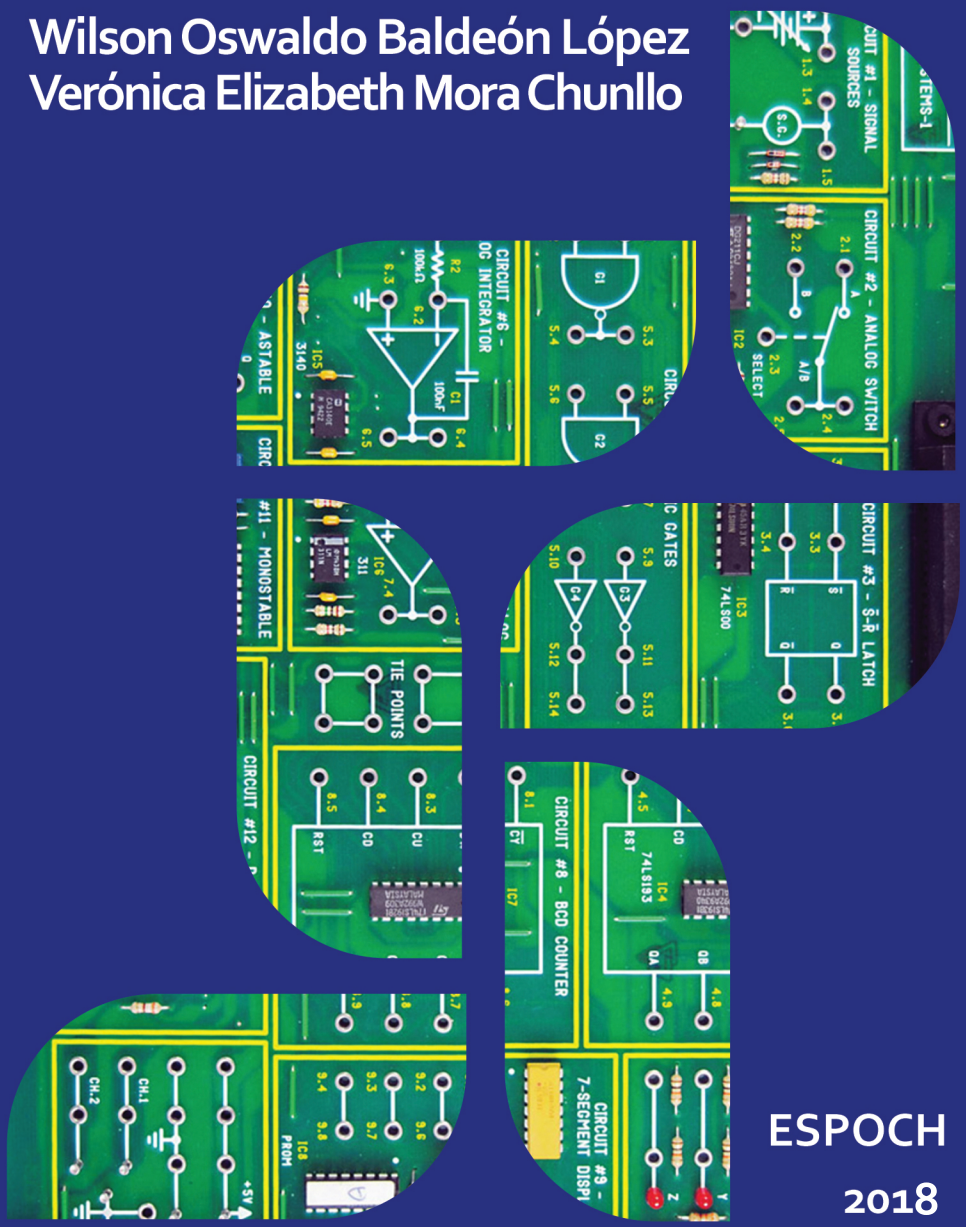
Sistemas digitales sincrónicos y VHDL

Diseño de máquinas de estado algorítmico con VHDL

Wilson Oswaldo Baldeón López
Verónica Elizabeth Mora Chunllo



ESPOCH
2018

SISTEMAS DIGITALES SINCRÓNICOS Y VHDL
DISEÑO DE MÁQUINAS DE ESTADO ALGORÍTMICAS CON VHDL

SISTEMAS DIGITALES SINCRÓNICOS Y VHDL
DISEÑO DE MÁQUINAS DE ESTADO ALGORÍTMICAS CON VHDL

WILSON BALDEÓN
VERÓNICA MORA

DIRECCIÓN DE PUBLICACIONES

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

SISTEMAS DIGITALES SINCRÓNICOS Y VHDL
DISEÑO DE MÁQUINAS DE ESTADO ALGORÍTMICAS CON VHDL

Panamericana Sur, kilómetro 1 ½
Dirección de Publicaciones Científicas
Riobamba, Ecuador
Teléfono: (593 03) 299 8200
Código postal: EC0600155

Aval ESPOCH

Este libro se sometió a arbitraje bajo el sistema de doble ciego

(*peer review*)

Corrección y diseño

La Caracola Editores

Editorial Politécnica ESPOCH

Impreso en Ecuador

CDU: 004.3 + 004.4
SISTEMAS DIGITALES SINCRÓNICOS Y VHDL
DISEÑO DE MÁQUINAS DE ESTADO ALGORÍTMICAS CON VHDL
Riobamba: Escuela Superior Politécnica de Chimborazo
Dirección de Publicaciones, 2018
86 pp. vol: 17 x 24 cm
ISBN: 978-9942-35-651-2
1. Ciencia y tecnología de los ordenadores
2. *Hardware*. Componentes físicos del ordenador.
3. *Software*. Equipo lógico, componentes lógicos.

A Solange Baldeón

A Efraín Baldeón

A Sean Risley

ÍNDICE

ACERCA DE LOS AUTORES

Wilson Oswaldo Baldeón López es ingeniero electrónico graduado en la Escuela Superior Politécnica del Litoral; máster en Informática graduado en Chile; obtuvo su máster en Diseño de Sistemas Electrónicos por la Universidad Tecnológica de La Habana, Cuba; es magíster en Gestión Académica Universitaria, tiene un diplomado superior en Pedagogía Universitaria y es experto en procesos *e-learning*.

Es miembro fundador del grupo de investigación GITCE. Sus intereses de investigación son los sistemas digitales, el modelamiento y predicción del índice de radiación ultravioleta (IUV). Ha publicado algunos textos básicos y varios artículos en revistas científicas; fue Ingeniero de Diseño Digital en ELECSA; ha ganado varios premios.

Es docente titular en la Facultad de Informática y Electrónica de la Escuela Superior Politécnica de Chimborazo; fue autoridad académica y profesor titular en la Facultad de Ingeniería de la Universidad Nacional de Chimborazo; es miembro de la Asociación Mundial de Tutores virtuales (ATM).

Verónica Elizabeth Mora Chunllo es ingeniera en Electrónica y Computación graduada en la Escuela Superior Politécnica de Chimborazo, magíster en Ingeniería de *Software* graduada en la Escuela Superior Politécnica del Ejército; es máster en Diseño de Sistemas Electrónicos por la Universidad Tecnológica de La Habana, Cuba; es magíster en Educación a Distancia. Tiene un diplomado superior en las Nuevas Tecnologías de Información y Comunicación Aplicadas a la práctica docente; es experta en procesos *e-learning*.

Es docente titular en la Facultad de Informática y Electrónica de la Escuela Superior Politécnica de Chimborazo, es directora y fundadora del grupo de investigación GITCE. Sus intereses de investigación son los sistemas digitales. Es miembro de la Asociación Mundial de Tutores virtuales (ATM), fue miembro de la Comisión Editorial de la revisa científica *Perspectivas* FIE-ESPOCH; fue Subdirectora de Posgrado en la ESPOCH. Ha publicado algunos textos básicos y varios artículos en revistas científicas nacionales.

PREFACIO

Este libro está basado en las experiencias académicas que los autores adquirieron en dos importantes universidades. Este es el último volumen de cuatro libros escritos sobre máquinas secuenciales sincrónicas.

Esta obra está diseñada para un segundo curso de sistemas digitales que es fundamental en las carreras de Ingeniería Electrónica, Telecomunicaciones, Control y afines.

Un diseñador de sistemas digitales debe dominar las técnicas de diseño de sistemas digitales combinacionales y secuenciales, así como también las aplicaciones informáticas que permiten el diseño asistido por computadora (CAD).

La ciencia del diseño de sistemas digitales, en lo relativo a sus conceptos y fundamentos, no ha sufrido un cambio radical todavía; sin embargo, la aparición de las herramientas CAD y, en la década de los ochenta de los lenguajes de descripción de *hardware*, como VHDL, han cambiado sustancialmente la forma como se diseñan e implementan sistemas digitales.

Este libro pretende mostrar los conceptos fundamentales del diseño de máquinas secuenciales sincrónicas mediante la técnica de diagrama de máquinas de estado algorítmicas (ASM) codificado en VHDL.

Existen infinidad de libros sobre sistemas digitales; sin embargo, este libro se adapta a las necesidades académicas y de laboratorios de las carreras de Ingeniería Electrónica de la Espoch. De ahí que uno de los objetivos de este libro es resolver estas necesidades.

Este texto presenta los conceptos fundamentales de diseño de máquinas de estado algorítmicas mediante cartas o diagramas ASM. VHDL es utilizado para implementar estos diagramas. Se exponen ejemplos mediante los cuales el estudiante aprende a construir algoritmos con ASM e implementarlos con VHDL.

CAPÍTULO 1 DISEÑO DE MÁQUINAS DE ESTADO ALGORÍTMICAS CON VHDL

1.1. Introducción a los sistemas controladores digitales

En los tres libros anteriores, se estudiaron las máquinas o circuitos secuenciales sincrónicos. Estos circuitos secuenciales, junto con los circuitos combinacionales, son las partes pequeñas pero fundamentales de un sistema digital.

En general, cuando se va a diseñar un sistema digital, se debe, en primer lugar, analizar y definir el tipo de circuito que se requiere, es decir, si es un circuito combinacional o un circuito secuencial.

Si es combinacional, entonces, la relación que existe entre las señales de entrada y salida (la función de transferencia) está dada por una tabla de verdad y, por lo tanto, la técnica de diseño consiste en la elaboración de una tabla de verdad que relaciona las entradas con las salidas.

Si se trata de un circuito secuencial, el algoritmo que establece la relación entre las entradas y las salidas se representa en forma gráfica, por ejemplo, mediante un diagrama de estados, un diagrama MDS o un diagrama ASM. Dicho de otra manera, la técnica de diseño de este tipo de circuitos es mediante algún diagrama.

Los diagramas de estado se utilizan para diseñar máquinas tipo Mealy y Moore que normalmente no tienen muchas entradas y tampoco muchas salidas; es decir, son sistemas pequeños. Una técnica para diseñar sistemas digitales grandes y complejos es mediante los diagramas o cartas ASM ASM significa máquina de estado algorítmica y estas siglas tienen su origen en *Algorithmic State Machine*.

En general, un sistema digital grande y complejo puede considerarse como compuesto por dos grandes bloques que interactúan entre sí. Uno de ellos es el sistema controlador y el otro es el subsistema (formado por bloques combinacionales y/o secuenciales). El propósito de este libro es el diseño de este tipo de sistemas.

1.1.1 El sistema controlador

Un sistema controlador, es un sistema secuencial; tiene señales de entrada y señales de salida. La información presente en las señaes de entrada es procesada por el sistema controlador y, en función de esto, genera información que es puesta en las señales de salida.

La figura 1.1 muestra un ejemplo de un sistema digital. Allí se puede ver el sistema controlador junto con otros bloques. Estos bloques, en conjunto, forman lo que se denomina el subsistema. Al sistema controlador y al subsistema, en conjunto, se denomina partición funcional. La señal de reloj (CLK); en un sistema secuencial sincrónico, genera cada intervalo de tiempo (cada período) en que el sistema controlador debe realizar alguna acción.

La figura 1.1 muestra un ejemplo de un sistema digital. En este se puede observar las señales de entrada al sistema controlador y son: S, R y N. S y R son las señales que reciben información desde el mundo exterior, pues no son generadas por el subsistema. N es generada por el subsistema. Las señales de salida son: LED, G y C. La señal LED envía información al mundo exterior, así como también al subsistema. El subsistema tiene dispositivos combinacionales y secuenciales.

El sistema controlador que muestra la figura 1.1 recibe en sus entradas (S, R y N) información, procesa esa información y, en respuesta a este procesamiento, genera los respectivos bits de información en sus salidas G, C y LED.

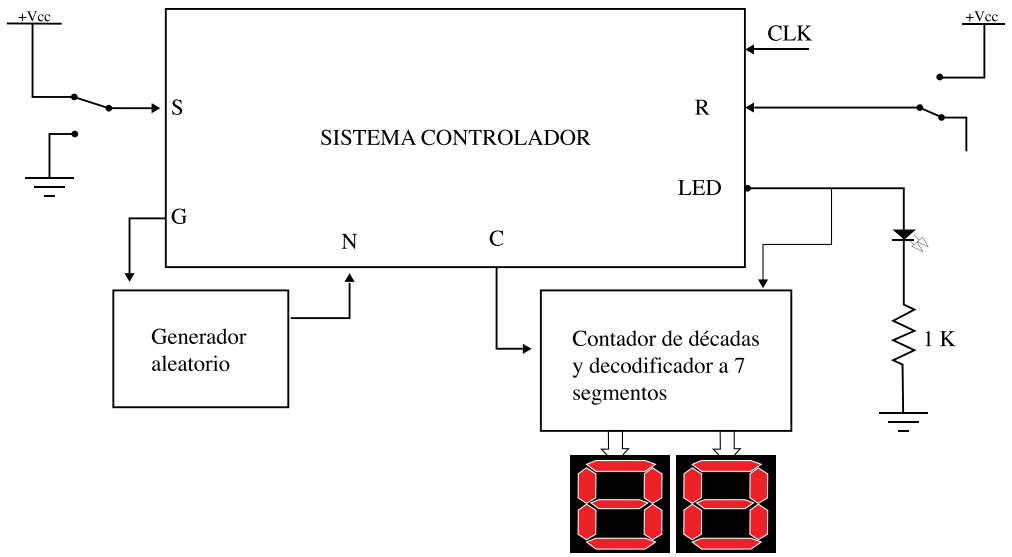


Figura 1.1. Diagrama de bloques de un sistema digital, la partición funcional

1.2 Diseño del sistema controlador con ASM

El diseño de un sistema controlador no es siempre una tarea simple, sobre todo en sistemas controladores digitales que tienen que coordinar una gran cantidad de entradas y salidas.

La coordinación entre las entradas y las salidas se establece a través de un algoritmo a partir del cual se construye el *hardware* del sistema controlador.

Se requiere, por lo tanto, de algún método para elaborar algoritmos en forma de un gráfico. Uno de ellos es el *algorithmic state machine chart* (carta ASM) que es un gráfico con símbolos especiales que describe el funcionamiento de un sistema controlador. Este diagrama fue desarrollado por Chris Clare. Otro tipo de diagramas son los MDS (mnemonic documented state diagrams) que son una generalización de los diagramas de estado.

En este libro, se utilizan los diagramas ASM y el lenguaje de descripción de *hardware* para circuitos integrados de muy alta velocidad (VHDL) para diseñar sistemas controladores. El *hardware* del sistema controlador puede ser construido sobre un FPGA o podría ser un ASIC.

1.2.1 Diagramas ASM

Un diagrama ASM se parece a un diagrama de flujo, pero, mientras un diagrama de flujo describe macrooperaciones que no actúan directamente sobre el *hardware*, un diagrama ASM trabaja con señales que actúa directamente sobre los circuitos.

Los símbolos que se utilizan para elaborar un diagrama ASM son:

- El diamante
- El rectángulo
- El rectángulo con bordes suaves o curvos

El diamante es el símbolo que representa una decisión, en su interior contiene el nombre de una variable que representa a una entrada al sistema controlador o una expresión que contiene varias entradas. Una entrada, o una expresión, siempre tiene uno de sus dos posibles valores, puede ser verdadera o falsa; si el valor

de la entrada es verdadero toma un camino y, si es falso toma otro camino. Se parece a las instrucciones del tipo: IF (entrada) THEN (camino 1) ELSE (camino 2).

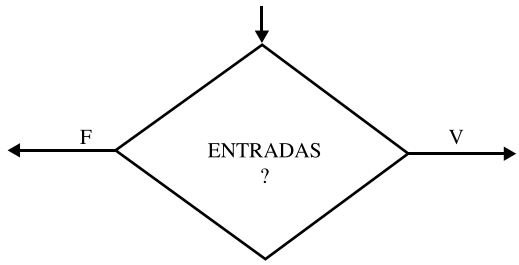


Figura 1.2. Símbolo para tomar una decisión

El rectángulo representa un estado cualquiera del sistema controlador. Un estado, en función del tiempo, está representado por un ciclo del reloj. Un sistema controlador puede encender alguna(s) señal(es) de salida(s) en cada estado. La figura 1.3 muestra un estado con su respectiva simbología.

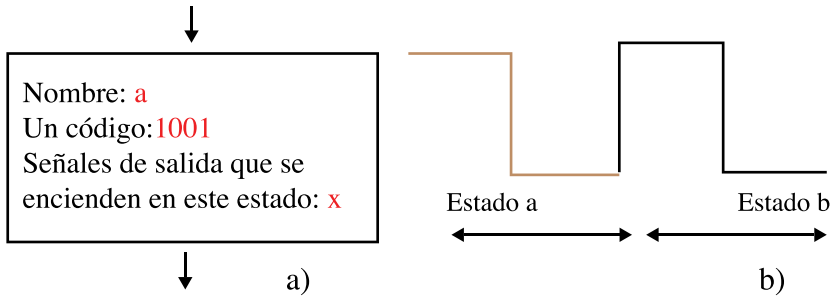


Figura 1.3. Símbolo de un estado

Como muestra la figura 1.3, un estado debe tener un nombre, por ejemplo "a"; un código, por ejemplo "1001"; y las señales de salida que se activen en ese estado, por ejemplo "X". X es la salida que se enciende o activa en el estado "a". El tiempo que se mantiene una señal de salida encendida es igual al período del reloj del sistema controlador.

El rectángulo con esquinas curvas tiene la(s) señal(es) que debe(n) encenderse o activarse en forma condicional al valor de una entrada o una expresión (en caso de varias entradas). La figura 1.4 muestra que si la entrada "X" es verdadera, se enciende la señal de salida W, y si "X" es falsa, se enciende la salida B.

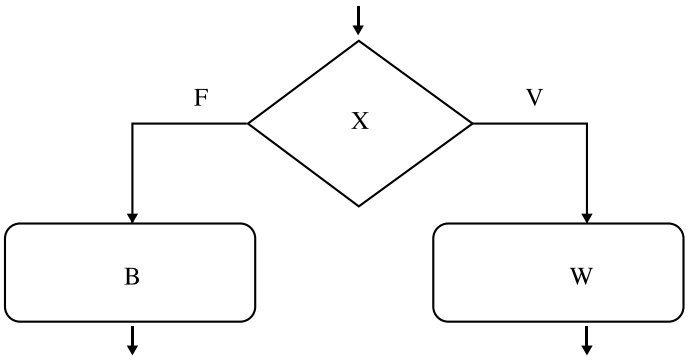


Figura 1.4. Salidas condicionales

Un ejemplo, de una parte de un diagrama ASM lo muestra la figura 1.5. En este diagrama, en el estado "a", se pregunta por la entrada "X". Si X es verdadera, se enciende la salida "W" y, si X es falsa, se enciende la salida "B". Los siguientes estados son "c" y "b".

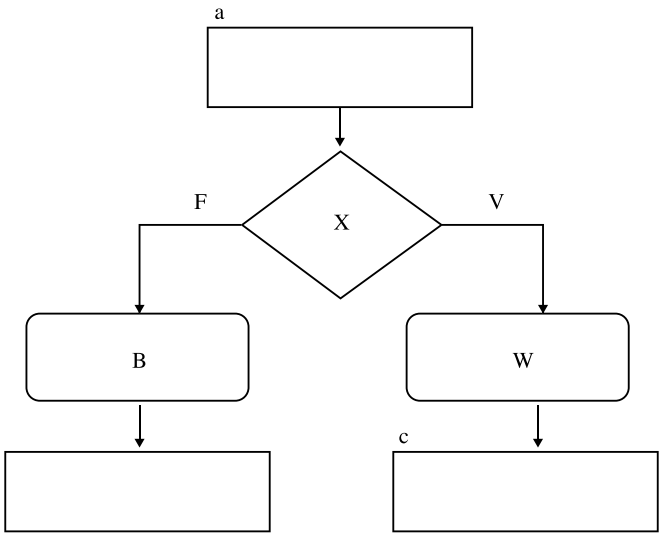


Figura 1.5. Un ejemplo de diagrama ASM

El nombre de un estado y su código se suele escribir en diferentes posiciones en el rectángulo que representa al estado. Por ejemplo, una forma es escribir el nombre del estado en la esquina superior izquierda del rectángulo y fuera de él y el código del estado se escribe en la esquina superior derecha y fuera del rectángulo, como muestra la figura 1.6. RE es la señal que se activa en este estado a.

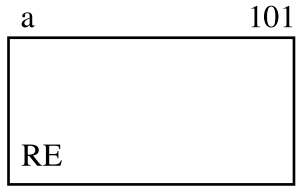


Figura 1.6. Un estado con su nombre, su código y la señal que se activa

En ocasiones, es necesario escribir alguna acción que se debe ejecutar en un estado, como, por ejemplo, incrementar la cuenta de un registro (c<- c+1). Esta acción se escribe dentro del rectángulo como muestra la figura 1.7

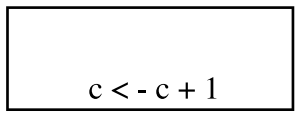


Figura 1.7. Un estado con su nombre, código y una acción

Se acostumbra escribir los nombres de los estados con letras minúsculas como, por ejemplo a, b, etc. Sin embargo, el nombre de un estado puede ser cualquier nombre por ejemplo: estado_1, estado_de_reset, inicio, etc. Pero cuando, a partir de un diagrama ASM, se va a escribir un programa o código en VHDL, los nombres deben estar en concordancia con las reglas que se utiliza en VHDL para escribir identificadores.

Se puede también, como ya se indicó, en un estado, realizar operaciones sobre registros o entre registros. Estas operaciones se especifican dentro del rectángulo (estado) utilizando la simbología de operaciones para registro, por ejemplo, si un registro A debe cargarse con el contenido de algún otro registro B, simbólicamente, esta operación se representa de la manera siguiente:

$$A \leftarrow B$$

Si un registro C, por ejemplo, debe cargarse con un valor constante, como cero, simbólicamente se representa de la manera siguiente:

$$C \leftarrow 0$$

El incremento de un registro se representa de la siguiente forma:

$$A \leftarrow A + 1$$

El decremento de un registro simbólicamente se escribe como:

$$A \leftarrow A - 1$$

Debe notarse que, cuando las salidas se encienden en forma incondicional, es decir, en un estado, estas salidas corresponden a una máquina de tipo Moore. Si las salidas se encienden en forma condicional, corresponden entonces a una máquina tipo Mealy. Esto implica que un diagrama ASM puede tener salidas de los dos tipos de máquinas.

A continuación, con la finalidad de adquirir cierta experiencia con los diagramas ASM, se proponen y resuelven algunos ejemplos que más bien son simples.

Ejemplo 1.1

Se pide diseñar una máquina electrónica que mida el tiempo de reacción del sentido de la visión (los ojos) de un ser humano, para lo cual se debe tomar en consideración los siguientes requisitos de diseño.

La máquina debe tener un *switch* de inicio que, al ser presionado por el usuario, haga que la máquina esté lista para iniciar su funcionamiento.

La forma como debe funcionar la máquina es la siguiente: luego de que el usuario haya presionado el *switch* de inicio, se debe encender un led en cualquier momento y cuando el usuario se percate de que el led se encendió, debe presionar el *switch* de reacción tan rápido como le sea posible. El sistema debe exhibir el tiempo que transcurrió entre el encendido del led y la presión del *switch* de reacción por parte del usuario.

Solución. Primero se debe realizar la partición funcional del sistema. Aquí se deben especificar con claridad las señales que entran al sistema controlador

y las que salen de él, así como también, todos los bloques que conforman el subsistema.

Primero se dibuja, como muestra la figura 1.8, un bloque que represente al sistema controlador; los otros bloques y dispositivos que se encuentran alrededor del sistema controlador forman el subsistema, y todo este conjunto (sistema controlador y subsistema) toma el nombre de partición funcional.

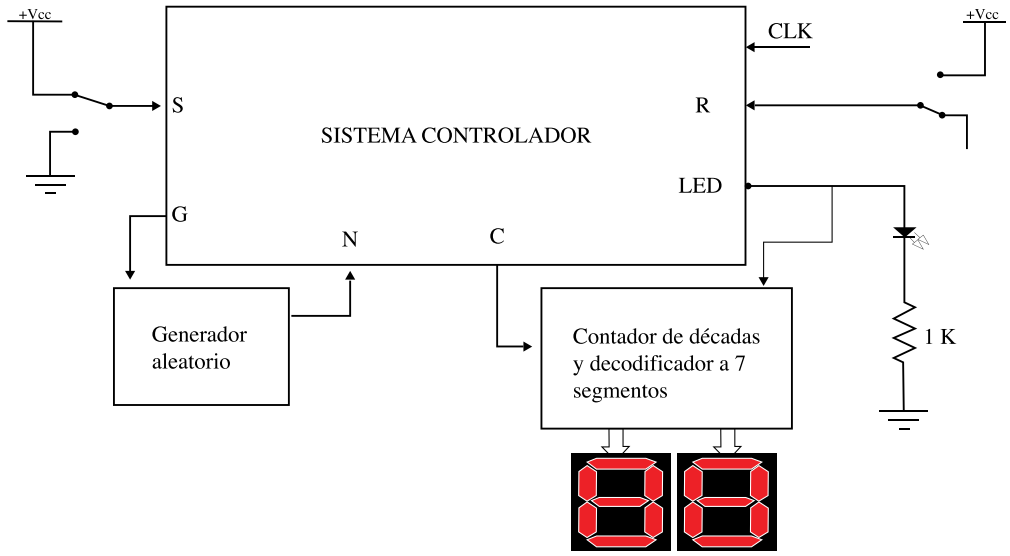


Figura 1.8. Diagrama de la partición funcional

Cada señal, ya sea de entrada o de salida, debe tener un nombre que la identifique. En la figura 1.8, la señal S representa al *switch* de inicio; R, al *switch* de reacción que debe presionar el usuario; la señal LED enciende el led y también activa el contador indica que se generó el tiempo aleatorio; y C pone a cero el contador CLK es el reloj del sistema.

El diagrama ASM del sistema que se va a diseñar se muestra en la figura 1.9. La descripción de este diagrama es la siguiente: cuando la máquina se encuentra en el estado "a", pregunta si el usuario activó el *switch* de inicio S. Si la respuesta es falsa, se queda esperando en el estado "a" hasta que el usuario presione el *switch* R.

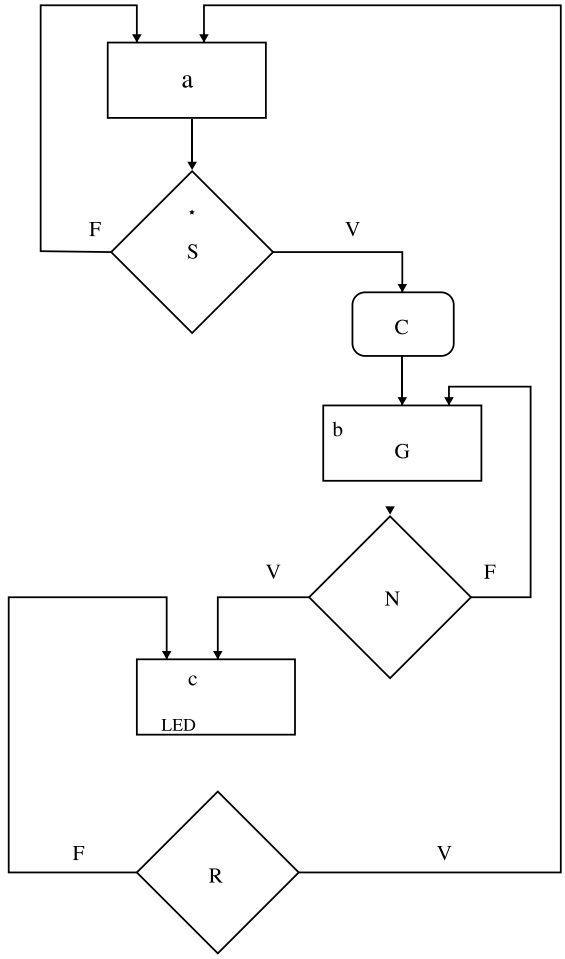


Figura 1.9. Diagrama ASM del medidor de reacción

Cuando la señal S es verdadera, se enciende la señal C en forma condicional y el contador pone en cero su cuenta. La máquina va al estado "b" y activa la señal G que es la que pone en funcionamiento el generador de tiempo aleatorio (que permite que el led se encienda en cualquier momento) y se queda en este estado "b" hasta el momento en que el generador enciende la señal N que es la que indica que se generó el tiempo aleatorio.

La máquina, luego, va al estado "c" y aquí activa la señal LED que enciende el diodo led que genera la señal luminosa que está esperando el usuario para proceder a presionar el *switch* R. El sistema cuenta (mediante el contador) el tiempo

que transcurrió entre el encendido del led y la presión del *switch* R. Este tiempo es la medida de qué tan rápido es el sentido de la visión del usuario. Del estado "c", la máquina regresa al estado "a" para iniciar otra medición. Debe notarse que la cuenta en el contador se mantiene hasta que sea presionado el *switch* S.

El generador aleatorio que en realidad es un seudogenerador, puede ser construido de diferentes maneras. Una de ellas es mediante dos contadores que trabajan sus relojes a diferentes frecuencias. Las salidas de los contadores alimentan a las entradas de un comparador y, cuando la cuenta de los dos contadores se iguala se enciende la salida denominada igual del comparador y es esa señal la que indica que se ha generado un tiempo aleatorio. En el ejemplo, la señal N indica la generación de un tiempo aleatorio. Por supuesto las señales S, R y N deben estar sincronizadas con el sistema controlador.

La señal G activa el generador aleatorio mediante una operación AND (Y) entre la señal G y cada reloj de cada contador de la siguiente manera: G. clk1, G. clk2.

clk1 y clk2 son respectivamente los relojes de los 2 contadores del generador aleatorio. Los contadores de décadas, el decodificador y el *display* a siete segmentos se pueden adquirir en forma de circuitos integrados.

1.3 Implementación de diagramas ASM con VHDL

Un diagrama ASM es un algoritmo que describe el funcionamiento de una máquina secuencial y, a partir de este diagrama, se puede sintetizar el *hardware* del sistema controlador. Tiene un número finito de estados plenamente identificados, por los cuales un sistema secuencial sincrónico debe pasar. En VHDL, se puede identificar plenamente a cada uno de estos estados definiéndolos como un tipo particular de datos, es decir, de tipo enumerados cuyos elementos son exclusivamente los estados del diagrama ASM. La declaración de estos datos enumerados es como se indica a continuación:

TYPE estado IS (estado_0, estado_1, estado_2,……estado_n);

TYPE, declara que estado es un tipo de dato enumerado y sus miembros son los nombres de todos los estados que contiene el diagrama ASM. En este caso, se supone que los nombres de los diferentes estados son: estado0, estado_1,…, estado_n.

VHDL, a cada estado le asigna un código en concordancia con el orden en que han sido escritos en la declaración TYPE. Por ejemplo, el estado0 tendrá un

código, el estado1 tendrá otro código y así sucesivamente. El número de bits que requiere cada estado depende del número de estados que contenga el diagrama ASM, por ejemplo, si hay 4 estados diferentes, se necesitarán dos bits para cada estado y a cada estado se asigna uno de los siguientes códigos: 00, 01, 10 y 11. La relación matemática entre el número de estados y los bits de cada estado está dado por 2n. El resultado de esta operación da el número de estados que se pueden codificar con n bits.

Como se indicó antes, el código de cada estado está en las salidas de los *flip-flops*, y estas salidas se actualizan únicamente cuando el flanco de subida (o bajada) del reloj de estos *flip-flops* se ha producido, y es en ese instante donde el estado presente se convierte en el estado siguiente e, inmediatamente este estado siguiente es el nuevo estado presente.

Un proceso (PROCESS) se utiliza en VHDL para pasar de un estado presente a un estado siguiente. La sintaxis de este proceso se escribe a continuación.

```
PROCESS (reloj)
BEGIN
IF (reloj'EVENT AND reloj='1') THEN
estado_presente <= estado_siguiente;
END IF;
END PROCESS
```

El siguiente paso es recorrer secuencialmente por cada uno de los estados. Para ejecutarlo, en VHDL, se requiere escribir otro PROCESS (proceso). Este proceso debe considerar que, de un estado cualquiera, se pueden ir a uno, y solo uno, de varios estados siguientes. Como muestra la figura 1.10, del estado "c" se puede ir al estado "a" o al estado "d". El estado al que vaya depende, por supuesto, del estado presente "c" y de la condición (verdadera o falsa) de la entrada R en el estado "c".

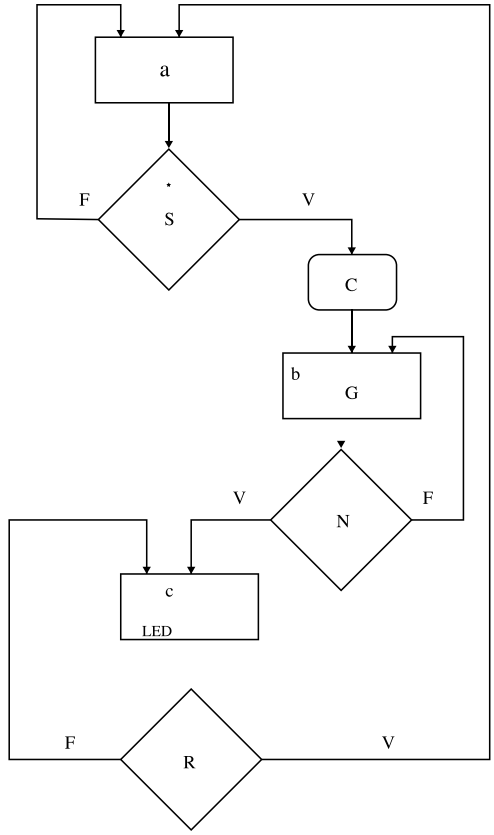


Figura 1. 10. Ejemplo de un diagrama de estados

Una parte del proceso, que permite recorrer a través de los diferentes estados de un diagrama ASM, puede ser escrito de la siguiente manera.

```
PROCESS (entradas, estado_presente)

CASE estado_presente IS
WHEN estado0 =>
IF (entradas = ...) THEN
salidas<= valor_salidas;
estado_siguiente <= estado1;
ELSE ...
```

END IF;……

Debe notarse que la ejecución de este proceso depende de las entradas así como del estado presente. El estado presente y siguiente son señales y deben ser declaradas como tales en la arquitectura en la sección de declaración de señales. Esta declaración es como se indica a continuación:

SIGNAL estado_presente, estado_siguiente: estado.

La sentencia CASE, que está dentro del proceso, debe tener tantos WHEN. cuantos estados tenga el diagrama ASM. La señal de reloj marca el instante de tiempo que permite ir de estado a estado en forma secuencial.

A continuación, se escribe un posible código VHDL, para un diagrama ASM general. Como se podrá observar en los siguientes ejemplos, la estructura del programa siempre es muy parecida para cualquier diagrama ASM.

En el primer proceso, se ha incluido la señal de *reset* para obligar a que el sistema vaya al estado 0 cuando el *reset* sea activado.

.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------------------------------------------------------
ENTITY nombre_entidad IS
PORT (
entradas: IN tipo_datos_entradas;
reset, reloj: IN STD_LOGIC;
salidas: OUT tipo_datos_salidas);
END nombre_entidad;
-------------------------------------------------------------------------------------------
ARCHITECTURE nombre_arquitectura OF nombre_entidad IS
TYPE estado IS (estado_0, estado_1, estado_2,……estado_n);
SIGNAL estado_presente, estado_siguiente: estado;
BEGIN
```

```
-------------------------------------------------------------------------------------------------

    PROCESS (reset, reloj)
    BEGIN
    IF (reset='1') THEN
    estado_presente <= estado_0;
    ELSIF (reloj'EVENT AND reloj='1') THEN
    estado_presente <= estado_siguiente;
    END IF;
    END PROCESS;

-------------------------------------------------------------------------------------------------

    PROCESS (entradas, estado_presente)
    BEGIN
    CASE estado_presente IS
    WHEN estado_0 =>
    IF (entradas = ...) THEN
    salidas<= valor_salidas;
    estado_siguiente <= estado_1;
    ELSE ...
    END IF;
    WHEN estado_1 =>
    IF (entradas = ...) THEN
    salidas<= valor_salidas;
    estado_siguiente <= estado_2;
    ELSE ...
    END IF;
    WHEN estado_2 =>
    IF (entradas = ...) THEN
```

```
salidas<= valor_salidas;
estado_siguiente <= estado_3;
ELSE ...
END IF;
WHEN estado_3 =>
IF (entradas = ...) THEN
salidas<= valor_salidas;
estado_siguiente <= estado_4;
ELSE ...
END IF;
                    ……..
                    ……
                    ……
WHEN estado_n-1 =>
IF (entradas = ...) THEN
salidas<= valor_salidas;
estado_siguiente <= estado_n;
ELSE ...
END IF;
...
END CASE;
END PROCESS;
END nombre_arquitectura;
```

Ejemplo 1.2

Escriba el código VHDL que describa el comportamiento del sistema del ejemplo 1.1.

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;
-------------------------------------------------------------------------------------------
ENTITY sistema_controlador IS
PORT (
S, N, R, reset, reloj: IN std_logic;
C, LED: OUT STD_LOGIC
);
END sistema_controlador;
-------------------------------------------------------------------------------------------
ARCHITECTURE a_sistema_controlador OF sistema_controlador IS
TYPE estado IS (a, b, c);
SIGNAL estado_presente, estado_siguiente: estado;
BEGIN
-------------------------------------------------------------------------------------------
PROCESS (reset, reloj)
BEGIN
IF (reset='1') THEN
estado_presente <= a;
ELSIF (reloj'EVENT AND reloj='1') THEN
estado_presente <= estado_siguiente;
END IF;
END PROCESS;
-------------------------------------------------------------------------------------------
PROCESS (S, N, R, estado_presente)
BEGIN
CASE estado_presente IS
WHEN a =>
```

```
IF (S = ‘1’) THEN
C<= ‘1’;
estado_siguiente <= b;
ELSE
                                    estado_siguiente <= a;
 END IF;
WHEN b =>
IF (N=’1’) THEN
estado_siguiente <= c;
ELSE
estado_siguiente <= b;
END IF;
WHEN c =>
IF (R =’1’) THEN
LED<= ‘1’;
estado_siguiente <= a;
ELSE
estado_siguiente <= c;
END IF;
END CASE;
END PROCESS;
END a_sistema_controlador;
```

Ejemplo 1.3

Diseñe un sistema controlador para dos motores A y B. El motor A (MA) y el motor B (MB) deben cumplir el ciclo que se indica a continuación en forma indefinida:

MA	OFF	ON	ON	OFF
MB	OFF	OFF	ON	ON

Tabla 1.1 Ciclo que deben ejecutar los motores MA y MB

Realice la partición funcional, el diagrama ASM y escriba el código VHDL.

La partición funcional, dada las características del sistema, es solo el sistema controlador y se muestra en la figura 1.11. Se asume que los motores se encienden con un nivel alto y están apagados con un nivel bajo.

Las señales que encienden los motores son respectivamente MA y MB, la única entrada que tiene el sistema controlador es el reloj.

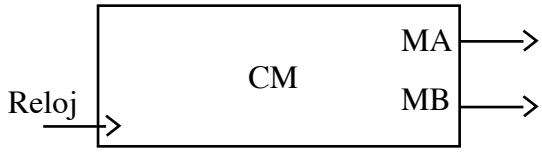


Figura 1.11. Partición funcional del controlador de los motores

El diagrama ASM es el que se indica en la figura 1.12. Los nombres de los estados son a, b, c, d. Las señales de salida son las que encienden cada motor y son MA y MB.

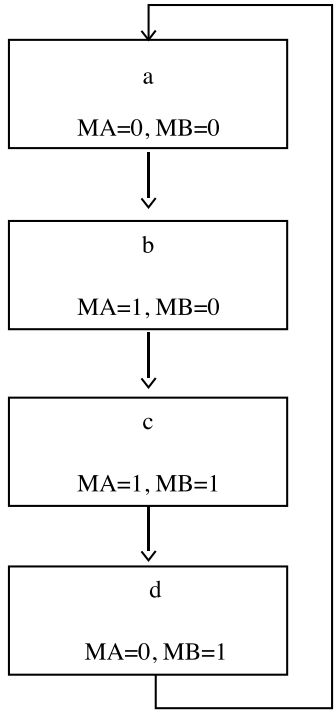


Figura 1.12. Diagrama ASM del ejemplo 1.3

El código es el siguiente:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------------------------------------------------------
ENTITY CM IS
PORT (
reloj: IN std_logic;
MA, MB: OUT STD_LOGIC
);
END CM;
-------------------------------------------------------------------------------------------
```

```
ARCHITECTURE a_CM OF CM IS
TYPE estado IS (a, b, c, d);
SIGNAL estado_presente, estado_siguiente: estado;
BEGIN
-----------------------------------------------------------------------------------------
PROCESS (reloj)
BEGIN
IF (reloj'EVENT AND reloj='1') THEN
estado_presente <= estado_siguiente;
END IF;

END PROCESS;
-----------------------------------------------------------------------------------------
PROCESS (estado_presente)
BEGIN
CASE estado_presente IS
WHEN a =>
MA<='0' ;MB<='0';
estado_siguiente <= b;
WHEN b =>
MA<='1' ;MB<='0';
estado_siguiente <= c;
WHEN c =>
MA<='1' ; MB<='1';
estado_siguiente <= d;
WHEN d =>
MA<='0'; MB<='1';
```

```
estado_siguiente <= a;
END CASE;
END PROCESS;
END a_CM;
```

La figura 1.12 a) y b) muestra, respectivamente, la simulación y el circuito sintetizado con Quartus II, del sistema controlador de este ejercicio.

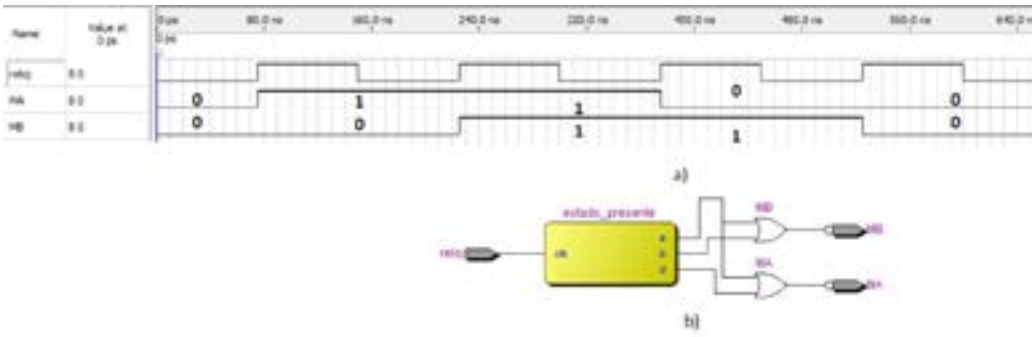


Figura 1.12. Simulación y síntesis del sistema que controla dos motores.

Ejemplo 1.4

Diseñe un sistema que detecte la secuencia "**1001" sin traslapamiento**. Asuma que cada bit ingresa al sistema con el flanco de subida del reloj. El sistema debe generar una salida denominada F cada vez que la secuencia es detectada.

El traslapamiento ocurre cuando el último bit de una secuencia es el primer bit de la siguiente secuencia. En este caso, el bit "1" del final de una secuencia no forma parte del primer bit "1" de la siguiente secuencia. Por ejemplo, la siguiente secuencia es sin traslapamiento: 100110011001….., como se observa, el bit "1" solo forma parte de la secuencia "1001" anterior y no de la siguiente. El diagrama de bloques se indica en la figura 1.13.

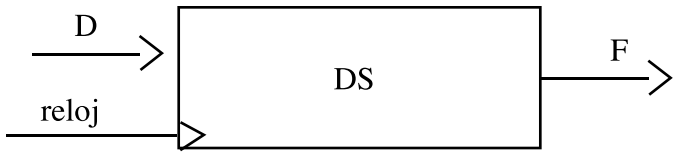


Figura 1.13. Partición funcional del detector de código sin traslapamiento

El diagrama ASM del detector de código se muestra la figura 1.14.

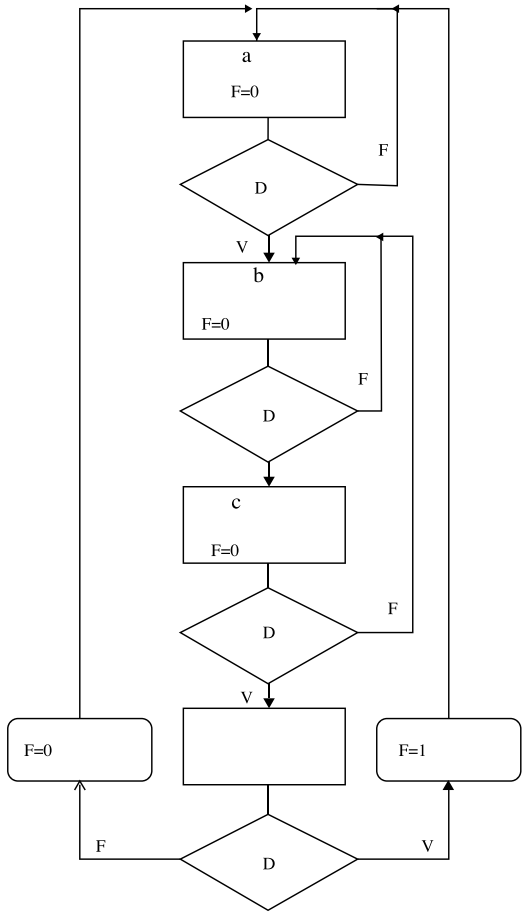


Figura 1.14. Diagrama ASM del detector de código sin traslapamiento

```
                    LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
ENTITY DS IS
PORT (
        D, reloj: IN STD_LOGIC;
        F: OUT STD_LOGIC
);
END DS;
-------------------------------------------------------------------------------------------
ARCHITECTURE aDS OF DS IS
        TYPE estado IS (a, b, c, e);
        SIGNAL estado_presente, estado_siguiente: estado;
BEGIN
-------------------------------------------------------------------------------------------
PROCESS (reloj)
BEGIN
                IF (reloj'EVENT AND reloj='1') THEN
                        estado_presente <= estado_siguiente;
        else
                         null;
        END IF;
END PROCESS;
-------------------------------------------------------------------------------------------
PROCESS (D, estado_presente)
                BEGIN
                        CASE estado_presente IS
                                WHEN a =>
                                        IF (D = '0') THEN
                                                F<= '0';
```

```
                    estado_siguiente <= a;
            ELSE
                    F<= '0';
                    estado_siguiente <= b;
            END IF;
    WHEN b =>
            IF (D ='1') THEN
                    F<= '0';
                    estado_siguiente <= b;
            ELSE
                    F<= '0';
                    estado_siguiente <= c;
            END IF;
    WHEN c =>
            IF (D='1' ) THEN
                    F<= '0';
                    estado_siguiente <= b;
            ELSE
                    F<= '0';
                    estado_siguiente <= e;
            END IF;
    WHEN e =>
            IF (D ='1' ) THEN
                    F<= '1' ;
                    estado_siguiente <= a;
            ELSE
                    F<= '0';
```

```
                                    estado_siguiente <= a;
                                END IF;
                            END CASE;
                        END PROCESS;
    END aDS;
```

La figura 1.15 a) y b), muestra la simulación y el RTL del detector de código respectivamente. La simulación muestra la detección de dos códigos seguidos sin traslapamiento.

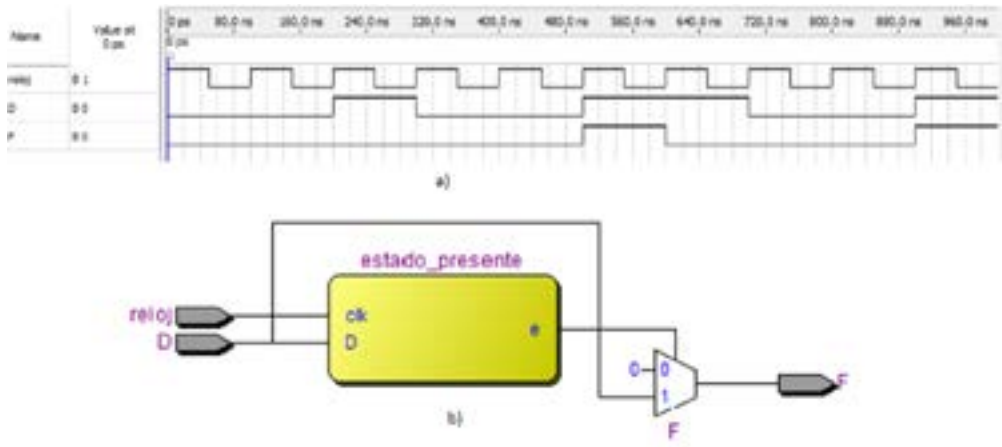


Figura 1.15. Simulación y RTL del sistema que detecta la secuencia 1001

Ejemplo 1.5

Diseñe un sistema que detecte la secuencia "1001" **con** traslapamiento. Cada bit ingresa al sistema con el flanco de subida del reloj. El sistema debe generar una salida denominada F cada vez que la secuencia indicada es detectada.

El diagrama de bloques se indica en la figura 1.13. El sistema tiene, además del reloj una señal de entrada de datos que se llama D y una salida F.

La partición funcional, para este ejemplo, está formada solo por el sistema controlador. El traslapamiento ocurre cuando dos o más secuencias se han generado en forma consecutiva y significa, para la secuencia de este ejemplo, que el

bit ”1” del final de una secuencia forma parte del primer bit “1” de la siguiente secuencia, por ejemplo: 100***1***00***1***00***1***….., como se observa el bit ***“1”*** forma parte de la secuencia “1001” anterior y siguiente. El diagrama ASM se indica en la figura 1.16.

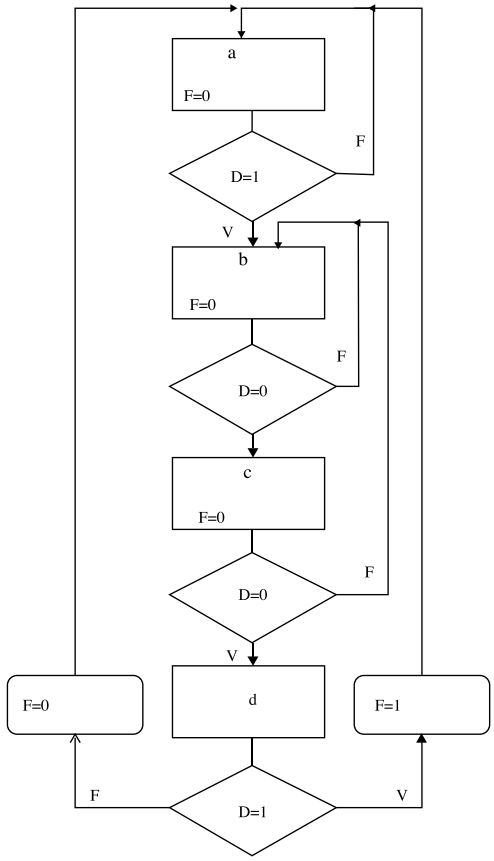


Figura 1.16. Diagrama ASM del detector de código con traslapamiento

El diagrama ASM del detector de secuencia es similar al del ejemplo anterior. La diferencia está en que, una vez detectada la secuencia en el estado e, en lugar de retornar al estado a, el sistema va al estado b, ya que el bit “1”, último de la secuencia anterior, es el bit “1” de la siguiente secuencia.

El código VHDL del diagrama ASM se indica a continuación:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
ENTITY DS IS
PORT (
        D, reloj: IN STD_LOGIC;
        F: OUT STD_LOGIC
);
END DS;
-------------------------------------------------------------------------------------------
ARCHITECTURE aDS OF DS IS
        TYPE estado IS (a, b, c, e);
        SIGNAL estado_presente, estado_siguiente: estado;
BEGIN
-------------------------------------------------------------------------------------------
PROCESS (reloj)
BEGIN
            IF (reloj'EVENT AND reloj='1') THEN
                estado_presente <= estado_siguiente;
else
                 null;
        END IF;
END PROCESS;
-------------------------------------------------------------------------------------------
PROCESS (D, estado_presente)
            BEGIN
                CASE estado_presente IS
                    WHEN a =>
                        IF (D = '0') THEN
                            F<= '0';
                            estado_siguiente <= a;
```

```
            ELSE
                F<= '0';
                estado_siguiente <= b;
            END IF;
        WHEN b =>
            IF (D ='1') THEN
                F<= '0';
                estado_siguiente <= b;
            ELSE
                F<= '0';
                estado_siguiente <= c;
            END IF;
        WHEN c =>
            IF (D='1' ) THEN
                F<= '0';
                estado_siguiente <= b;
            ELSE
                F<= '0';
                estado_siguiente <= e;
            END IF;
        WHEN e =>
            IF (D ='1' ) THEN
                F<= '1' ;
                estado_siguiente <= b;
            ELSE
                F<= '0';
                estado_siguiente <= a;
```

```
                    END IF;
                END CASE;
            END PROCESS;
    END aDS;
```

Ejemplo 1.6

Diseñe una máquina que venda refrescos.

Para diseñar cualquier sistema se debe primero tener una descripción de sus requerimientos. Estos pueden ser establecidos por el solicitante o por el ingeniero (diseñador).

El enunciado del ejercicio es simple y solo indica que se debe diseñar una máquina que venda refrescos. En este caso, el ingeniero (el diseñador) debe establecer los requerimientos más adecuados. Los requerimientos que la máquina debe tener, para este ejemplo, son los siguientes:

Módulo receptor de monedas, el receptor de monedas debe tener un orificio, por el cual se las ingresa una a la vez, y que estará abierto solo cuando reciba la orden de abrir, mediante la señal abrir receptor (AR). Si esta señal es verdadera, el orifico se abre hasta que pase una y solo una moneda, el orificio se cierra automáticamente cuando una moneda ha pasado, el receptor debe aceptar solo monedas de cinco, 10, 25, 50 y 100 (un dólar) centavos.

El receptor de monedas debe receptar las monedas, una a la vez y solo después de que el usuario ha presionado la tecla refresco.

En el receptor de monedas, hay cinco líneas denominadas: A1, A2, A3, A4, A5, (a todas estas líneas se las identifica en conjunto con una sola letra: la A) una línea por tipo de moneda, y solo puede estar encendida una línea a la vez, por ejemplo, si el usuario ingresó una moneda de un dólar, la línea que se enciende es la A1 y las demás están apagadas. Cada línea A se codifica con su respectivo valor en centavos de dólar en las líneas D (D0, D1, etc.), como se indica en la tabla 1.2. La señal DR decrementa la suma de las moneadas en 1 centavos.

El receptor de monedas deberá también calcular el valor total en centavos que el usuario a ingresado, y debe generar dos señales que indiquen si el valor total es igual (VI), o mayor (VM) que el valor del refresco. Lógicamente si no es igual y no es mayor significa que es menor; de ahí que solo son necesarias dos señales.

Centavos	CÓDIGO MONEDA					VALOR DE LA MONEDA							
	A 5	**A 4**	**A 3**	**A 2**	**A 1**	D 7	D 6	D 5	D 4	D 3	D 2	D 1	D 0
100=1 dólar	1	0	0	0	0	0	1	1	0	0	1	0	0
50	0	1	0	0	0	0	0	1	1	0	0	1	0
25	0	0	1	0	0	0	0	0	1	1	0	0	1
10	0	0	0	1	0	0	0	0	0	1	0	1	0
5	0	0	0	0	1	0	0	0	0	0	1	0	0

Tabla 1.2. Codificación de las monedas

El módulo de refresco dispone de una tecla R que al ser presionada por el usuario, genera la señal R, que le indica al sistema controlador que el usuario desea comprar un refresco.

El módulo de aviso está compuesto por un sintetizador de voz, que continuamente está emitiendo los fonemas: “Si desea un refresco presione la tecla R”. Este módulo deja de emitir los fonemas cuando el usuario ha presiona la tecla R.

El módulo que alerta al usuario para que ingrese una moneda está compuesto por un sintetizador de voz que emite los fonemas: “Ingrese una moneda” cuando recibe en su entrada la señal denominada: IM (ingrese moneda).

El módulo que da vuelto, recibe la señal dar vuelto (DV) para que inicie esta operación. Este módulo solo da el vuelto en monedas de cinco centavos. Una vez que ha dado de vuelto, una moneda de cinco centavos retorna la señal vuelto dado (VD).

El módulo de entrega del refresco tiene una señal para dar el refresco (DR). Cuando esta es verdadera, entrega el refresco y retorna una señal de refresco dado (RD) cuando ha entregado el refresco al usuario.

Con estas especificaciones, se elabora la partición funcional, que no es otra cosa que un diagrama de bloques muy detallado, que incluye todas las señales y todos los elementos que conforman el sistema, es decir, el sistema controlador y los subsistemas, este diagrama se muestra en la figura 1.17. A partir de la partición funcional, se desarrolla el diagrama ASM, que es el algoritmo que describe el funcionamiento del sistema controlador. La figura 1.18 muestra este diagrama.

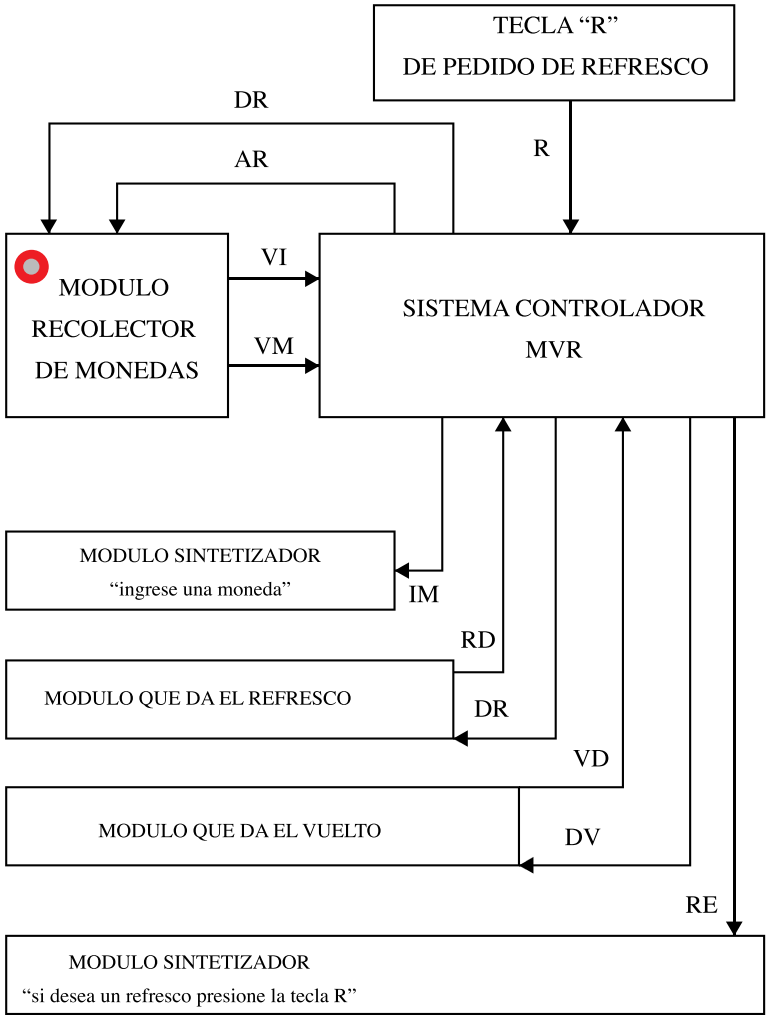


Figura 1.17. Partición funcional de la máquina vendedora de refrescos

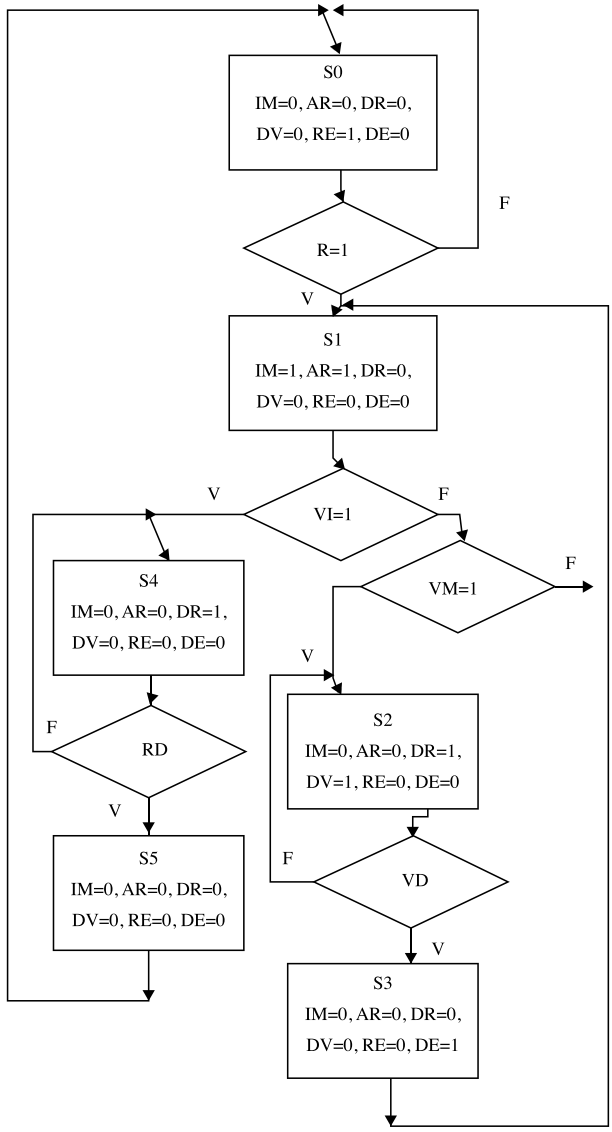


Figura 1.18. Diagrama ASM de la máquina vendedora de refrescos

En la partición funcional no se ha incluido un interruptor de encendido, que alimentaria a la máquina con energía. Tampoco se ha incluido un botón de inicio, botón que pondría a la máquina lista para trabajar.

Se ha incluido un botón de *reset* en el programa en VHDL, a pesar de que este botón no está en la partición funcional. Cuando es verdadero, posiciona a

la máquina en el estado S0 y, cuando es falso, permite que la máquina realice su función.

El programa contiene dos procesos. El uno permite recorrer el diagrama ASM estado por estado y el otro proceso enciende o apaga las señales de salida en cada estado. El código VHDL para la máquina vendedora de refrescos no es nada complicado de entender, por esta razón no se explica, y es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
entity mvr is
    port
    (
                        clk             : in     std_logic;
            r, vi,vm, rd, vd, reset     : in     std_logic;
            im,ar, dr, dv, re,de            :  out        std_logic
    );

end mvr;
architecture rtl of  mvr is
    type state_type is (s0, s1, s2, s3,s4,s5);
    signal state : state_type;
begin
    process (clk, reset)
    begin
            if reset = '1' then
                    state <= s0;
            elsif (rising_edge(clk)) then
                    case state is
                            when s0=>
```

```
                if r = '1' then
                        state <= s1;
                else
                        state <= s0;
                end if;
        when s1=>
                if vi = '1' then
                        state <= s4;
                        elsif vm='1' then
                        state <= s2;
                else
                        state <= s1;
                end if;
        when s2=>
                if vd = '1' then
                        state <= s3;
                else
                        state <= s2;
                end if;
        when s3=> state<= s1;
        when s4=>
                if rd = '1' then
                        state <= s5;
                else
                        state <= s4;
                end if;
```

```
                    when s5=> state<= s0;
                end case;

            end if;
    end process;
    process (state, r,vi,rd,vd)
    begin
case state is
    when s0=> im<='0'; ar<='0'; dr<='0'; dv<='0'; re<='1'; de<='0';
    when s1=> im<='1'; ar<='1'; dr<='0'; dv<='0'; re<='0'; de<='0';
    when s2=> im<='0'; ar<='0'; dr<='0'; dv<='1'; re<='0'; de<='0';
    when s3=> im<='0'; ar<='0'; dr<='0'; dv<='0'; re<='0'; de<='1';
    when s4=> im<='0'; ar<='0'; dr<='1'; dv<='0'; re<='0'; de<='0';
    when s5=> im<='0'; ar<='0'; dr<='0'; dv<='0'; re<='0'; de<='0';
end case;
    end process;
end rtl;
```

La figura 1.19 muestra el RTL obtenido mediante Quartus II del programa de la máquina vendedora de refrescos. Como se puede ver, el bloque *state* contiene todas las señales de entrada y salida a la máquina.

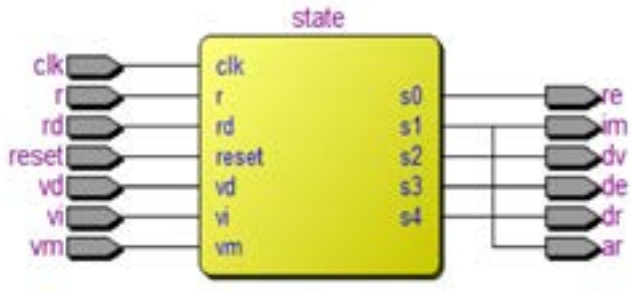


Figura 1.19. RTL de la máquina vendedora de refrescos

En la figura 1.20, se puede ver el diagrama de estados de la máquina vendedora de refrescos, este diagrama ha sido obtenido con la herramienta CAD Quartus II.

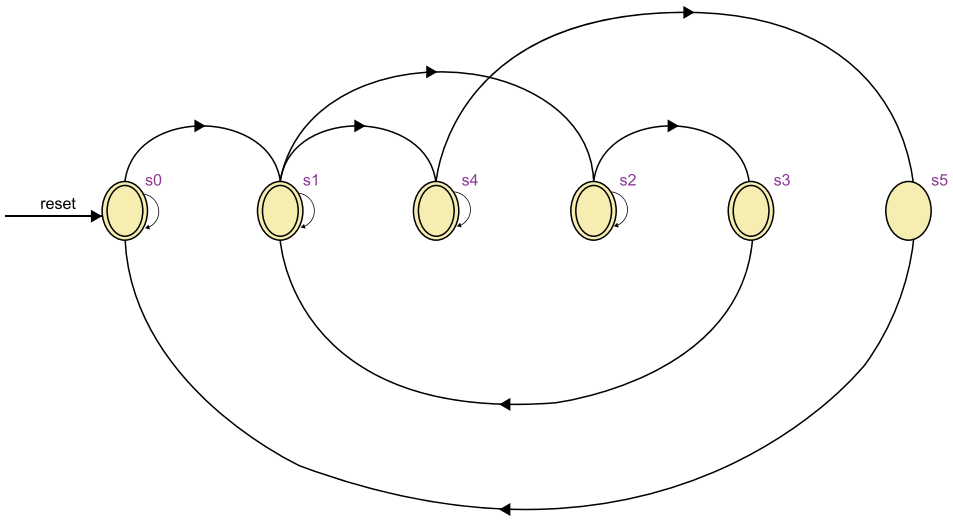


Figura 1.20. Diagrama de estados de la máquina vendedora de refrescos

En la figura 1.21, se puede ver el resultado de una simulación del programa de la máquina vendedora de refrescos.

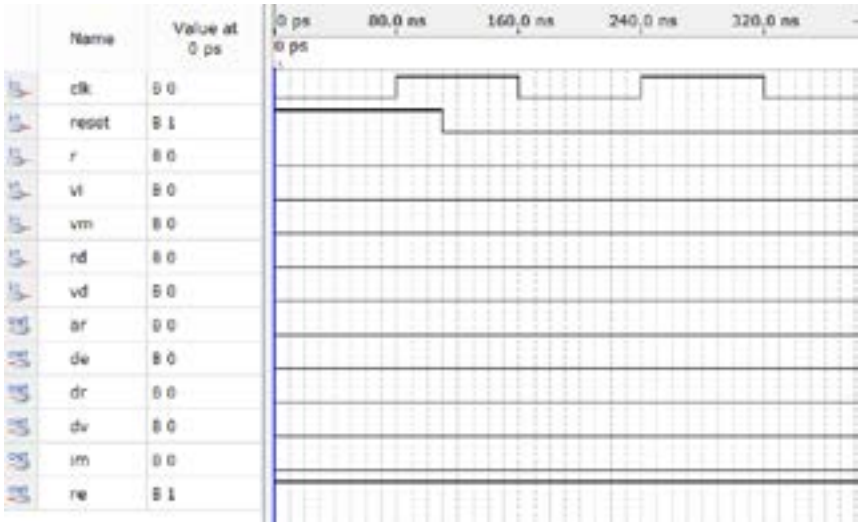


Figura 1.21. Simulación para la señal *reset* igual a cero

Como se puede ver en la figura 1.21, cuando la señal de *reset* es igual a un uno lógico, la máquina va al S0 y se mantiene allí mientras la señal r sea falsa.

En el estado S0, la única señal que se enciende es la señal re que activa el módulo que le indica al usuario que seleccione un refresco.

La figura 1.22 muestra la simulación de la máquina cuando la señal de *reset* es igual a cero y la señal r se hace verdadera en el segundo flanco del reloj. Allí se encienden las señales Ar e Im como indica el diagrama ASM, la señal va a cero porque esta solo se enciende en el estado S0.

La figura 1.23 muestra la simulación cuando r es verdadero y vi es verdadera. Del diagrama ASM se puede ver que la máquina debe ir al estado S4 y encender la salida. Como se puede apreciar en el diagrama de tiempo de la simulación, la máquina efectivamente va al estado S4 y enciende la salida DR.

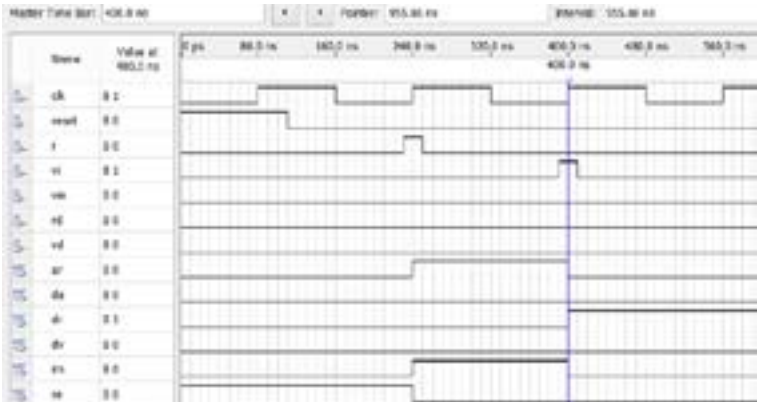


Figura 1.22. Simulación para la señal r verdadera

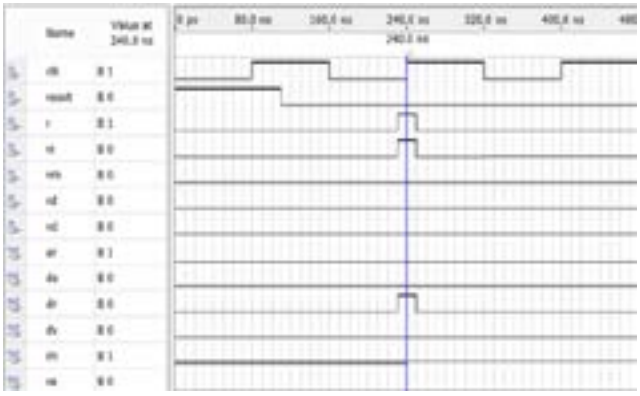


Figura 1.23. Simulación para las señales r y vi verdaderas

Ejemplo 1.7

Elabore el diagrama ASM de un sistema controlador que obtenga el complemento a dos de un número de cuatro bits, los bits ingresan uno a uno y con cada flanco de subida del reloj.

El diagrama de bloques del sistema controlador se muestra en la figura 1.24. Tiene una señal de entrada de datos denominada D aparte del reloj y una señal de salida denominada C. Se supone que la señal D recibe una cadena infinita de bits en serie y sincronizados con el flanco de subida del reloj. Cada cuatro bits se tiene un dato de información donde el bit menos significativo llega primero a D. En la salida C se tiene el complemento a dos de los bits de entrada.

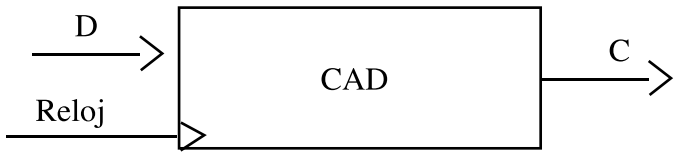


Figura 1.24. Sistema que genera el complemento a dos

Para hallar el complemento a dos de un número de cuatro bits se aplica el siguiente algoritmo a la cadena de bits: se analiza el dato empezando desde el bit menos significativo. Se busca el primer uno; hallado este en el dato, se lo deja igual y el *reset* o de los bits del dato se complementan (para el ejemplo el dato está compuesto por cuatro bits). Este algoritmo se muestra en la tabla 1.3.

		LSB	MSB			LSB	MSB			LSB
DATO	…	0	0	0	0	1	0	1	1	0
Ca2 DEL DATO	…	0	1	1	1	1	1	0	1	0

Tabla 1.3. Algoritmo para obtener el complemento a dos de un número

El diagrama ASM del algoritmo para obtener el complemento a dos de un número de cuatro bits se muestra en la figura 1.25.

El programa o codificación en VHDL del diagrama ASM de la figura 1.25 se muestra a continuación.

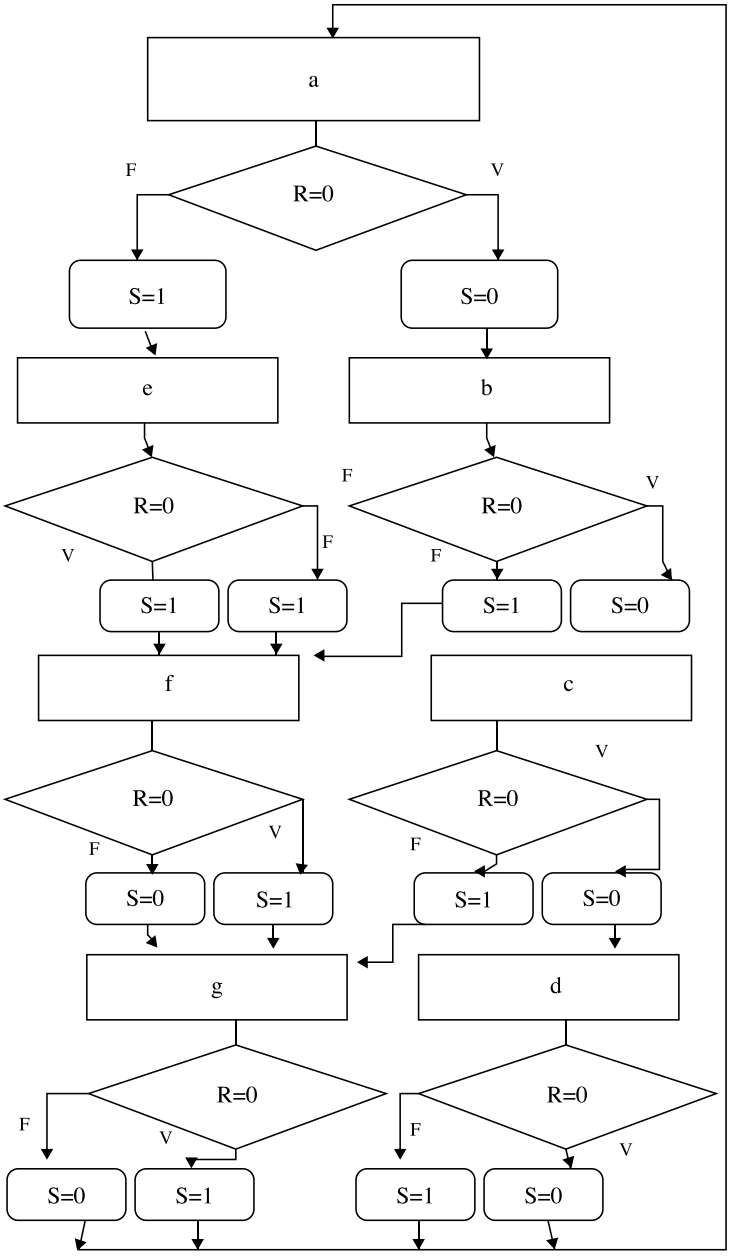


Figura 1.25. Diagrama ASM para obtener el complemento a dos

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------------------------------------------------------
ENTITY CAD IS
PORT (
R, reloj: IN STD_LOGIC;
S: OUT STD_LOGIC;
);
END CAD;
-------------------------------------------------------------------------------------------
ARCHITECTURE aCAD OF CAD IS
TYPE estado IS (a, b, c, d, e, f, g);
SIGNAL estado_presente, estado_siguiente: estado;
BEGIN
-------------------------------------------------------------------------------------------
PROCESS (reloj)
BEGIN
                        IF (reloj'EVENT AND reloj='1') THEN
estado_presente <= estado_siguiente;
 END IF;
END PROCESS;
-------------------------------------------------------------------------------------------
PROCESS (R, estado_presente)
BEGIN
CASE estado_presente IS
WHEN a =>
IF (R = '0') THEN
```

```
S<= ‘0’;
estado_siguiente <= b;
ELSE
S<= ‘1’;
estado_siguiente <= e;
END IF;
WHEN b =>
IF (R =’0’ THEN
S<= ‘0’;
estado_siguiente <= c;
ELSE
                                S<= ‘1’;
estado_siguiente <= f;
END IF;
WHEN c =>
IF (R=’0’) THEN
S<= ‘0’;
estado_siguiente <= d;
ELSE
S<= ‘1’;
estado_siguiente <= g;
END IF;
WHEN d =>
IF (R =’0’) THEN
S<= ‘0’;
estado_siguiente <= a;
ELSE
S<= ‘1’;
```

```
estado_siguiente <= a;
 END IF;
WHEN e =>
IF (R ='0' THEN
S<= '1';
estado_siguiente <= f;
ELSE
                                S<= '0';
estado_siguiente <= f;
END IF;
WHEN f =>
IF (R='0') THEN
S<= '1';
estado_siguiente <= g;
ELSE
S<= '0';
estado_siguiente <= g;
END IF;
WHEN g =>
IF (R ='0') THEN
S<= '1'; estado_siguiente <= a;
ELSE
S<= '0'; estado_siguiente <= a;
          END IF;
 END CASE;
END PROCESS;
END aCAD;
```

La figura 1.26 muestra la simulación del complemento a dos del número 1111, el resultado es 0001. En la figura 1.27 se puede ver la simulación del complemento a dos del número 1011, el bit menos significativo recibe primero el resultado es 0101.

La figura 1.26 muestra la simulación del complemento a dos del número 1111.

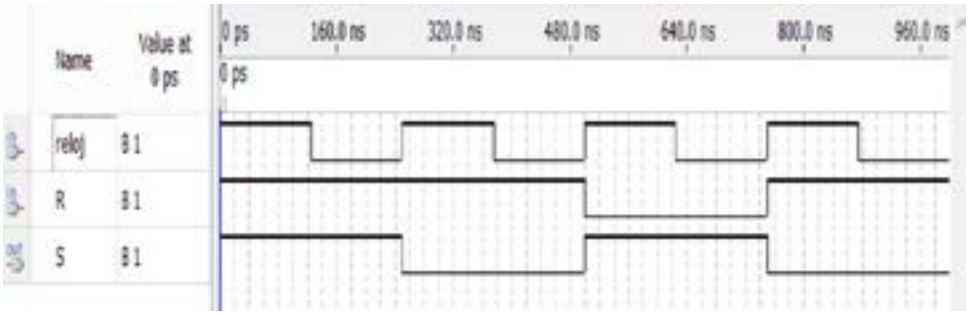


Figura 1.27. Simulación del complemento del número 1101.

Ejemplo 1.8

Muestre las plantillas disponibles en Quartus II para diseñar una máquina Mealy de cuatro estados.

```
-- Quartus II VHDL Template
-- Four-State Mealy State Machine
-- A Mealy machine has outputs that depend on both the state and
-- the inputs.          When the inputs change, the outputs are updated
-- immediately, without waiting for a clock edge.  The outputs
-- can be written more than once per state or per clock cycle.
```

```
library ieee;
use ieee.std_logic_1164.all;
entity four_state_mealy_state_machine is

    port
    (
            clk             : in    std_logic;
            input   : in    std_logic;
            reset   : in    std_logic;
            output  : out   std_logic_vector(1 downto 0)        );
end entity;
architecture rtl of four_state_mealy_state_machine is

    -- Build an enumerated type for the state machine
    type state_type is (s0, s1, s2, s3);

    -- Register to hold the current state
    signal state : state_type;

begin

    process (clk, reset)
    begin

            if reset = '1' then
                    state <= s0;

            elsif (rising_edge(clk)) then
```

```
-- Determine the next state synchronously, based on
-- the current state and the input
case state is
        when s0=>
                if input = '1' then
                        state <= s1;
                else
                        state <= s0;
                end if;
        when s1=>
                if input = '1' then
                        state <= s2;
                else
                        state <= s1;
                end if;
        when s2=>
                if input = '1' then
                        state <= s3;
                else
                        state <= s2;
                end if;
        when s3=>
                if input = '1' then
                        state <= s3;
                else
                        state <= s1;
                end if;
end case;
```

```
            end if;
    end process;


    -- Determine the output based only on the current state
    -- and the input (do not wait for a clock edge).
    process (state, input)
    begin
                case state is
                        when s0=>
                                if input = '1' then
                                        output <= "00";
                                else
                                        output <= "01";
                                end if;
                        when s1=>
                                if input = '1' then
                                        output <= "01";
                                else
                                        output <= "11";
                                end if;
                        when s2=>
                                if input = '1' then
                                        output <= "10";
                                else
                                        output <= "10";
                                end if;
                        when s3=>
```

```
                                    if input = '1' then
                                        output <= "11";
                                    else
                                        output <= "10";
                                    end if;
                        end case;
    end process;

end rtl;
```

Ejemplo 1.9

Muestre las plantillas disponibles en Quartus II para diseñar una máquina Moore de cuatro estados.

```
-- Quartus II VHDL Template
-- Four-State Moore State Machine

-- A Moore machine's outputs are dependent only on the current state.
-- The output is written only when the state changes.  (State
-- transitions are synchronous.)

library ieee;
use ieee.std_logic_1164.all;

entity four_state_moore_state_machine is

    port(
        clk             : in    std_logic;
```

```
        input   : in    std_logic;
        reset   : in    std_logic;
        output  : out   std_logic_vector(1 downto 0)
    );
end entity;

architecture rtl of four_state_moore_state_machine is

    -- Build an enumerated type for the state machine
    type state_type is (s0, s1, s2, s3);

    -- Register to hold the current state
    signal state   : state_type;

begin

    -- Logic to advance to the next state
    process (clk, reset)
    begin
        if reset = '1' then
            state <= s0;
        elsif (rising_edge(clk)) then
            case state is
                when s0=>
                    if input = '1' then
                        state <= s1;
                    else
```

```
                                        state <= s0;
                                end if;
                        when s1=>
                                if input = '1' then
                                        state <= s2;
                                else
                                        state <= s1;
                                end if;
                        when s2=>
                                if input = '1' then
                                        state <= s3;
                                else
                                        state <= s2;
                                end if;
                        when s3 =>
                                if input = '1' then
                                        state <= s0;
                                else
                                        state <= s3;
                                end if;
                end case;
        end if;
end process;
-- Output depends solely on the current state
process (state)
begin
        case state is
```

```
                when s0 =>
                        output <= “00”;
                when s1 =>
                        output <= “01”;
                when s2 =>
                        output <= “10”;
                when s3 =>
                        output <= “11”;
        end case;
    end process;

end rtl;
```

Ejemplo 1.10

Muestre las plantillas disponibles en Quartus II para Safe State Machine

```
-- Quartus II VHDL Template
-- Safe State Machine

library ieee;
use ieee.std_logic_1164.all;

entity safe_state_machine is

    port(
        clk             : in    std_logic;
        input   : in    std_logic;
        reset   : in    std_logic;
```

```
        output   : out    std_logic_vector(1 downto 0)
    );

end entity;

architecture rtl of safe_state_machine is

    -- Build an enumerated type for the state machine
    type state_type is (s0, s1, s2);
    -- Register to hold the current state
    signal state   : state_type;

    -- Attribute "safe" implements a safe state machine.
    -- This is a state machine that can recover from an
    -- illegal state (by returning to the reset state).
    attribute syn_encoding : string;
    attribute syn_encoding of state_type : type is "safe";

begin

    -- Logic to advance to the next state
    process (clk, reset)
    begin
        if reset = '1' then
            state <= s0;
        elsif (rising_edge(clk)) then
            case state is
```

```
                        when s0=>
                                if input = '1' then
                                        state <= s1;
                                else
                                        state <= s0;
                                end if;
                        when s1=>
                                if input = '1' then
                                        state <= s2;
                                else
                                        state <= s1;
                                end if;
                        when s2=>
                                if input = '1' then
                                        state <= s0;
                                else
                                        state <= s2;
                                end if;
                end case;
        end if;
end process;

-- Logic to determine output
process (state)
begin
        case state is
                when s0 =>
```

```
                output <= “00”;
        when s1 =>
                output <= “01”;
        when s2 =>
                output <= “10”;
    end case;       end process;

end rtl;
```

Ejemplo 1.11

Muestre las plantillas disponibles en Quartus II para User-Encoded State Machine

```
-- Quartus II VHDL Template
-- User-Encoded State Machine

library ieee;
use ieee.std_logic_1164.all;

entity user_encoded_state_machine is

    port
    (
        updown          : in std_logic;
        clock    : in std_logic;
        lsb             : out std_logic;
        msb             : out std_logic
    );

end entity;
```

```
architecture rtl of user_encoded_state_machine is

    -- Build an enumerated type for the state machine
    type count_state is (zero, one, two, three);

    -- Registers to hold the current state and the next state
    signal present_state, next_state        : count_state;
    -- Attribute to declare a specific encoding for the states
    attribute syn_encoding                  : string;
    attribute syn_encoding of count_state : type is "11 01 10 00";
begin
    -- Determine what the next state will be, and set the output bits
    process (present_state, updown)
    begin
        case present_state is
            when zero =>
                if (updown = '0') then
                    next_state <= one;
                    lsb <= '0';
                    msb <= '0';
                else
                    next_state <= three;
                    lsb <= '1';
                    msb <= '1';
                end if;
            when one =>
```

```
            if (updown = '0') then
                next_state <= two;
                lsb <= '1';
                msb <= '0';
            else
                next_state <= zero;
                lsb <= '0';
                msb <= '0';
            end if;
        when two =>
            if (updown = '0') then
                next_state <= three;
                lsb <= '0';
                msb <= '1';
            else
                next_state <= one;
                lsb <= '1';
                msb <= '0';
            end if;
        when three =>
            if (updown = '0') then
                next_state <= zero;
                lsb <= '1';
                msb <= '1';
            else
                next_state <= two;
                lsb <= '0';
```

```
                                    msb <= '1';
                                end if;
                        end case;
        end process;
        -- Move to the next state
        process
        begin
                        wait until rising_edge(clock);
                        present_state <= next_state;
        end process;
end rtl;
```

CAPÍTULO 2. PARÁMETROS QUE SE DEBEN TENER EN CONSIDERACIÓN EN EL DISEÑO DE SISTEMAS DIGITALES

2.1 Introducción

Un problema importante de un sistema controlador surge debido a que no puede saber cuándo una o algunas de sus señales de entrada cambiarán, ni cuánto tiempo durará este cambio.

Un sistema controlador bien diseñado debe estar en capacidad de superar este problema.

Un sistema controlador que no tenga adecuadamente sincronizadas sus señales de entrada, puede ocasionar que el sistema presente las siguientes fallas.

- Los cambios muy breves en las señales de entrada no pueden ser detectados por el sistema controlador.
- El aparecimiento de un estado de transición que no haya sido definido.

2.2 Cambios muy breves de las señales de entrada

Este problema ocurre cuando la señal de entrada cambia dentro del período del reloj y su tiempo de cambio es más pequeño que un período de reloj.

La figura 2.1 muestra esta situación, la señal de entrada x, que es asíncrona, cambia dentro del período T del reloj y no pudo ser detectada, pues en el siguiente flanco de subida del reloj la señal X ha vuelto a cero y el sistema controlador no pudo detectar ese cambio, pues solo puede detectar la condición en la que se encuentra una entrada, nivel alto o bajo, cuando el reloj hace su transición, el flanco del reloj.

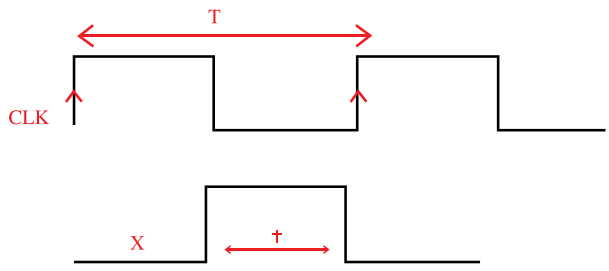


Figura. 2.1. La señal x no sincronizada

Si la señal X tuviera una duración mayor que la duración del período del reloj o si el cambio se hubiera producido en el flanco del reloj. Estas situaciones muestra la figura 2.2.

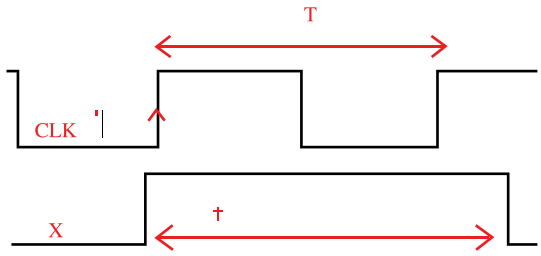


Figura. 2.2. Señal detectada por el sistema controlador

Si la señal X, hubiese retornado a cero luego de haberse producido el flanco de subida del reloj, habría sido detectada sin problema.

De lo anterior se puede concluir que, si el tiempo que se mantiene en alto una señal de entrada, X por ejemplo, es t y el período del reloj es T, entonces debe cumplirse la siguiente relación: $t > T$ y $1/t<1/T$, pero 1/T es la frecuencia del reloj y, por lo tanto, $1/t < f$, donde f es la frecuencia del reloj.

Sin embargo si el tiempo t es muy pequeño la frecuencia del reloj debe ser muy grande para que se cumpla $t<T$. En muchos sistemas, no se puede incrementar indiscriminadamente la frecuencia del reloj. Por lo tanto, esta no es una solución práctica.

Cuando una señal de entrada cambia su estado por un tiempo t menor que el período T del reloj, es necesario sincronizar esta señal para que pueda ser reconocida por el sistema controlador.

El proceso de sincronización puede ser realizado de diferentes maneras. Una manera es utilizando un *latch* básico, cuya función sería atrapar el pulso de corta duración, y un *flip-flop* cuya salida Q se encarga de mantener la señal de entrada, de corta duración hasta que sea reconocida por el sistema controlador.

La figura 2.1 muestra un posible circuito, está compuesto por dos etapas. La primera etapa es un *latch* básico, construido con puertas NAND. S está conectada, a través de un inversor a la señal de entrada que se quiere sincronizar, X en este caso. La segunda etapa contiene un *flip-flop* tipo D, la salida Q de este *flip-flop* tiene la señal X ya sincronizada.

Su funcionamiento es el siguiente: las señales Ql y Q deben estar en cero. Cuando la señal de entrada X es igual a cero SET y RESET son iguales a 1. Cuando la entrada X es igual a uno, SET=0, RESET=1, Ql=1 y Q=0.

Cuando se produce el flanco de bajada del reloj Q=Ql= 1, RESET=0 y SET=0. Con SET=RESET=0, Ql se hace indefinida.

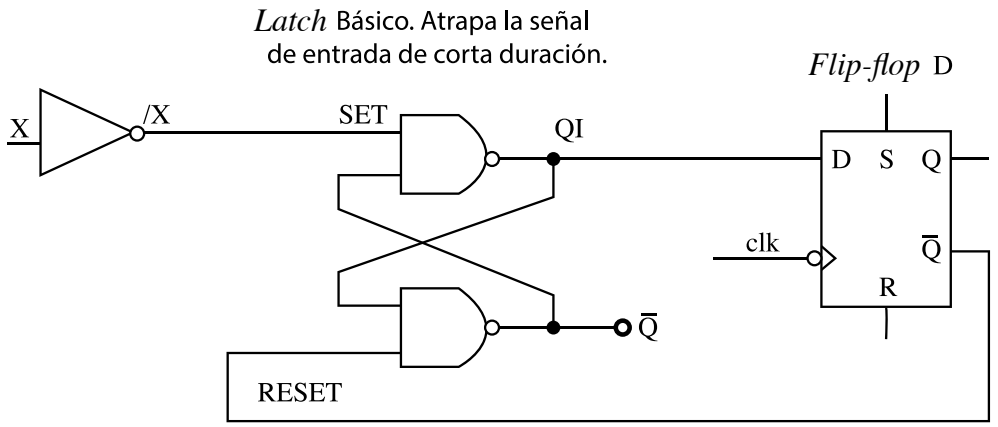


Figura. 2.2. Circuito de sincronización

Cuando X=0, SET=1, RESET=0 y Ql=0, y con el siguiente flanco de bajada Q=0 hasta que la señal de entrada X vuelva a cambiar y el proceso de sincronización inicia otra vez.

La figura 2.2 muestra el diagrama de tiempo del proceso de sincronización de la señal de entrada X. Como puede notarse la salida Q que es la señal X sincronizada esta lista para que pueda ser reconocida, como X=1, por el sistema.

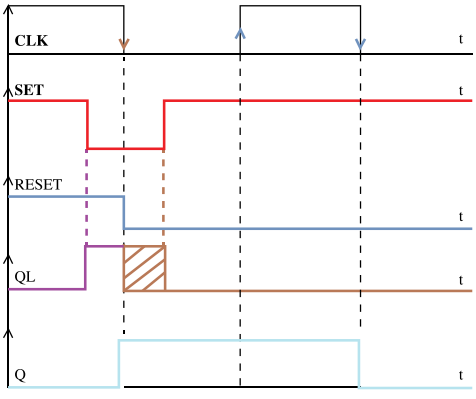


Figura. 2.2. Diagrama de tiempo del proceso de sincronización

La señal RESET que muestra la figura 2.2 puede ser también controlada por el sistema controlador, es decir, retornada a su estado original donde la señal RESET debe ser igual a 1.

La figura 2.3 muestra como la señal RESET puede ser controlada por el sistema controlador.

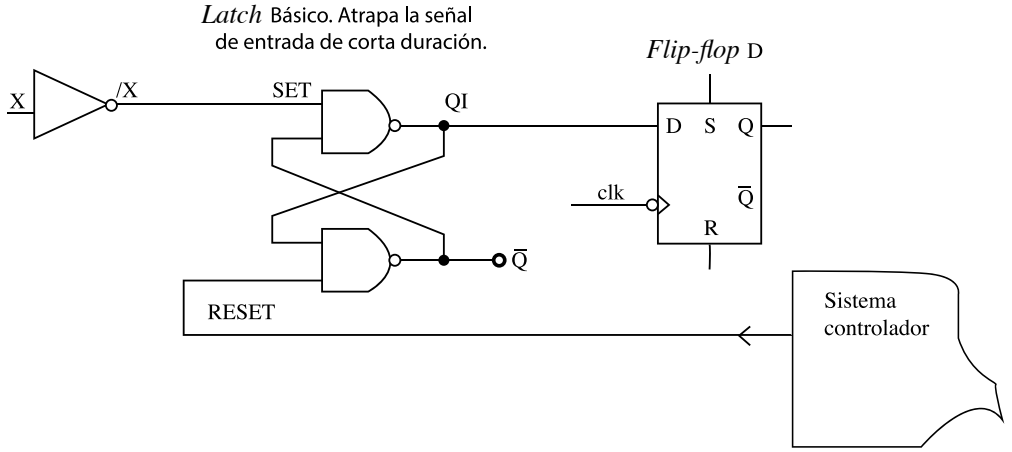


Figura. 2.3. Circuito de sincronización con el RESET controlado por el sistema controlador

A continuación, se presenta el programa en VHDL del circuito de sincronización. Debe tener dos entradas, el reloj y la señal que va a recibir la entrada de muy corta duración, y una salida que es la señal sincronizada.

```
entity DS is
port(
set, clk: in bit;
q: buffer bit
            );
end DS;
architecture aDS of DS is
signal s,Ql,r: bit;
   begin
```

```
        process (set,ql)
                begin
                s<=  not set;
r<= not q;
ql<=(not s )or (ql and r);
end process;
        process(clk,ql )
                begin
                        if (clk'event and clk='1') then
q<=ql;
else
                                null;
end if;
end process;
end aDS;
```

La figura 2.4 muestra la simulación del circuito diseñado. Como se puede ver, en la simulación, la señal de entrada tiene menos duración que el período del reloj. La simulación se realiza para los casos de interés que son:

El punto A, en la figura 2.4, representa a una señal de entrada, no sincronizada, que dura menos que el período del reloj y se genera entre el nivel alto y bajo del reloj, el resultado es una señal sincronizada, q, que dura un período del reloj y que puede ser reconocida sin dificultad por el sistema digital.

El punto B, en la figura 2.4, representa a la señal de entrada que se produce cuando se genera el nivel bajo del reloj, el resultado es una señal, q sincronizada, que dura un período del reloj.

El punto C, en la figura 2.4, representa a la señal de entrada que se produce cuando se genera el nivel bajo del reloj. El resultado es una señal q sincronizada que dura un período del reloj.

El punto E, en la figura 2.4, representa a la señal de entrada que se produce cuando se genera el nivel bajo del reloj. El resultado es una señal q sincronizada, que dura un período del reloj.

El punto F, en la figura 2.4, representa a la señal de entrada que se produce cuando se genera el flanco de subida del reloj, el resultado es una señal q sincronizada, que dura un período del reloj.

El punto G, en la figura 2.4, representa a la señal de entrada que se produce cuando se genera el flanco de bajada del reloj, el resultado es una señal q sincronizada que dura un período del reloj.

De esta manera, se ha demostrado la respuesta del circuito sincronizador para diferentes posiciones de la señal de entrada con relación al reloj. En todos los casos, la respuesta del circuito es la generación de una señal sincronizada que dura un período del reloj. Por lo tanto, este circuito es adecuado para sincronizar señales de corta duración.

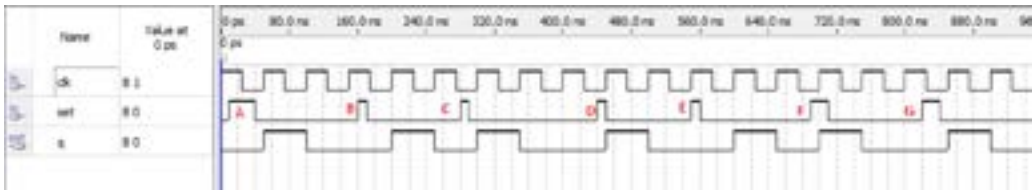


Figura 2.4. Simulación del circuito que sincroniza una señal

Debe notarse que la señal sincronizada se genera en el siguiente ciclo del reloj. Si se desea que la señal sincronizada se genere en el mismo ciclo del reloj y con el primer flanco de subida, entonces, el circuito debe ser sensible al flanco de bajada del reloj, para lograr este fin en el programa en VHDL, anterior, solo hay que cambiar en la línea: if (clk'event and clk='1') el clk de la siguiente manera: if (clk'event and clk='0'). El programa completo se muestra a continuación.

```
entity DS is
port(
set, clk: in bit;
q: buffer bit
                );
end DS;
architecture aDS of DS is
signal s,Ql,r: bit;
```

```
    begin
            process (set,ql)
                    begin
                    s<=  not set;
r<= not q;
ql<=(not s )or (ql and r);
end process;
            process(clk,ql )
                    begin
                            if (clk'event and clk='0') then
q<=ql;
else
                                    null;
end if;
end process;
end aDS;
```

La figura 2.5 muestra la síntesis del circuito de sincronización diseñado.

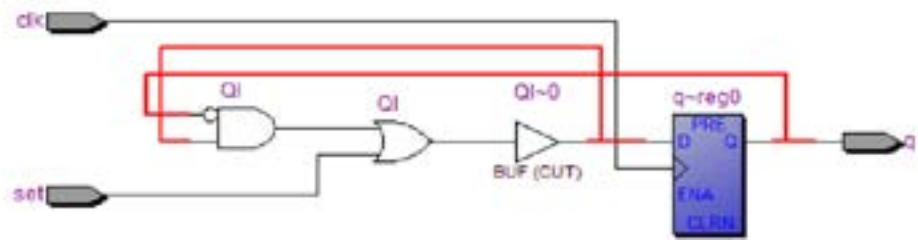


Figura 2.5. Síntesis del circuito sincronizador

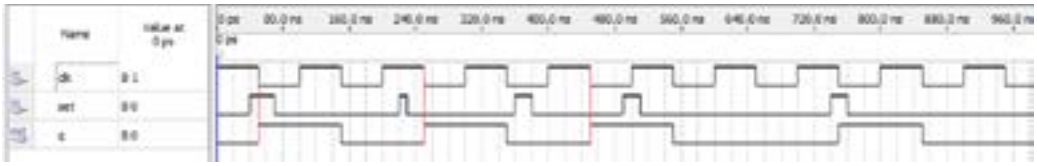


Figura 2.6. Simulación del circuito activo con el flanco de bajada

La figura 2.7 muestra la síntesis del circuito sincronizador, sensible al flanco de bajada del reloj.

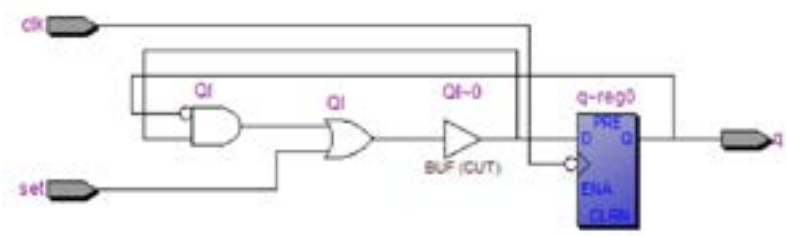


Figura 2.7. Síntesis del circuito sincronizador activo con el flanco de bajada

2. 3 Estados de transición no definidos

Los *flip-flop´s* son los elementos de memoria que, en sus salidas, almacenan los códigos por los cuales una máquina secuencial sincrónica debe pasar. Si más de un *flip-flop* (su salida) debe cambiar para que la máquina vaya de un estado presente a un estado siguiente, debido a que el retardo de propagación de todo *flip-flop* es característico de cada *flip-flop*; es decir, los retardos no son iguales, la máquina puede ir a un estado o estados que no han sido definidos.

En la figura 2.8, se muestran los cambios que podrían ocurrir si la máquina debe pasar del estado presente que tiene el código 111 al estado siguiente cuyo código es 000. Nótese que para realizar este cambio de estado, deberían cambiar los tres bits simultáneamente; sin embargo, por el efecto del retardo de propagación, esto no es posible y solo un bit cambiará a la vez y siempre será, el bit que este en la salida del *flip-flop* más rápido.

La figura 2.4 muestra los nueve estados posibles no definidos por los cuales podría pasar la máquina al pasar de 111 a 000. Por supuesto, de esas nueve po-

sibilidades, solo tres ocurrirían. Por ejemplo, del estado 111 podría pasar al 110 luego al estado 100 y finalmente al estado 000, asumiendo que el *flip-flop* C es más rápido que el B y el B más rápido que el A, se supone por supuesto, que hay tres *flip-flops* con sus salidas denominadas como ABC, A es el más significativo.

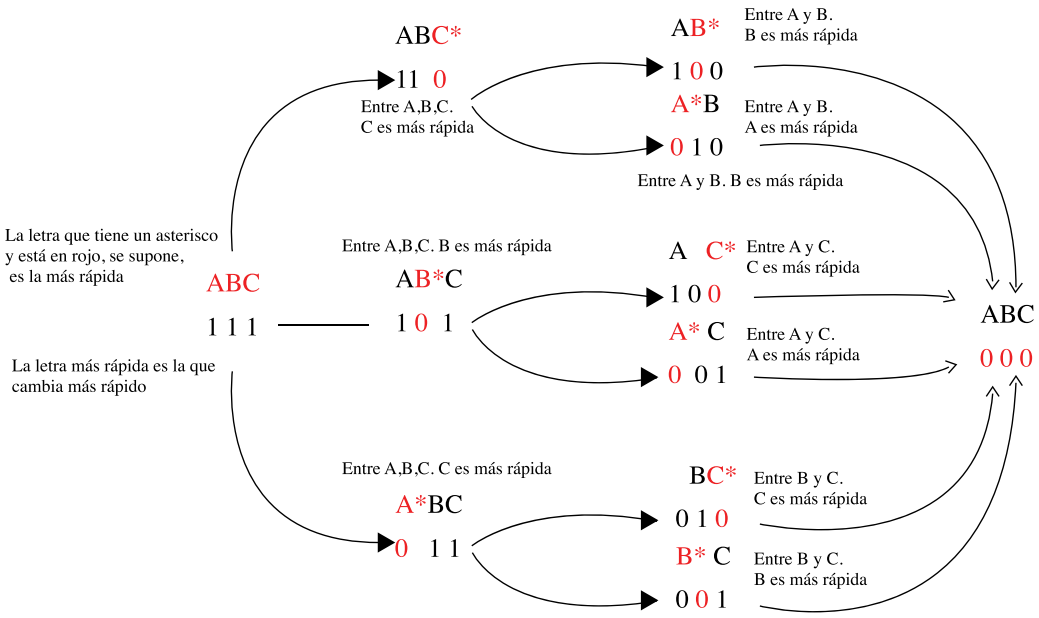


Figura 2.8. Posibles estados de transición

Si entre un estado presente y un estado siguiente cambia un solo bit este problema no ocurriría, sin embargo, esto en la práctica no es posible, pues, de un estado presente, la máquina casi siempre tendrá varios estados siguientes, por lo tanto, hacer que solo un bit cambie entre un estado presente y varios siguientes es bastante complicado y se complica a un más mientras más estados siguientes existan.

En el caso muy especial cuando, de un estado presente, se tenga un solo estado siguiente, la asignación de un código donde solo un bit cambie es bastante sencillo, pues, se podría utilizar un código de distancia unitaria como el Gray. A continuación se muestra este código para cuatro bits.

ABCD

0 0 0 0

0 0 0 1

0 0 1 1

0 0 1 0

0 1 1 0

0 1 1 1

0 1 0 1

0 1 0 0

1 1 0 0

1 1 0 1

1 1 1 1

1 1 1 0

1 0 1 0

1 0 1 1

1 0 0 1

1 0 0 0

Al examinar este código, se puede notar que efectivamente recorriendo el código en el orden que está escrito un solo bit cambia a la vez e incluso si del último código, 1000, se retorna al código primero, 0000, un solo bit cambia.

La asignación de un código a cada estado está íntimamente relacionado con la complejidad del decodificador de salidas, así como también, por los posibles estados no definidos por los que la máquina podría pasar.

2.4 El decodificador de salidas y la asignación de código

Independiente de si se trata de una máquina Mealy o Moore, el decodificador de salidas está en función del código que se asigne a cada estado porque es función en el primer caso de las entradas y del estado y, para el segundo, solo del estado.

Si hipotéticamente, para el caso Moore, por ejemplo, el código de las salidas al mundo exterior son exactamente igual al código de cada estado, el decodificador de salidas prácticamente sería los cables que van de las salidas de los *flip-flops* al mundo exterior.

Por ejemplo, un contador binario de dos bits cuyo diagrama de estados, para una máquina Moore, muestra la figura 2.9.

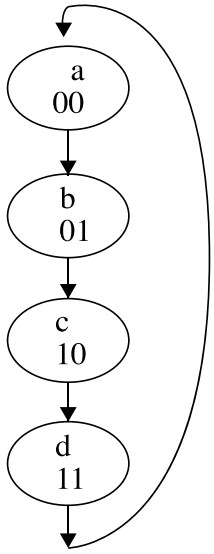


Figura. 2.9. Diagrama de estados de un contador de dos bits

Como se puede ver en el diagrama de estados, el código de cada estado es el mismo de las salidas del contador, por lo tanto, el decodificador de salidas no existe.

El diagrama de estados para el mismo contador, pero para una máquina Mealy, se muestra en la figura 2.10. Para este caso, las salidas siguen teniendo el mismo código de cada estado, por lo que el decodificador de salidas tampoco existe.

En general, cuando el código de cada estado coincide con el código de las salidas, el decodificador de salidas no existe. Por otro lado cuando se asigna el código a cada estado y se trata de que este código sea lo más parecido (difieren en pocos bits, 110, 111) al código de las salidas, entonces el decodificador de salidas será siempre más simple. Por el contrario, si el código de los estados es

totalmente diferente (difieren en muchos bits, 1111, 0000) al código de las salidas el decodificador de salidas será mucho más complejo.

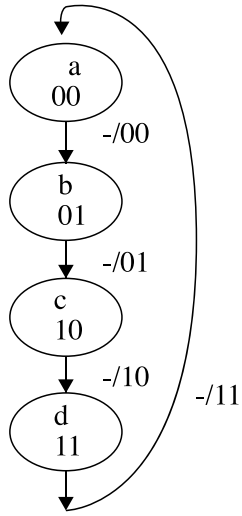


Figura 2.10. Diagrama de estados, del contador de dos bits para una máquina Mealy

Hay una asignación de código de estado conocido como: UNO ACTIVO (*one-hot encoding*), que consiste en que cada estado tenga tantos bits (*flip-flops*) cuantos estados diferentes tenga el diagrama. Por ejemplo, si hay cuatro estados diferentes, debe haber cuatro bits para codificar cada estado. El bit activo, caliente, es el que tiene el valor de uno. Esta técnica incrementa el número de *flip-flops* pero se podría tener un decodificador de salidas mucho más simple.

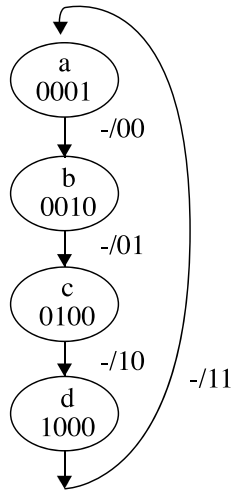


Figura 2.11. Codificación 1 activo

Para el caso del contador de dos bits, el diagrama de estados se muestra en la figura 2.11. Se requieren de cuatro *flip-flops* en lugar de dos.

2.5 Desviación del reloj

La desviación del reloj ocurre cuando se tiene un banco de *flip-flops* que comparten el mismo reloj y deben recibir, todos y al mismo tiempo, la misma señal. Pero ocurre que alguno o algunos de estos *flip-flops* no reciben el reloj al mismo tiempo. Este problema se debe a que el retardo de propagación desde la fuente del reloj a cada reloj de cada *flip-flop* no es la misma, la figura 2.12 muestra este fenómeno.

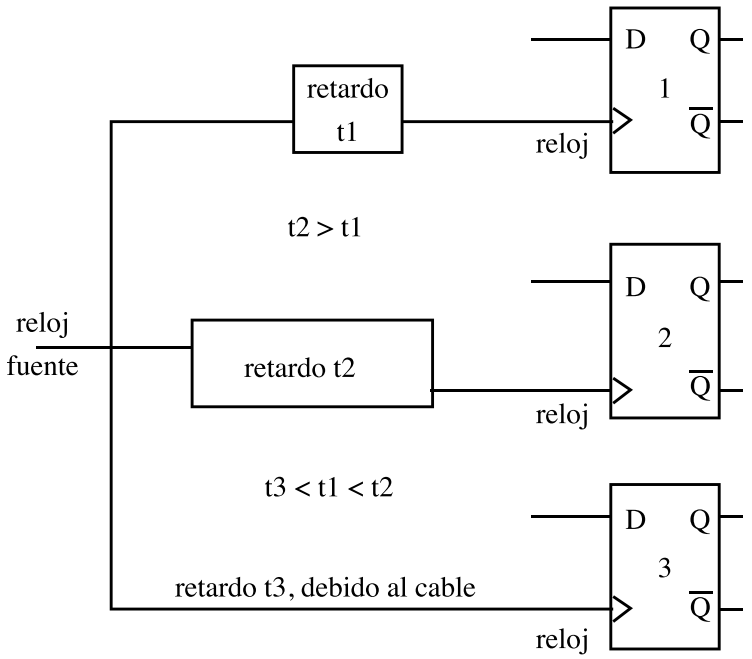


Figura 2.12. Sistema con desviación del reloj

Los retardos son producidos por los circuitos lógicos y por el cable que conecta los diferentes dispositivos. El la figura 2.12, se puede notar que los relojes de los *flip-flops* reciben la señal del reloj fuente a distintos tiempos. El reloj del *flip-flop* 1 va a recibir la señal más pronto que el *flip-flop* 2 y el *flip-flop* 3. En el

diagrama de tiempo que muestra la figura 2.9, se puede ver el desfase con que el reloj fuente llega a cada *flip-flop*.

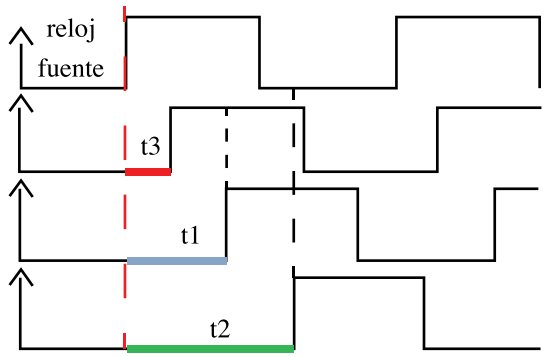


Figura 2.13. Diagrama de tiempo que muestra la desviación del reloj

El primer flanco de subida del reloj llega al *flip-flop* 3, primero, porque el cable representa el menor retardo, t3. Luego, ese mismo flanco del reloj llega el *flip-flop* 1 y finalmente al *flip-flop* 2.

El efecto de la no sincronía del reloj sobre las salidas Q de los *flip-flop´s* significa que no se actualizan al mismo tiempo. Además, cada *flip-flop* tiene su propio retardo.

Para disminuir el efecto de la desviación del reloj se debe evitar conectar dispositivos a los relojes de los *flip-flop´s*. Debe observarse que los mayores retardos los producen los circuitos lógicos conectados a los relojes.

2.6 Red de distribución del reloj

Para disminuir el efecto del retardo producido por los cables se utiliza una red de distribución del reloj en árbol H, (se llama H porque se parece esta letra), que muestra la figura 2.14. Como se puede observar, la distancia que debe recorrer la señal del reloj hasta llegar a cada *flip-flop* es la misma cuando los relojes están conectados a esta red.

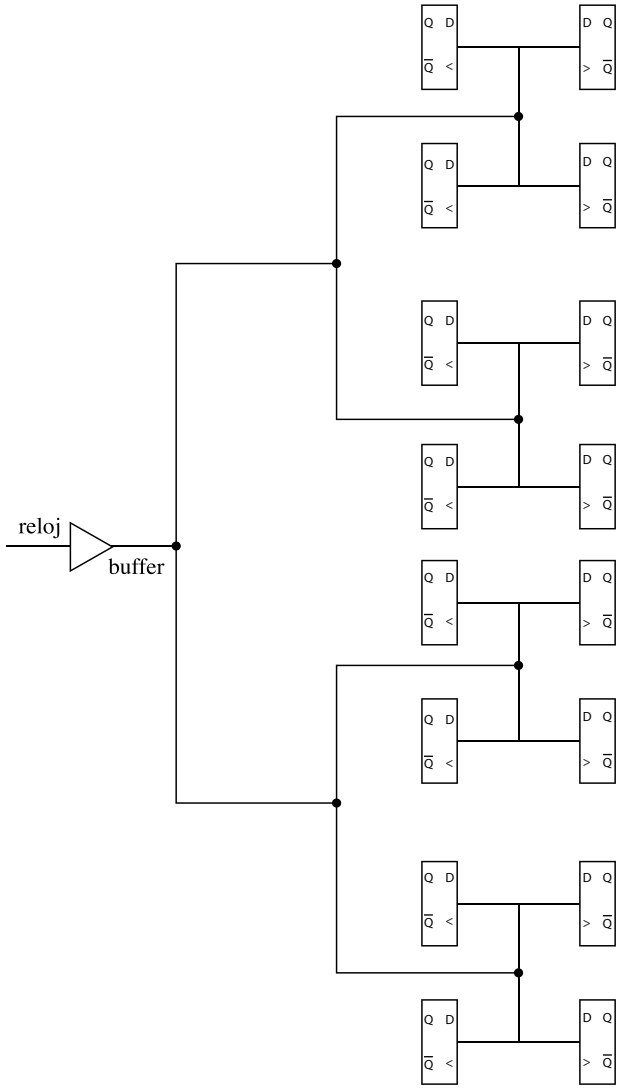


Figura 2.14. Red de distribución del reloj en árbol H

2. 7 Sincronización de señales con *flip-flop*

Para que una señal que se aplica a la entrada de un *flip-flop* sea reconocida adecuadamente, esta señal, debe permanecer estable por un tiempo, denominado tiempo de preparación (tp), antes de que se produzca el flanco del reloj y debe también mantenerse estable luego de que se haya producido el flanco del reloj por un tiempo denominado tiempo de espera (te). La figura 2.15 muestra un diagrama donde se puede ver el tiempo de preparación y el de espera.

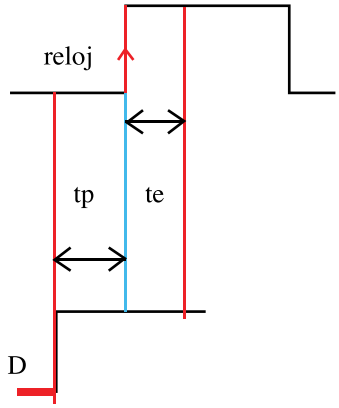


Figura 2.15. Tiempos de preparación y espera

Con la finalidad de asegurar que una señal asincrónica no rompa los tiempos tp y te, una señal se sincroniza como muestra la figura 2.16.

Como se puede ver, son dos *flip-flops* conectados en cascada y al mismo reloj, esta conexión asegura que, si la señal aplica al primer *flip-flop* produce una salida metaestable (A). Tendrá el tiempo suficiente para estabilizarse y luego pasar a la salida del segundo *flip-flop* completamente estable.

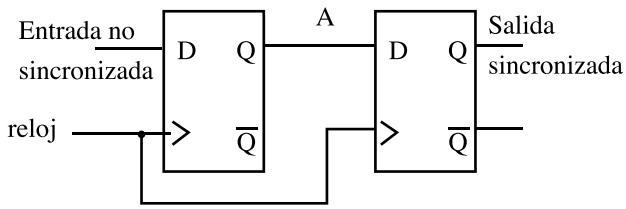


Figura. 2.16. Circuito para sincronizar señales.

La frecuencia del reloj debe ser la adecuada para permitir que la salida A salga de su estado metaestable.

2.8. Ejercicios propuestos

1. ¿Qué es un sistema controlador?

2. ¿Por qué un sistema controlador sincrónico debe tener un reloj y qué parámetros dependería la frecuencia máxima de ese reloj?.

3. ¿Existiría alguna condición o condiciones bajo las cuales un interruptor conectado a un sistema controlador no pueda ser reconocido por este? Explique.

4. Una señal de salida de un sistema controlador dura un ciclo de reloj o un período del mismo. ¿Se podría hacer que esta señal dure solo una parte del ciclo del reloj? Explique.

5. Un interruptor conectado a un sistema controlador es un dispositivo que trabaja en sincronía con el reloj. Explique.

6. ¿Se podrían consultar por el estado de dos interruptores simultáneamente en un estado cualquiera?

7. Un sistema controlador puede tener entradas que trabajen en sincronía con el reloj y en asincrónica con el mismo. Explique.

8. ¿Cuáles serían las consecuencias para el sistema controlador si este tiene elementos de memoria (que guardan el código del estado) que se activen por el nivel del reloj y no por el flanco?

9. ¿Cuánto tiempo dura un estado en un diagrama ASM y cuánto tiempo dura el código de un estado en los elementos de memoria?

10. ¿Qué es una partición funcional?

11. ¿Qué es un algoritmo?

12. ¿Qué es un diagrama de flujo?

13. De ser el caso, un ingeniero podría crear sus propios símbolos para graficar un algoritmo de *hardware* (algo parecido a un diagrama ASM) ¿Qué consecuencias podría tener esta actitud? Explique

14. ¿Cuál es la diferencia entre un diagrama de flujo y un diagrama ASM?

15. Explique los elementos que conforman un diagrama ASM.

16. ¿Cuál es la sintaxis para declarar datos enumerados?

17. Explique el código en VHDL que permite pasar de un estado a otro.

18. En un estado cualquiera, si se consulta por dos entradas, ¿debe necesariamente consultarse por los cuatro posibles valores de las entradas o solo por algunos de ellos? Explique.

19. En un diagrama ASM de un estado cualquiera (el rectángulo) se puede ir a consultar por el estado de una señal de entrada (diamante) que está en cualquier otro estado del diagrama ASM o necesariamente debe irse a un estado. Es decir, ¿se puede ir de un rectángulo a un diamante que pertenece a otro estado, o solo de rectángulo a rectángulo? Explique.

20. Explique el código en VHDL que permite recorrer, estado por estado un diagrama ASM.

21. Explique cuál es la diferencia, en caso de que exista, entre el estado siguiente y el estado presente en un diagrama ASM.

22. Si en un estado de un diagrama ASM se debe preguntar por el estado de dos señales, ¿la consulta debe hacerse en forma independiente a cada señal o se consulta el estado de las señales simultáneamente? Explique.

23. ¿Cuál es la función del *reset* en un sistema controlador? Explique mediante un diagrama ASM.

24. ¿Cuál es la diferencia al declarar una señal de entrada como de tipo boolean o de tipo std_logic.? Explique desde el punto de vista de facilidad de codificación en VHD?.

25. Se requiere diseñar una cerradura digital. La cerradura se abre cuando el usuario a tecleado la clave correcta, si ha ingresado una clave incorrecta, el sistema debe ir a un estado de falla y permanecer en ese estado hasta que el usuario presione una tecla de desbloqueo luego de lo cual el sistema estará listo para recibir una clave. Si el usuario equivoca la clave en tres intentos el sistema se irá a un estado de alerta y permanecerá en ese estado hasta que el usuario presione dos teclas de desbloqueo.

Elabore la partición funcional, el diagrama ASM y escriba el programa en VHDL.

La clave queda a elección del diseñador, al igual que las teclas de desbloqueo y cualquier otro detalle que crea necesario para este diseño.

26. Diseñe, utilizando la técnica de diagramas de estado, un contador binario de cuatro bits. Este contador debe tener una señal de *reset* sincrónica.

27. Diseñe un generador de cuatro ondas cuadradas. Una onda debe tener una frecuencia que sea igual a la mitad de la frecuencia del reloj, la otra debe tener una frecuencia igual a la cuarta parte de la frecuencia del reloj y la última, la octava parte de la frecuencia del reloj.

28. Modifique el diagrama ASM de la máquina vendedora de refrescos para que una parte del módulo de recepción de monedas este integrado en el sistema controlador.

29. Modifique el diagrama ASM de la máquina vendedora de refrescos para que el usuario tenga la posibilidad de elegir un refresco de entre cuatro tipos.

30. Escriba el programa en VHDL para los diagramas ASM de los ejercicios 28 y 29.

31. Diseñe un sistema que permita a un robot salir de un recinto lleno de obstáculos. El robot tiene un sensor de acercamiento que le envía una señal cuando está muy cerca de un obstáculo. El robot puede girar a la izquierda, a la derecha, avanzar y retroceder. Hay una sola puerta de salida. El robot tiene tres llantas, dispuestas en forma triangular, una en la parte delantera y que está ubicada en el vértice del triángulo que apunta hacia delante y dos en la parte posterior y ubicada frente a frente en los otros dos vértices del triángulo. Las llantas de la parte posterior tienen un motor cada una, y se activan y desactivan en forma independiente.

El robot avanza o retrocede si las dos llantas posteriores giran al mismo tiempo y en el mismo sentido hacia adelante o en reversa respectivamente. El robot gira a la izquierda cuando el motor de la llanta izquierda está detenido y el motor de la llanta derecha está activado; y gira a la derecha cuando el motor de la llanta derecha está detenido y el motor de la llanta

izquierda, activado. Elabore la partición funcional, el diagrama ASM y el programa en VHDL.

32. Explique ¿por qué hay que sincronizar una señal?

33. ¿Cuál de los dos circuitos de sincronización, el sensible al flanco de subida o el sensible al flanco de bajada, es el más adecuado para sincronizar señales? Discuta.

34. Si una señal de entrada a un sistema digital sincrónico tiene una duración de 20 milisegundos, ¿cuál debe ser la frecuencia mínima del reloj del sistema para que esa señal de entrada pueda ser detectada sin problema?

35. Escriba un programa en VHDL que permita sintetizar el circuito que se muestra en la figura 1.

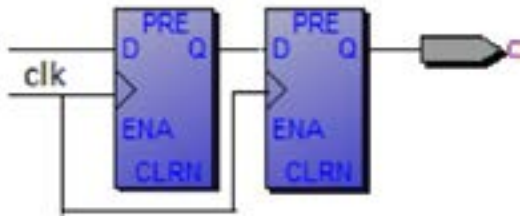


Figura 1. Circuito para el ejercicio

36. Haciendo referencia al ejercicio anterior, explique cuál es la función del circuito.

37. Utilizando un registro de desplazamiento universal de ocho bits, diseñe un sistema que obtenga el complemento a dos de un número de ocho bits que debe ser ingresado por las entradas paralelas del registro. El complemento a dos del número debe ser almacenado en las ocho salidas del registro. Elabore la partición funcional y diseñe el sistema controlador utilizando VHDL. La operación del registro de desplazamiento muestra la tabla siguiente.

S0	S1	Acción
0	0	Mantiene el estado
0	1	Desplaza a la derecha un bit
1	0	Desplaza a la izquierda un bit
1	1	Realiza una carga paralela

Tabla 1. Operación del registro

Utilice todos los circuitos combinaciones y secuenciales que considere necesarios para elaborar la partición funcional.

El registro de desplazamiento tiene una señal de *clear* (aclarado) que asincrónicamente pone a cero sus salidas. Asuma que el número, de ocho bits, del que se obtendrá el complemento a dos es generado por algún sistema que está en sincronía con el sistema que va a diseñarse.

BIBLIOGRAFIA

Volnei, A. (2004). *Circuit Design and Simulation with VHDL*. Cambridge, Massachusetts: MIT Press.

Maxinez, J. y Alcalá, L. (2003). *VHDL, el arte de programar sistemas digitales*. México DF: CECSA.

Chu, P. (2006). *RTL Hardware Design Using VHDL*. Nueva Jersey: John Wiley & Sons.

Brown, S., y Vranesic, Z. (2006). *Fundamentos de lógica digital con diseño VHDL*. México DF: Mc Graw Hill.

En general, cuando se va a diseñar un sistema digital, se debe, en primer lugar, analizar y definir el tipo de circuito que se requiere, es decir, si es un circuito combinacional o un circuito secuencial.
Si es combinacional, entonces, la relación que existe entre las señales de entrada y salida (la función de transferencia) está dada por una tabla de verdad y, por lo tanto, la técnica de diseño consiste en la elaboración de una tabla de verdad que relaciona las entradas con las salidas.
Si se trata de un circuito secuencial, el algoritmo que establece la relación entre las entradas y las salidas se representa en forma gráfica, por ejemplo, mediante un diagrama de estados, un diagrama MDS o un diagrama ASM. Dicho de otra manera, la técnica de diseño de este tipo de circuitos es mediante algún diagrama.
Los diagramas de estado se utilizan para diseñar máquinas tipo Mealy y Moore que normalmente no tienen muchas entradas y tampoco muchas salidas; es decir, son sistemas pequeños. Una técnica para diseñar sistemas digitales grandes y complejos es mediante los diagramas o cartas ASM ASM significa máquina de estado algorítmica y estas siglas tienen su origen en Algorithmic State Machine.
En general, un sistema digital grande y complejo puede considerarse como compuesto por dos grandes bloques que interactúan entre sí. Uno de ellos es el sistema controlador y el otro es el subsistema (formado por bloques combinacionales y/o secuenciales). El propósito de este libro es el diseño de este tipo de sistemas.

Wilson Oswaldo Baldeón López es ingeniero electrónico graduado en la Escuela Superior Politécnica del Litoral; es máster en Informática graduado en Chile; es máster en Diseño de Sistemas Electrónicos por la Universidad Tecnológica de La Habana, Cuba; es magíster en Gestión Académica Universitaria, tiene un diplomado superior en Pedagogía Universitaria y es experto en procesos *e-learning*.

Verónica Elizabeth Mora Chunllo es ingeniera en electrónica y computación graduada en la Escuela Superior Politécnica de Chimborazo; magíster en Ingeniería de Software graduada en la Escuela Superior Politécnica del Ejército; es máster en Diseño de Sistemas Electrónicos por la Universidad Tecnológica de La Habana, Cuba; es magíster en Educación a Distancia, tiene un diplomado superior en las Nuevas Tecnologías de Información y Comunicación Aplicadas a la Práctica Docente, es experta en procesos *e-learning*.

ISBN: 978-9942-35-651-2