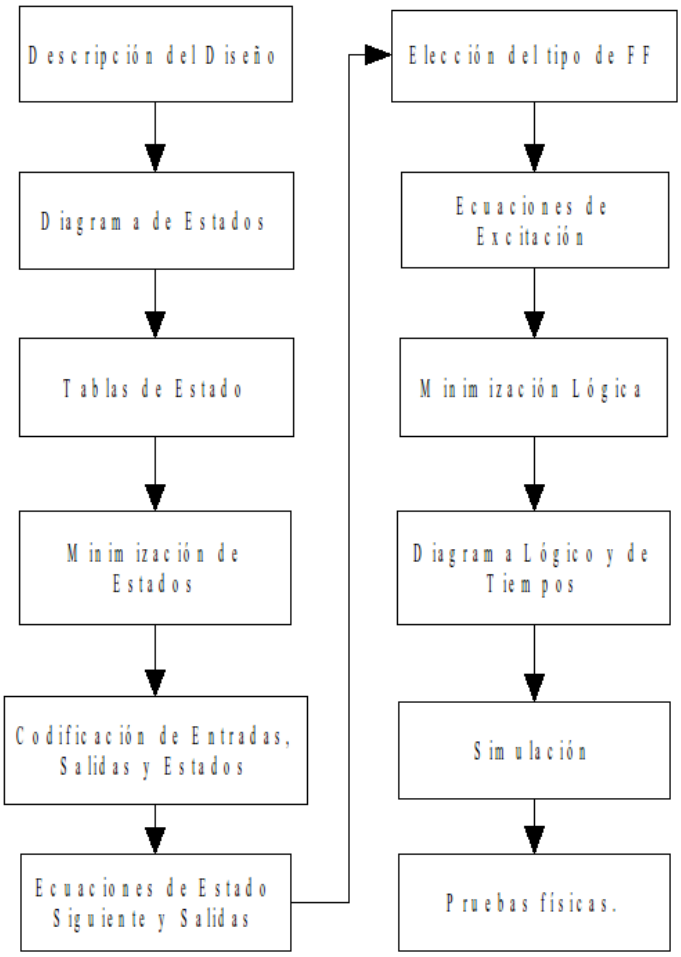

Figura 6. Diagrama de flujo del proceso de Síntesis para FSM

Para entender mejor el proceso de síntesis realizaremos paso a paso un sencillo ejemplo. Desarrollando todas las posibilidades de implementación para buscar la más óptima.

1.4.3. Control de Motor de Paso.

Un motor de paso a diferencia de los motores de Corriente Continua necesita una secuencia determinada en sus cuatro terminales para originar el giro de su rotor. La secuencia necesaria para controlar el motor es la siguiente:

	A	B	C	D
S1	V+	GND	V+	GND
S2	V+	GND	GND	V+
S3	GND	V+	GND	V+
S4	GND	V+	V+	GND

Para que el motor gire un paso (normalmente 1.8 grados) es necesario que se aplique S1 y luego S2, o S2 y S3 o S3 y S4 o S4 y S1. Si se desea que el motor gire cinco pasos se debe seguir la secuencia S1, S2, S3, S4, S5. La inversión del sentido de giro se logra invirtiendo la secuencia anterior, es decir, S1, S4, S3, S2, S1.

Diagrama de Caja Negra El primer paso consiste en realizar un diagrama de caja negra en el que se indiquen las entradas y salidas (Figura 7).

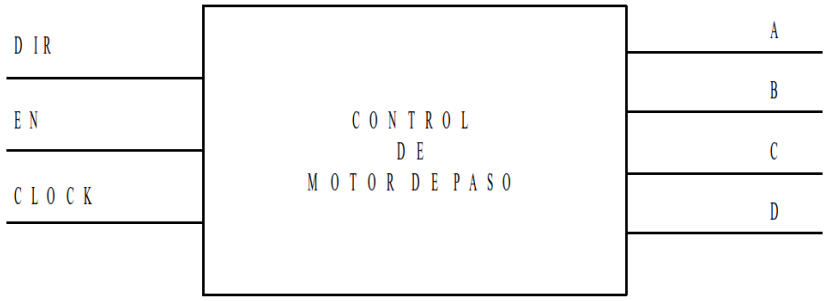

Figura 7. Diagrama de Caja Negra del Controlador de Motor de Paso.

A continuación realizaremos una breve descripción del funcionamiento del circuito. Como se observa en la Figura 7. Se tienen tres entradas:

DIR: Encargada de indicar la dirección de giro del motor. DIR = 0 Secuencia Directa (S1, S2, S3, S4), DIR = 1 Secuencia Invertida (S4, S3, S2, S1) EN: ENABLE, encargada de habilitar nuestro control Si EN = 1 el circuito realizará su función si EN = 0 el control conservará el último estado de las salidas.

CLOCK: Es el reloj del sistema y gobierna todas las transiciones entre estados. Y las cuatro salidas A, B, C, D son las encargadas de manejar los terminales del motor de paso.

Diagrama de Estado Una vez se conoce el funcionamiento del circuito, o las funciones que debe cumplir se procede a realizar el diagrama de estados del mismo. La Figura 8 muestra este diagrama para el controlador de motor de paso.

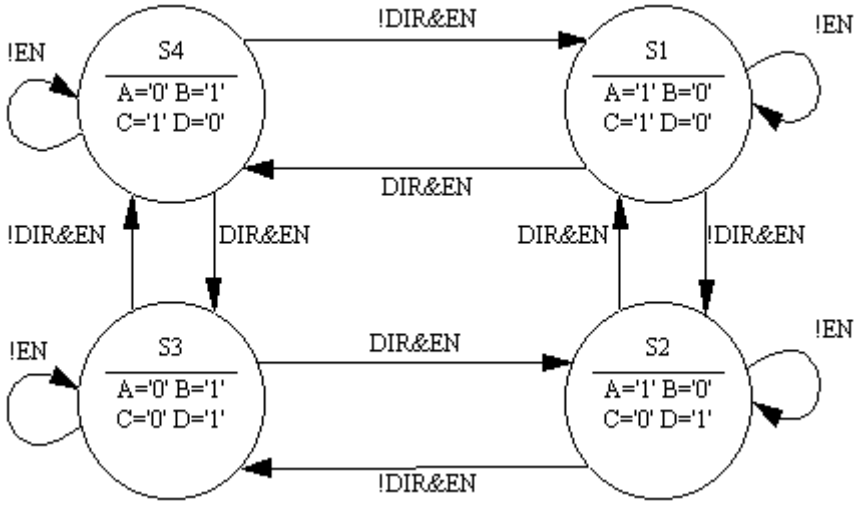

Figura 8 Diagrama de Estados del Controlador de Motor de Paso.

Tabla de estados Una vez realizado el diagrama de estados se procede a la realización de la tabla de estados y salidas. Dicha tabla para el controlador de motor de paso se presente a continuación:

	Estado Siguiente				Salidas
Estado Actual	!EN.!DIR	!EN.DIR	EN.!DIR	EN.DIR	A, B, C, D
S1	S1	S1	S2	S4	1, 0, 1, 0
S2	S2	S2	S3	S1	1, 0, 0, 1
S3	S3	S3	S4	S2	0, 1, 0, 1
S4	S4	S4	S1	S3	0, 1, 1, 0

Minimización de Estados El objetivo de la minimización de estados es la reducción del número de Flip-Flops necesarios para la implementación, reduciendo de este modo el costo de la misma. Sin embargo para reducir el número de Flip-Flops de un circuito es necesario reducir el número de

estados en múltiplos de 2. Por ejemplo, supongamos que tenemos 7 estados, para lo cual necesitamos 3 FFs, para utilizar sólo 2 FFs necesitamos reducir el número de estados de 7 a 4 o menos.

La minimización de estados se basa en la equivalencia funcional, por ejemplo, se dice que dos circuitos son equivalentes cuando proporcionan la misma salida ante los mismos cambios de entrada. Dos o más estados son equivalentes sí:

Ambos estados producen las mismas salidas ante igual cambio en las señales de entrada. Ambos estados tienen los mismos estados siguientes ante los mismos cambios de las señales de entrada.

En nuestro caso no tenemos equivalentes ya que todos tienen diferentes valores de salida y diferentes estados siguientes ante variaciones de las entradas.

Codificación de estados La codificación de estados consiste en la asignación de valores a las salidas de los FFs para cada estado, estos valores deben ser únicos para cada estado, es decir, no se deben repetir. Debido a que nuestra máquina tiene cuatro estados, tenemos 2 FFs y por lo tanto debemos asignar a cada estado un código formado por dos bits. Existen muchas posibles codificaciones para los cuatro estados, unas más eficientes que otras. En este libro no se tratarán las técnicas existentes para hallar la codificación óptima ya que esta función las realizan las herramientas CAD y además, se sale del objetivo de este libro. Para nuestro ejemplo utilizaremos la siguiente codificación:

	Q1	Q0
S1	0	0
S2	0	1
S3	1	0
S4	1	1

Donde Q1 y Q0 son las salidas del primer y segundo FF correspondientemente. Con esta asignación de estados la tabla de estados queda de la siguiente forma:

Estado Actual (S)		Estado Siguiente (S^*)						SALIDAS
		!EN		EN				
				!DIR		DIR		A, B, C, D
Q1	Q0	$Q1^*$	$Q0^*$	$Q1^*$	$Q0^*$	$Q1^*$	$Q0^*$	
0	0	0	0	0	1	1	1	1, 0, 1, 0
0	1	0	1	1	0	0	0	1, 0, 0, 1
1	0	1	0	1	1	0	1	0, 1, 0, 1
1	1	1	1	0	0	1	0	0, 1, 1, 0

Donde $Q1^*$ y $Q0^*$ representan los valores siguientes de las señales Q1 y Q0 respectivamente. Note que no se representaron los casos !EN.!DIR y !EN.DIR, esto se debe a que cuando la señal En tiene un valor lógico bajo la FSM conserva su estado sin importar el valor de DIR.

Ecuaciones de estado siguiente Una vez realizada la codificación de estados se procede a la obtención de las ecuaciones de estado siguiente $Q1^*$ y $Q0^*$. Para obtener estas ecuaciones debemos sumar todos los unos de la tabla de estados (suma de minitérminos). Para $Q1^*$ debemos observar todas las columnas correspondientes a $Q1^*$ y sumar los minitérminos asociados. Por ejemplo, la primera columna de Q1* correspondiente a la asociada con !EN, el primer minitérmino asociado es: !EN.Q1.!Q0 ya que está en la fila correspondiente a Q1 = 1 y Q0 = 0. Y el segundo !EN.Q1.Q0.

$Q1^*$ = !EN.Q1.!Q0 + !EN.Q1.Q0 + EN.!DIR.!Q1.Q0 + EN.!DIR.Q1.!Q0 + EN.DIR.!Q1.!Q0 + EN.DIR.Q1.Q0.

$Q0^*$ = !EN.!Q1.Q0 + !EN.Q1.Q0 + EN.!DIR.!Q1.!Q0 + EN.!DIR.Q1.!Q0 + EN.DIR.!Q1.!Q0 + EN.DIR.Q1.!Q0.

Estas ecuaciones deben pasar a través de un proceso de minimización para encontrar una implementación óptima.

Q1* = !EN.Q1(!Q0+Q1) + EN.(!DIR.(!Q1.Q0 + Q1.!Q0) + DIR (!Q1.!Q0 + Q1.Q0))
Q1* = !EN.Q1 + EN.(!DIR.Q1 Q0 + DIR.!(Q1 Q0))
Q1* = !EN.Q1 + EN.DIR (Q1 XOR Q0)
Q0* = !EN.Q0.(!Q1 + Q1) + EN.(!Q1.!Q0.(!DIR + DIR) + Q1.!Q0.(!DIR + DIR))
Q0* = !EN.Q0 + EN.(!Q1.!Q0 + Q1.!Q0)
Q0* = !EN.Q0 + EN.(!Q0.(!Q1+ Q1))
Q0* = !EN.Q0 + EN.!Q0
Q0* = EN XOR Q0
Las ecuaciones para las salidas son:
A = !Q1.!Q0 + !Q1.Q0 = !Q1 B = Q1.!Q0 + Q1.Q0 = Q1 C = !Q1.!Q0 + Q1.Q0 = !(Q1 Q0) D = !Q1.Q0 + Q1.!Q0 = Q1 Q0 A = !B = !Q1 C = !D = Q1 XOR Q0

Elección del tipo de Flip-Flop, ecuaciones de excitación y minimización Utilizando las ecuaciones obtenidas anteriormente para Q1* y Q0*, debemos escoger el tipo de Flip-Flop a utilizar. La siguiente tabla resume los valores necesarios en las entradas de los FFs para obtener los cambios indicados en sus salidas. Por ejemplo en un FF JK si Q tiene un valor de 0 lógico y se quiere que pase a tener un valor de 1 lógico es necesario que J = 1 y el valor de K no importa.

Q Q*	S R	J K	T	D
0 0	0 X	0 X	0	0
0 1	1 0	1 X	1	1
1 0	0 1	X 1	1	0
1 1	X 0	X 0	0	1

El diagrama de Karnough para la máquina de estados es el siguiente

	00	01	11	10
00	**0** 0	**0** 0	**1** 1	**0** 1
01	**0** 1	**0** 1	**0** 0	**1** 0
11	**1** 1	**1** 1	**1** 0	**0** 0
10	**1** 0	**1** 0	**0** 1	**1** 1

La región sombreada corresponde a los valores de las señales Q1* (en negrilla) y Q0*.
Para el FF D tenemos: Q* = D. Por lo tanto:
D1 = !EN.Q1 + EN.DIR (Q1 XOR Q0) D0 = EN XOR Q0
Para el FF JK debemos ordenar las ecuaciones obtenidas de la forma: Q* =J.!Q + !K.Q, por lo que para Q1:

	00	01	11	10
00	**0** X	**0** X	**1** X	**0** X
01	**0** X	**0** X	**0** X	**1** X
11	**X** 0	**X** 0	**X** 0	**X** 1
10	**X** 0	**X** 0	**X** 1	**X** 0

La región sombreada indican los valores que deben tener las señales J1(en negrilla) y K1. J1 = !Q1.!Q0.EN.DIR + !Q1.Q0.EN.!DIR = EN.!Q1(Q0 XOR DIR) K1 = Q1.!Q0.EN.DIR + Q1.Q0.EN.!DIR = EN.Q1.(Q0 XOR DIR)
Para Q0

	00	01	11	10
00	**0** X	**0** X	**1** X	**1** X
01	**X** 0	**X** 0	**X** 1	**X** 1

11	**X** 0	**X** 0	**X** 1	**X** 1
10	**0** X	**0** X	**1** X	**1** X

J0 = EN K0 = EN
Flip-Flop tipo T:
Para Q1:

	00	01	11	10
00	0	0	1	0
01	0	0	0	1
11	0	0	0	1
10	0	0	1	0

T1 = !Q0.EN.DIR + Q0.EN.!DIR = EN.(Q0 XOR DIR)
Para Q0:

	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

T0 = EN
Flip-Flop tipo SR.
Para Q1:

	00	01	11	10
00	**0** X	**0** X	**1** 0	**0** X
01	**0** X	**0** X	**0** X	**1** 0
11	**X** 0	**X** 0	**X** 0	**0** 1
10	**X** 0	**X** 0	**0** 1	**X** 0

S1 = !Q1.!Q0.EN.DIR + !Q1.Q0.EN.!DIR = EN.!Q1(Q0 XOR DIR)
R1 = Q1.!Q0.EN.DIR + Q1.Q0.EN.!DIR = EN.Q1.(Q0 XOR DIR)
Para Q0:

	00	01	11	10
00	**0** X	**0** X	**1** 0	**1** 0
01	**X** 0	**X** 0	**0** 1	**0** 1
11	**X** 0	**X** 0	**0** 1	**0** 1
10	**0** X	**0** X	**1** 0	**1** 0

S0 = EN.!Q0 R0 = EN.Q0

Del análisis anterior observamos que la implementación que ocupa el menor número de compuertas es la correspondiente a los Flip-Flops tipo T.

SIMULACION En la Figura 9 se muestra la implementación final de la máquina de Estados (Sin la Unidad de Salida) y en la Figura 10 la simulación correspondiente.

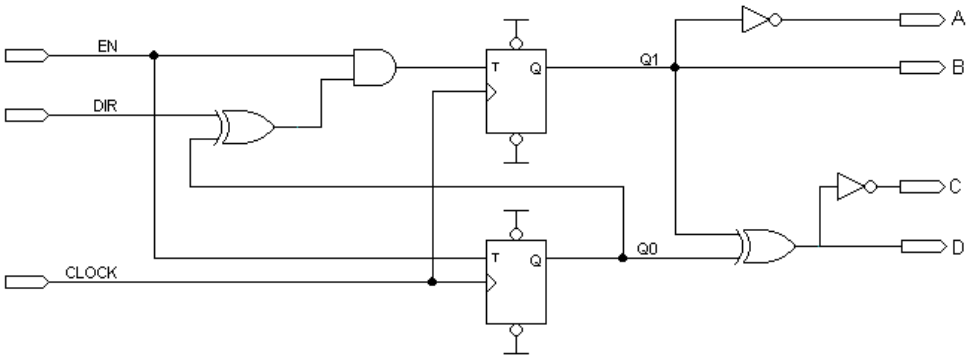

Figura 9. Diagrama a nivel de compuertas del controlador de Motor de Paso.

Como puede observarse en la Figura 9, la máquina de estados está formada por la lógica de estado siguiente (Compuertas XOR y AND), la memoria de estado (FFs tipo T con la misma señal de Reloj) y la lógica de salida, (Compuertas NOT y XOR) que en este caso corresponde al de una máquina de estados tipo MOORE.

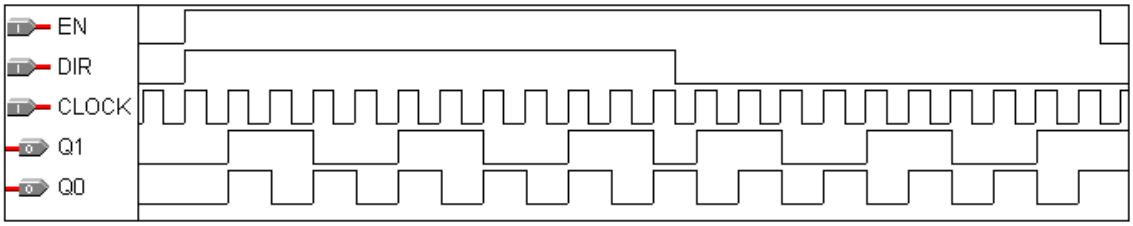

Figura 10. Simulación del Controlador de Motor de Paso.

1.5. DESCRIPCION VHDL DE UNA FSM

En esta sección se realizará la descripción en VHDL del controlador de motor de paso. Inicialmente se hará una descripción estructural, basándose en los resultados obtenidos en la sección 5.4.3.7. Y a continuación se utilizará la descripción funcional para la implementación del controlador.

1.5.1. DESCRIPCION ESTRUCTURAL

Recordemos que la descripción estructural en VHDL realiza la interconexión entre los componentes que forman el sistema. En nuestro caso (Ver Figura 9), primero debemos realizar la descripción de los componentes:

```
ENTITY and2 IS -- Debido a que AND es una palabra reservada del lenguaje
-- no puede utilizarse para nombrar una entidad
PORT(
     A : IN BIT;
     B : IN BIT;
     Y : OUT BIT);
     END and2;
ARCHITECTURE Y OF and2 IS
BEGIN
     Y <= A AND B;
END;
```

```
ENTITY xor2 IS -- Debido a que XOR es una palabra reservada del lenguaje
-- no puede utilizarse para nombrar una entidad
  PORT(
```

```
    A : IN BIT;
    B : IN BIT;
    Y : OUT BIT);
END xor2;

ARCHITECTURE XR OF xor2 IS
  BEGIN
    Y <= A XOR B;
  END;
```

```
INVERSOR ENTITY inv IS
  PORT(
    A : IN BIT;
    Y : OUT BIT);
END inv;
ARCHITECTURE NO OF inv IS
  BEGIN
    Y <= not (A);
  END;
```

```
FLIP FLOP TIPO T ENTITY FFT IS
  PORT(
    T,CLK : IN BIT;
    Q : BUFFER BIT);
END FFT;
ARCHITECTURE T OF FFT IIS
  BEGIN
   process (CLK) begin
    if (CLK'event and CLK = '1') then
     if (t = '1') then
      Q <= not (Q);
     else
      Q <= Q;
     end if;
    end if;
   end process;
  END;
```

Una vez realizada la implementación de los componentes, se procede a su unión. Para lograr esto se deben sintetizar las anteriores declaraciones, y sus archivos deben estar en el mismo directorio de trabajo.

```
CONTROL DE MOTOR DE PASO ENTITY StepMotor IS
  PORT(
    CLK, EN, DIR : IN BIT;
    AO, BO, CO, DO : BUFFER BIT);
END StepMotor;

ARCHITECTURE CONTROLLER OF StepMotor IS
  SIGNAL I1, T1, Q0N: BIT;
  COMPONENT AND2E
    PORT (A, B : IN BIT;
```

```
            Y : OUT BIT);
  END COMPONENT;
  COMPONENT XOR2
    PORT (A, B : IN BIT;
            Y : OUT BIT);
  END COMPONENT;
  COMPONENT INV
    PORT (A : IN BIT;
          Y : OUT BIT);
  END COMPONENT;
  COMPONENT FFT
    PORT(
    CLK, T: IN BIT;
        Q : OUT BIT);
  END COMPONENT;

  BEGIN
    X1: XOR2 port map(DIR, Q0N, I1);
    X2: AND2E port map(EN, I1, T1);
    X3: FFT port map(CLK, T1, BO);
    X4: FFT port map(CLK, EN, Q0N);
    X5: XOR2 port map(BO, Q0N, DO);
    X6: INV port map(BO, AO);
    X7: INV port map(DO, CO);
  END CONTROLLER;
```

1.5.2. DESCRIPCION FUNCIONAL

La descripción Estructural no es el mejor ejemplo de la potencialidad del VHDL, ya que como vimos, es necesario realizar todo el proceso de síntesis para obtener las ecuaciones de excitación de los Flip-Flops. Realizando la descripción funcional no es necesario preocuparse por escoger el Flip – Flop que reduzca el número de compuertas ni de minimizar las ecuaciones booleanas obtenidas, todos estos procesos los realizan automáticamente las herramientas CAD disponibles comercialmente. El código VHDL del controlador del motor de paso se muestra a continuación, y en la Figura 11 se muestra el resultado de la simulación.

```
library ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY FSMF IS
  PORT(
    clk          : IN std_logic;
    EN, DIR      : IN std_logic;
    A, B, C, D : OUT std_logic);
END FSMF;

ARCHITECTURE funcional OF FSMF IS
TYPE estados IS (S1, S2, S3, S4);
SIGNAL state : estados;

BEGIN
  PROCESS (clk)
   BEGIN
    IF (clk'event and clk = '1') then
     case state is
      when S1 => A <= '1'; B <= '0'; C <= '1'; D <= '0';
       IF (EN = '0') then
```

```
          state <= S1;
         ELSIF (DIR = '0') then
          state <= S2;
         ELSE
          state <= S4;
         END IF;

        when S2 => A <= '1'; B <= '0'; C <= '0'; D <= '1';
         IF (EN = '0') then
          state <= S2;
         ELSIF (DIR = '0') then
          state <= S3;
         ELSE
          state <= S1;
         END IF;

        when S3 => A <= '0'; B <= '1'; C <= '0'; D <= '1';
         IF (EN = '0') then
          state <= S3;
         ELSIF (DIR = '0') then
          state <= S4;
         ELSE
          state <= S2;
         END IF;
        when S4 => A <= '0'; B <= '1'; C <= '1'; D <= '0';
         IF (EN = '0') then
          state <= S4;
         ELSIF (DIR = '0') then
          state <= S1;
         ELSE
          state <= S3;
        END IF;
       END CASE;
      END IF;
   END PROCESS;
END funcional;
```

El código para la realización del Test Bench es:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY testbench IS
END testbench;
ARCHITECTURE behavior OF testbench IS
  COMPONENT fsmf
  PORT(
    clk : IN std_logic;
    EN : IN std_logic;
    DIR : IN std_logic;
    A : OUT std_logic;
    B : OUT std_logic;
    C : OUT std_logic;
    D : OUT std_logic
    );
  END COMPONENT;
  SIGNAL clk : std_logic;
  SIGNAL EN : std_logic;
  SIGNAL DIR : std_logic;
  SIGNAL A : std_logic;
```

```
  SIGNAL B : std_logic;
  SIGNAL C : std_logic;
  SIGNAL D : std_logic;
constant ncycles : integer := 40;
constant halfperiod : time  := 5 ns;
BEGIN
  uut: fsmf PORT MAP(
    clk => clk,
    EN => EN,
    DIR => DIR,
    A => A,
    B => B,
    C => C,
    D => D);
-- Generacion del Reloj
Clock_Source: process
begin
for i in 0 to ncycles loop -- Genera ncyclos de periodo 10 ns
clk <= '0';
wait for halfperiod;
clk <= '1';
wait for halfperiod;
end loop;
wait;
end process Clock_Source;
tb : PROCESS
  BEGIN
for i in 1 to ncycles/4 loop -- Durante ncyclos/4 genera las diferentes
wait until Clk='1' and Clk'event; -- Combinaciones de EN y DIR
EN <= '0';
DIR <= '0';
end loop;
for i in 1 to ncycles/4 loop
wait until Clk='1' and Clk'event;
EN <= '0';
  DIR <= '1';
end loop;
for i in 1 to ncycles/4 loop
wait until Clk='1' and Clk'event;
EN <= '1';
  DIR <= '0';
end loop;
for i in 1 to ncycles/4 loop
wait until Clk='1' and Clk'event;
EN <= '1';
  DIR <= '1';
end loop;
END PROCESS;
END;
```

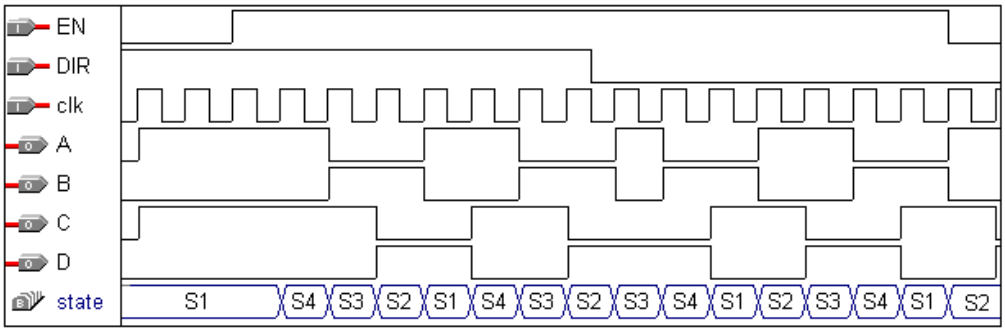

Figura 11. Simulación de la descripción funcional del controlador de motor de paso.

Con la descripción funcional no es necesario preocuparse por el tipo de Flip-Flop ni por los procesos de minimización.

Normalmente las herramientas CAD proporcionan información sobre resultado de ls síntesis del diseño, las ecuaciones implementadas en un CPLD por una de estas herramientas es:

clk : INPUT;
DIR : INPUT;
EN : INPUT;
A = DFFE(state˜2 $ VCC, GLOBAL(clk), VCC, VCC, VCC);
B = DFFE(!state˜2 $ VCC, GLOBAL(clk), VCC, VCC, VCC);
C = DFFE(!state˜1 $ state˜2, GLOBAL(clk), VCC, VCC, VCC);
D = DFFE(state˜1 $ state˜2, GLOBAL(clk), VCC, VCC, VCC);
state˜1 = TFFE(EN, GLOBAL(clk), VCC, VCC, VCC);
state˜2 = TFFE(_EQ001, GLOBAL(clk), VCC, VCC, VCC);
_EQ001 = !DIR & EN & state˜1 # DIR & EN & !state˜1;

En donde se puede observar que se llegó a las mismas ecuaciones obtenidas anteriormente. Observe que se utilizaron 2 Flip-Flops tipo T uno de los cuales tiene a la señal EN como entrada (primera seña que aparece dentro de los paréntesis después de TFFE.). Mientras que el otro tiene a la señal _EQ001 conectada a su entrada T. La ecuación para _EQ001 es:

!DIR & EN & state˜1 # DIR & EN & !state˜1 = EN.((!DIR.sate˜1) + (DIR.!state˜1)

Como vimos en capítulos anteriores la arquitectura de un CPLD no posee compuertas XOR, por lo tanto las ecuaciones son de la forma de suma de productos. Lo interesante es observar el ahorro de tiempo asociado a la utilización del VHDL como herramienta de diseño.

2. Máquinas de Estados Algorítmicas (ASM)

Una máquina de estados algorítmica (ASM) permite la implementación de cualquier tipo de algoritmo; está compuesta por el *camino de datos* y el *control*. El camino de datos, como su nombre lo indica, modifica los datos o variables del algorítmo; mientras que el control determina cuando son modificados. Los pasos que se realizan para el diseño e implementación de una máquina de estados algorítmica son los siguientes:

1. Elaboración del diagrama de flujo del algoritmo.
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

2.1. Multiplicación de números binarios usando una ASM

El algoritmo de multiplicación que se implementará se basa en productos parciales; el primer producto parcial siempre es cero (ver Figura 1, 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 dos 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.

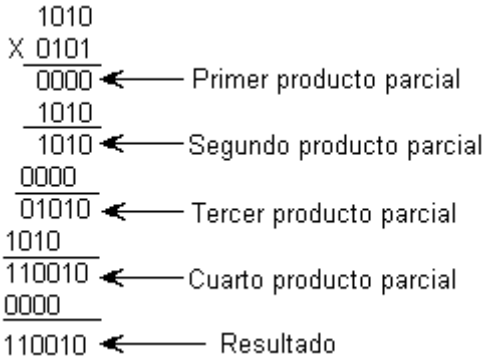

Figura 1: Multiplicación de numeros binarios usando productos parciales.

2.1.1. Diagrama de Flujo

En la Figura 2 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.

Una vez conocido el funcionamiento del sistema se procede a realizar el diagrama de caja negra de entradas y salidas. En la Figura 4 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

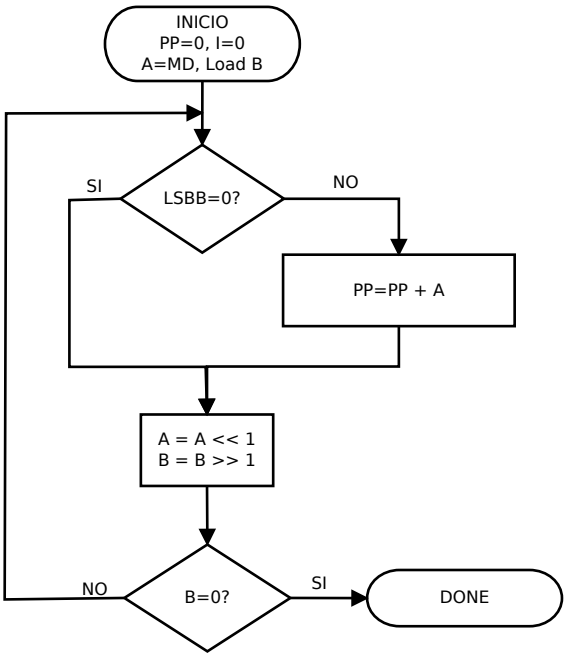

Figura 2: Diagrama de flujo para la multiplicación de numeros binarios.

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.

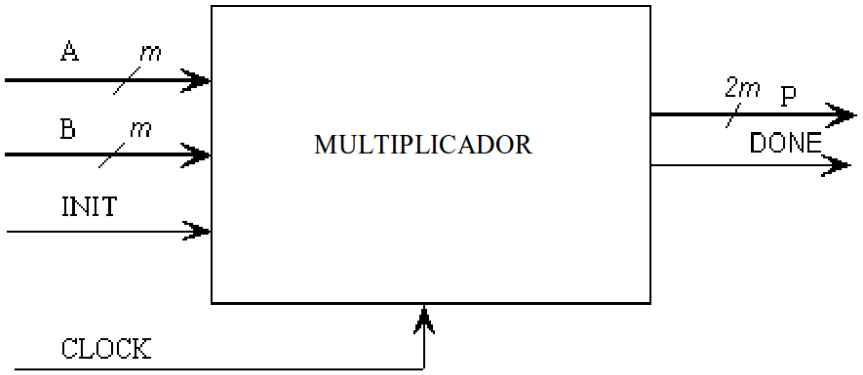

Figura 3: Diagrama de caja negra para el multiplicador de numeros binarios.

2.1.2. 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.

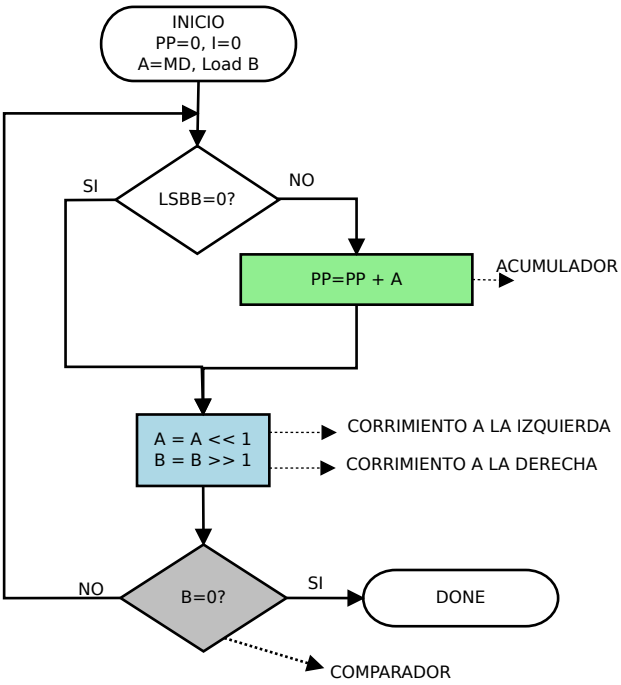

Figura 4: Identificación de los componentes del camino de datos para el multiplicador de numeros binarios.

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

En la figura 5 se muestra la interconexión de los componentes del camino de datos y las señales que lo controlan.

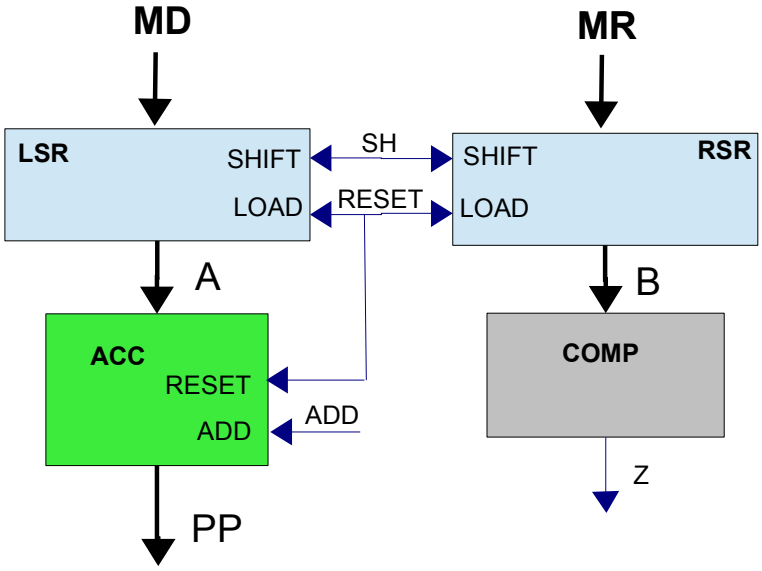

Figura 5: Identificación de las señales de control e interconexión del camino de datos.

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.

2.1.4. 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 6 se muestra la relación entre el diagrama de flujo y el diagrama de estados.

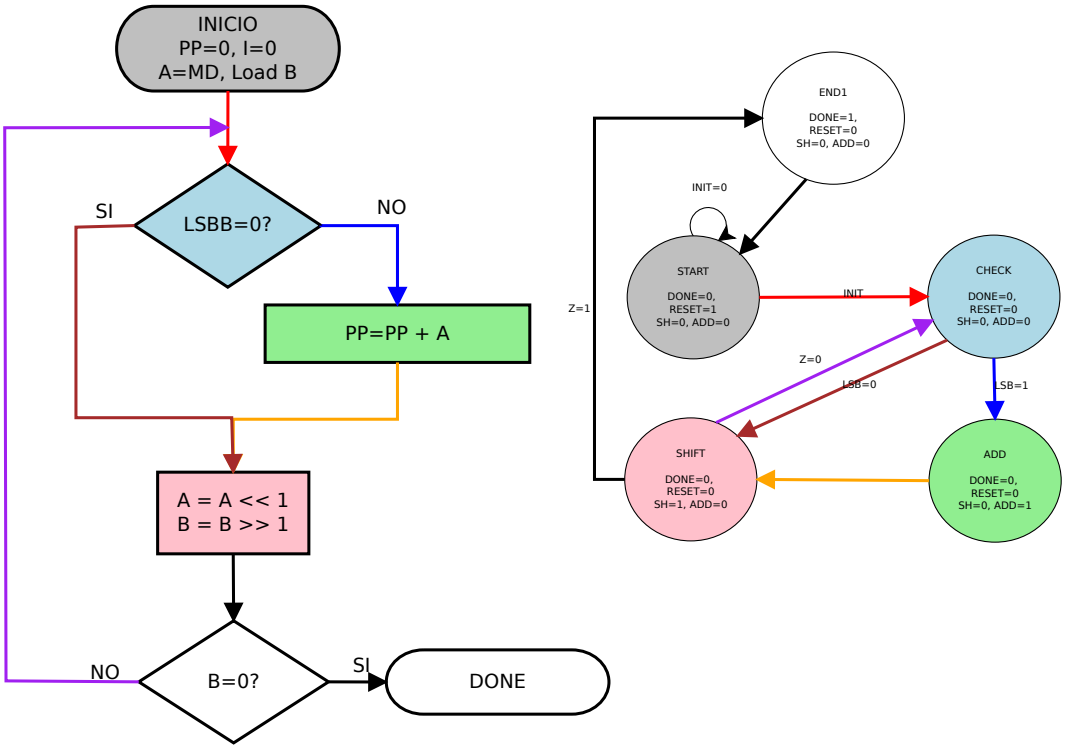

Figura 6: Diagrama de estados de la unidad de control del multiplicador binario.

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 6

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:

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	0	00000111
CHECK	0	00001110	0010	0	0	0	0	00000111
SHIFT	1	00011100	0001	0	1	0	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	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.

2.1.5. 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 estándard del lenguaje utilizad; por esto, se recomienda utilizar únicamente entrada de texto en las descripciones.

2.1.6. 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 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*.

2.1.7. 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.

2.2. 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 7 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.

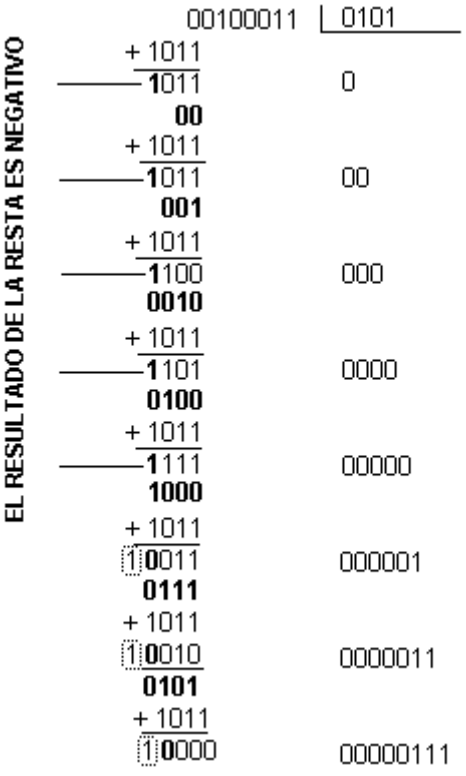

Figura 7: División de numeros binarios.

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 7), 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

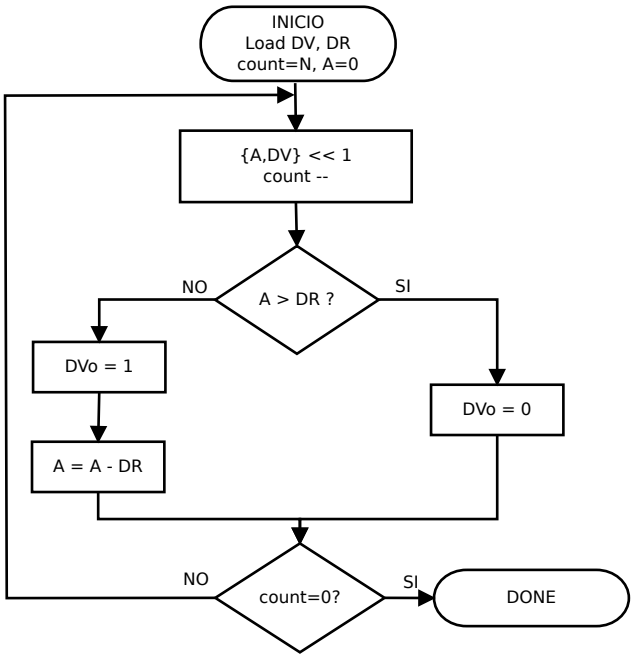

Figura 8: Algoritmo para la división de numeros binarios.

2.2.1. Identificación de componentes del camino de datos e interconexión

En la figura 9 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 10 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.

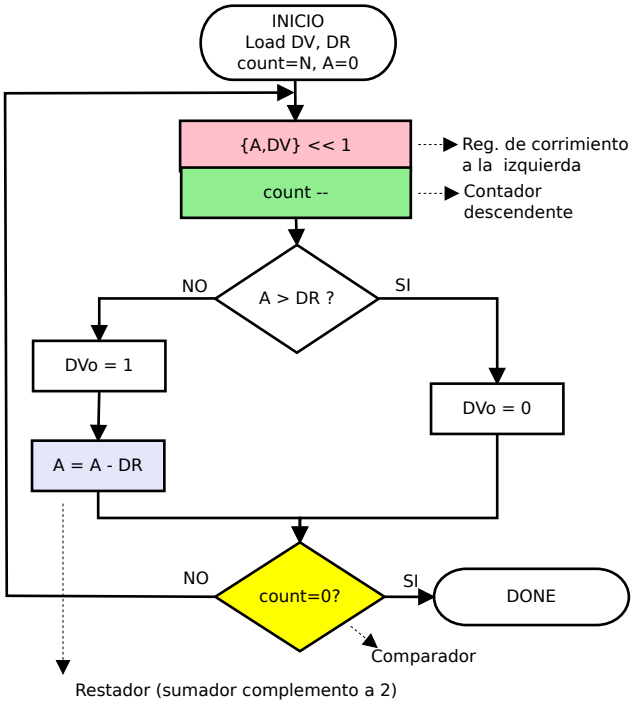

Figura 9: Identificación de componentes del camino de datos para la división de numeros binarios.

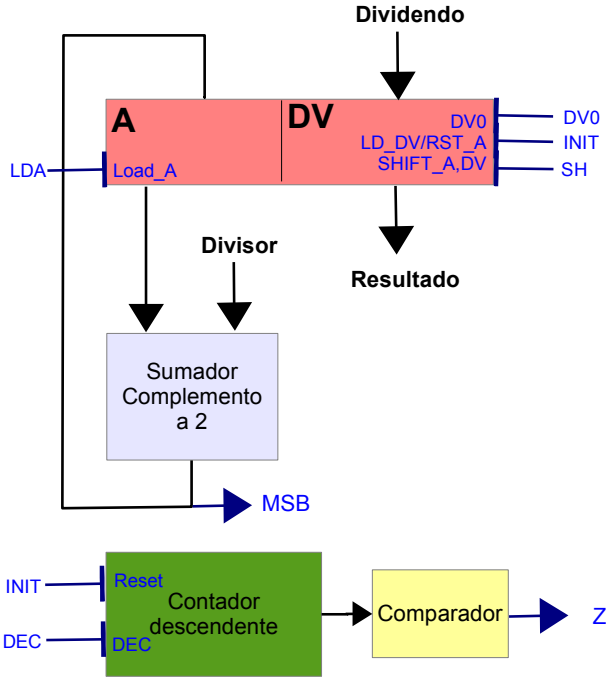

Figura 10: Interconexión del camino de datos para la división de numeros binarios.

2.2.2. Unidad de control

En la Figura 11 se muestra la relación existente entre el diagrama de flujo y el diagrama de estados de la unidad de control del divisor binario.

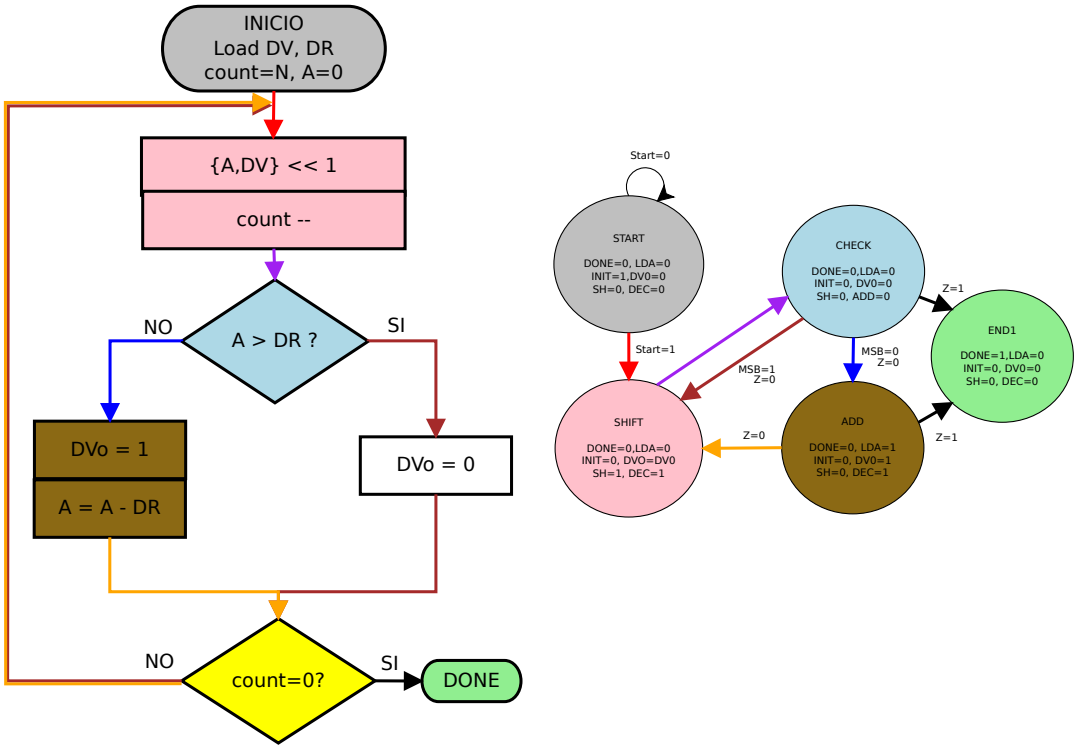

Figura 11: Diagrama de estados de la unidad de control para la división de numeros binarios.

2.3. Contador de número de unos

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

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