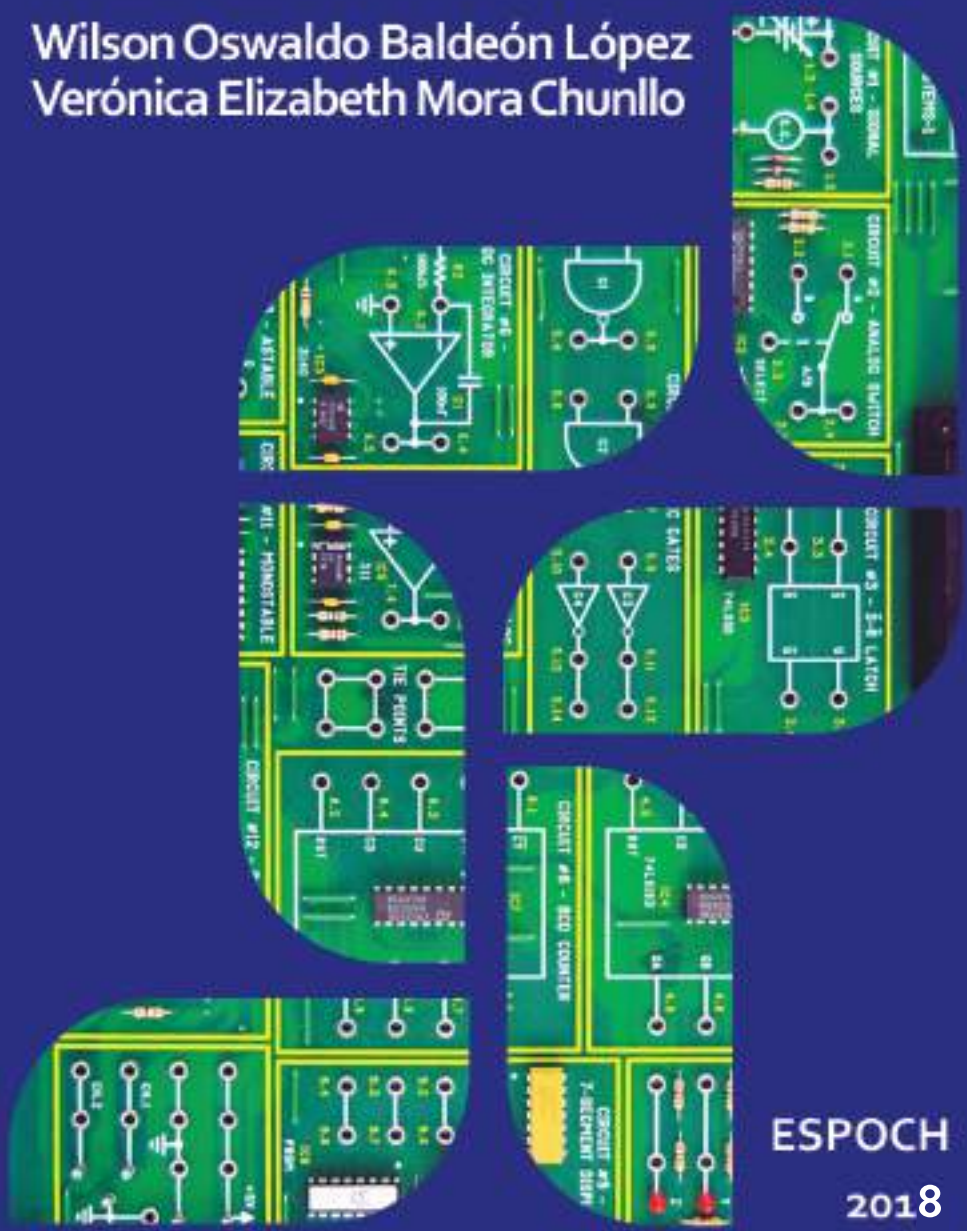
Sistemas digitales sincrónicos y VHDL

Diseño de circuitos secuenciales

Wilson Oswaldo Baldeón López
Verónica Elizabeth Mora Chunllo



ESPOCH
2018

SISTEMAS DIGITALES SINCRÓNICOS Y VHDL
DISEÑO DE CIRCUITOS SECUENCIALES

SISTEMAS DIGITALES SINCRÓNICOS Y VHDL
DISEÑO DE CIRCUITOS SECUENCIALES

WILSON BALDEÓN
VERÓNICA MORA

DIRECCIÓN DE PUBLICACIONES

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO
Fundada en 1972
Riobamba - Ecuador

SISTEMAS DIGITALES SINCRÓNICOS Y VHDL
DISEÑO DE CIRCUITOS SECUENCIALES

Panamericana Sur, kilómetro 1 ½
Dirección de Publicaciones Científicas
Riobamba, Ecuador
Teléfono: (593 3) 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 CIRCUITOS SECUENCIALES
Riobamba: Escuela Superior Politécnica de Chimborazo
Dirección de Publicaciones, 2018
119 pp. vol: 17 x 24 cm
ISBN: 978-9942-35-648-2
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; concluyó su máster en Diseño de Sistemas Electrónicos por la Universidad Tecnológica José Antonio Echeverría; 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; concluyó su máster en Diseño de Sistemas Electrónicos por la Universidad José Antonio Echeverría; 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 segundo 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. La sociedad del siglo XXI es una sociedad digital. Los dispositivos electrónicos digitales están presentes en todas las actividades humanas, de ahí su importancia y la necesidad de profesionales diseñadores de sistemas digitales que dominen las técnicas de análisis y diseño de sistemas digitales, así como también, las aplicaciones informáticas que permiten el diseño asistido por computadora (CAD) y un lenguaje de descripción de *hardware*.

La ciencia en la que se fundamenta los sistemas digitales, en lo relativo a sus conceptos y fundamentos, no ha sufrido un cambio radical todavía, sin embargo, el diseño y aplicación práctica de esta ciencia (la implementación) ha cambiado sustancialmente en las últimas décadas con la aparición de los lenguajes de descripción de *hardware* (HDL) y herramientas CAD.

Este libro expone los conceptos fundamentales de los diagramas de estado desde una perspectiva clásica, pues su fundamento científico no ha cambiado. El diseño mediante HDL y herramientas CAD es tratado en el tercer libro de esta obra.

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 su contenido de una forma que el lector pueda desarrollar habilidad intuitiva para entender y aplicar los conceptos fundamentales, la estructura, el análisis y diseño de máquinas secuenciales sincrónicas. Se exponen ejemplos que permiten reforzar los conocimientos que el lector va adquiriendo.

1. DISEÑO DE CIRCUITOS SECUENCIALES SINCRÓNICOS

1.1 Introducción

Una vez estudiados los *latch* básicos, los *latch* asincrónicos y los *flip-flops*, que son los elementos fundamentales a partir de los cuales se construyen máquinas secuenciales sincrónicas, el siguiente paso es el estudio de las técnicas de análisis y diseño de máquinas o circuitos secuenciales sincrónicos y asincrónicos.

Una máquina secuencial, en general, se caracteriza porque los valores que se encuentran presentes en sus salidas, en algún instante, dependen no solamente de los valores que se encuentran presentes en sus entradas en ese instante, sino también, de todos los valores que estuvieron en esas entradas, es decir, de la historia pasada de esas entradas, valga la redundancia.

Una máquina se llama secuencial porque, tiene que pasar, paso a paso, por un conjunto de estados. Si el paso de un estado a otro esta sincronizado por una señal de reloj, la máquina se llama secuencial sincrónica. Si la máquina no tiene una señal que sincronice los cambios de estado sino más bien los cambios de estado son realizados en el instante que alguna de sus señales de entrada ha cambiado, la máquina se llama secuencial asincrónica.

Los circuitos secuenciales sincrónicos se suelen clasificar de la siguiente manera:

- Generadores y detectores de código
- Contadores y registros
- Sistemas controladores multientrada

Las máquinas secuenciales tienen una estructura muy bien definida. Esta estructura es su arquitectura o diagrama general de bloques.

1.2 Clases o tipos de máquinas secuenciales

Los diferentes tipos de máquinas secuenciales tienen su origen en la arquitectura general de una máquina secuencial, que se muestra la figura 1.1

De la figura 1.1, se puede observar que una máquina secuencial está compuestas por tres bloques:

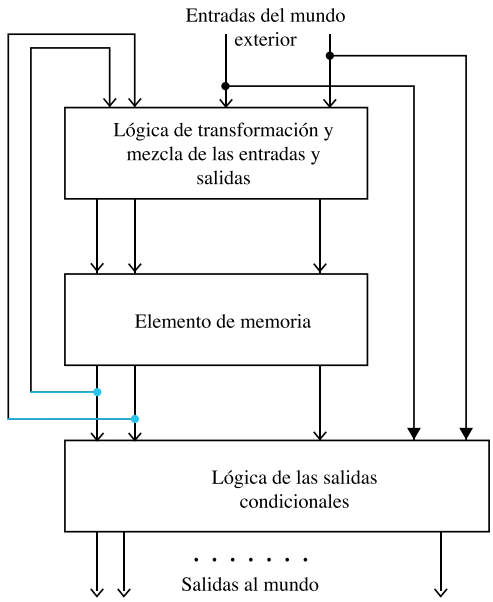


Figura 1.1 Diagrama de bloques general de una máquina secuencial

- El bloque de lógica de transformación y mezcla de entradas y salidas.
- El bloque de memoria, construido por *flip-flops* o por retardos de propagación.
- El bloque de la lógica de las salidas condicionales.
- El bloque de memoria puede estar formado por: un dispositivo físico, como por ejemplo *flip-flops*, o por una capacidad de memoria.

La capacidad de memoria, que lógicamente no es un dispositivo físico, se manifiesta debido a que toda compuerta tiene un efecto de memoria debido a su retardo de propagación, es decir, la memoria (virtual) es el retardo de propagación.

Las máquinas secuenciales sincrónicas suelen clasificarse en:

- Clase A o máquina Mealy
- Clase B o máquina Moore
- Clase C o máquina Moore sin decodificador de salidas

- Clase D o memoria de *look up*
- Clase E

1.2.1 Máquina Mealy o clase A

El diagrama general de bloques de una máquina Mealy se muestra en la figura 1.2. Como se puede apreciar, es el mismo bloque de la arquitectura general de una máquina secuencial indicada en la figura 1.1, con la diferencia de que está claramente indicado que el bloque de memoria está compuesto por *flip-flops*.

En esta máquina, el bloque de lógica de transformación y mezcla de entradas y salidas o decodificador de estado siguiente, recibe las señales de entrada del mundo exterior y las señales del estado presente de la máquina (salidas del elemento de memoria), procesa estas señales y genera el código del estado siguiente, estado, al que irá la máquina luego de que el reloj reciba su flanco de subida (o bajada si es sensible al flanco de bajada).

El decodificador de salidas tiene, como entradas, las señales del mundo exterior, las salidas del elemento de memoria (código del estado presente) y como salidas las salidas al mundo exterior.

El bloque de memoria está conformado por *flip-flops*, en las salidas de los *flip-flops* (las salidas Q), se encuentra guardado temporalmente el código del estado presente de la máquina y, en sus entradas, el código del estado siguiente. El cambio del código del estado presente al código del estado siguiente se da cuando se genera el flanco del reloj.

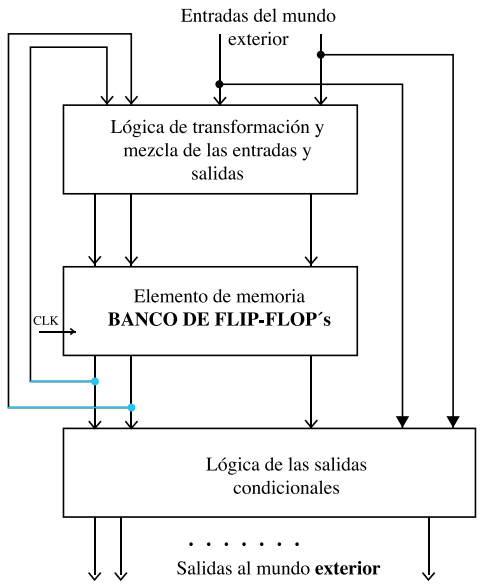


Figura 1.2. Diagrama de bloques de una máquina Mealy o clase A

Cuando el reloj de los *flip-flops* está en su estado verdadero, el código del estado presente se actualiza.

La diferencia sustancial entre una máquina Mealy y los otros tipos de máquinas está en que las salidas al mundo exterior están condicionadas o dependen directamente de las entradas del mundo exterior; por consiguiente, si una entrada del mundo exterior hace un cambio no deseado, por ejemplo, por ruido, las salidas también van a cambiar sin importar el estado del reloj. Debe recordarse que las salidas deben cambiar solo cuando se ha producido el flanco del reloj y deben mantenerse sin cambiar durante todo el período del reloj.

Como se verá más adelante, en los otros tipos de máquinas, las salidas al mundo exterior no dependen, para nada, de las entradas del mundo exterior, sino solo del estado presente de la máquina.

1.2.2 Máquina Moore o clase B

El diagrama general de bloques de una máquina Moore se muestra la figura 1.3. Como se puede ver, la diferencia con la máquina Mealy está en que las entradas del mundo exterior no se conectan al bloque de la lógica de las salidas condicionales.

El bloque de lógica de transformación y mezcla de entradas y salidas o decodificador de estado siguiente, es el mismo bloque que el de la máquina Mealy.

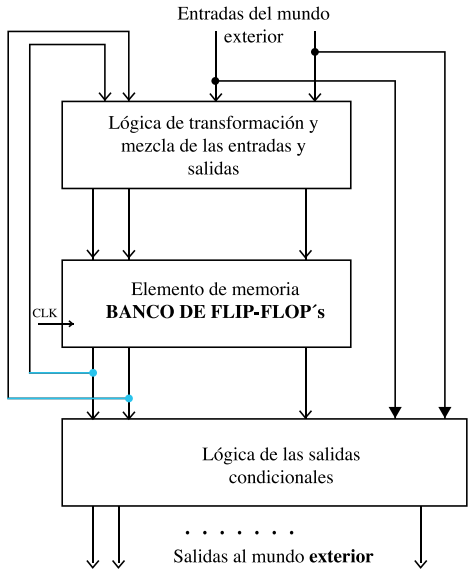


Figura 1.3. Diagrama de bloques de una maquina clase B o Moore

En las máquinas Moore, el decodificador de salidas tiene como entradas exclusivamente las salidas del elemento de memoria. Por lo tanto, las salidas al mundo exterior dependen solo del estado de la máquina.

1.2.3 Máquina Moore sin decodificador de salidas o máquina clase C

El diagrama de bloques de una máquina Clase C muestra la figura 1.4. Como se puede ver, en este tipo de máquinas, las señales de salida al mundo exterior son las mismas salidas de los *flip-flops*.

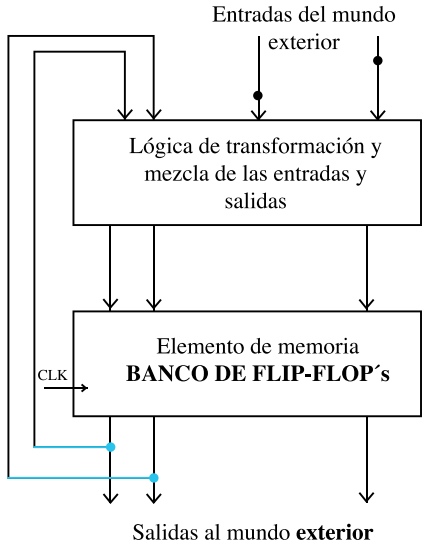


Figura 1.4 Diagrama de bloques de una máquina clase C

El bloque de decodificador de siguiente estado o de lógica de transformación y mezcla de las entradas y salidas, es el mismo que el de la máquina Mealy o Moore.

1.2.4 Máquina clase E

El diagrama de bloques de una máquina Clase E se muestra en la figura 1.5. No hay decodificador de salidas ni decodificador de estado siguiente. Por lo tanto, las salidas del elemento de memoria son las salidas al mundo exterior.

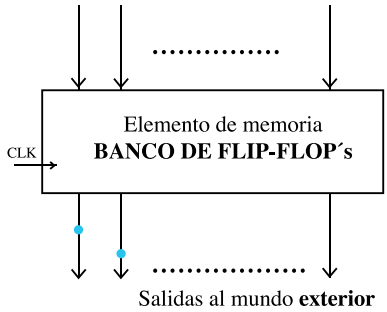


Figura 1.5. Diagrama de bloques de una máquina clase E

Las entradas al elemento de memoria son las entradas del mundo exterior, como se puede ver en la figura 1.5, estas máquinas son precisamente los *flip-flops*.

1.3 Diseño de circuitos secuenciales sincrónicos

1.3.1 Introducción

Cuando se requiere diseñar un circuito digital combinacional o secuencial es necesario aplicar algún algoritmo, alguna técnica de diseño. Para el diseño de circuitos combinacionales la técnica es mediante la elaboración de una tabla de verdad que relaciona las salidas con las entradas del circuito, es decir, la función de transferencia del circuito está dada por una tabla de verdad. En el caso de los circuitos secuenciales, una de las técnicas de diseño es el diagrama de estados.

1.3.2 El diagrama de estados

Con el fin de evitar definiciones complicadas y confusas, un diagrama de estados se define, en este libro, sencillamente como: "*un gráfico formado por símbolos especiales (óvalos o círculos) y segmentos de recta dirigidos, que describen gráficamente el comportamiento o funcionamiento de un circuito o máquina secuencial*".

El proceso de diseño de un circuito secuencial, mediante la técnica de diagramas de estado, se puede resumir en los siguientes términos:

- Se analiza las especificaciones que debe cumplir el circuito que va a diseñarse.

- Se dibuja un diagrama de bloques que represente al circuito. Este diagrama de bloques debe tener identificadas con claridad las entradas y salidas.
- Se establece la relación que existe entre las salidas y entradas (la función de transferencia), relación que está dada por las especificaciones de funcionamiento que debe cumplir el circuito que va a diseñarse.
- Se elabora un algoritmo mediante símbolos especiales (el diagrama de estados) que describe gráficamente el funcionamiento del circuito.
- Se implementa el circuito.

La figura 1.6 muestra la representación de un estado con las partes que lo conforman, su símbolo es un ovalo o puede ser un círculo.

A nivel de circuito, un estado se identifica de la siguiente manera:

- Un estado, medido en tiempo, queda determinado por el tiempo que dura un ciclo o período del reloj.
- Un estado se identifica también por su código. El código de un estado son los bits que se encuentran almacenados temporalmente en las salidas Q de los elementos de memoria, los *flip-flops*.

Desde el punto de vista de un diagrama de estados, a un estado se identifica por:

- Su forma
- Su nombre
- Su código

La forma es un óvalo o un círculo. Su nombre puede ser cualquier nombre, normalmente son letras y/o números simples (a, b, e1, etc.). Su código son las combinaciones de bits que se asigna a cada estado.

La figura 1.6 a) muestra un estado y contiene: una forma (óvalo), un nombre y un código. La figura 1.6 b) muestra un ejemplo de un estado con su forma, su nombre y su código. En la figura 1.6 b), el nombre del estado es a y el código es 1010.

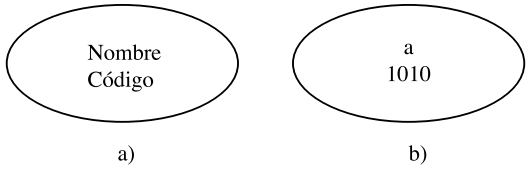


Figura 1.6 Representación de un estado

El número de bits del código de un estado depende del número de estados diferentes que tenga un diagrama de estados. Así, si hay dos estados, se requiere un bit para identificar a cada uno; un estado tendría el código 0 y el otro el código 1. Si existen cuatro estados, se necesitan dos bits, y los códigos de los estados podrían ser: 00, 01, 10 y 11. Para representar seis estados se requieren 3 bits y los códigos podrían ser: 000, 001, 010, 011, 100 y 101 y sobrarían dos códigos. En general, el número de estados diferentes que se pueden generar con n bits es: 2n, donde n es el número de bits. Si hay 12 estados, se requieren al menos cuatro bits, = 16, de las 16 combinaciones se utilizarían 12 y qudarían cuatro libres.

La figura 1.7 muestra la representación temporal de un estado. Cada estado dura un ciclo de reloj y, en cada estado, el circuito debe realizar alguna acción.

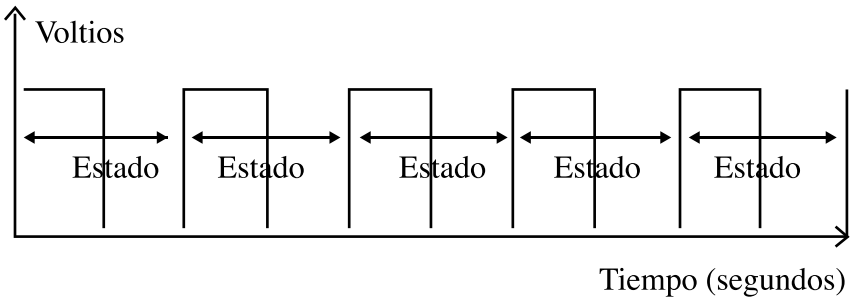


Figura 1.7. Representación temporal de un estado

La figura 1.8 muestra un ejemplo de un diagrama de estados para una máquina Mealy. En una máquina Mealy, las salidas al mundo exterior dependen del estado presente, así como, de las entradas del mundo exterior.

Los segmentos de rectas dirigidas indican el cambio de un estado a otro o incluso el retorno al mismo estado. Las entradas del mundo exterior definen cual será el estado siguiente. En cada estado, también se puede generar una salida o un grupo de salidas.

En la figura 1.8, si un observador se ubica en el estado a, este es entonces el estado presente. Del estado presente a y dependiendo de los valores de las entradas externas, el estado siguiente podría ser el estado b o el estado c.

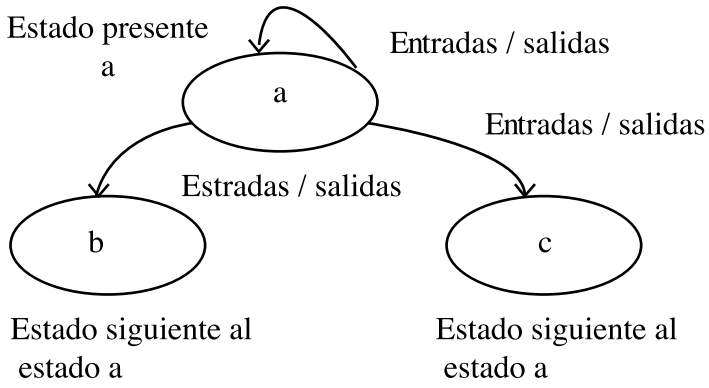


Figura 1.8. Un diagrama de estados con sus partes

En cada estado, se puede generar o encender salidas.

Ejemplo 1.1

Diseñe un circuito secuencial sincrónico, tipo máquina Mealy, que genere los siguientes códigos en forma indefinida: código 1: 00, 01, 10,11, y código 2: 11, 10, 01, 00. La elección del código se realiza mediante la señal x. Si x está en nivel alto, el código que se debe generar es el primero. Si x está en nivel bajo, el código que se debe generar es el segundo. Por la señal de entrada X se debe consultar solo en el estado de inicio.

Para diseñar este circuito se aplican los pasos de diseño indicados en las líneas anteriores

Paso 1. Se analiza las especificaciones que debe cumplir el circuito que se va diseñar.

Según las especificaciones dadas, se requiere diseñar un circuito que genere dos códigos diferentes de dos bits cada uno. La elección del código se realiza mediante la señal de entrada X, de esta descripción se concluye que el circuito debe tener una entrada a la que se va a llamar X. El reloj del circuito es otra entrada y normalmente se le denomina CLK. Las salidas son dos, y se les denomina A y B. En A y B van a estar los códigos solicitados.

Paso 2. El diagrama de bloque que representa al circuito se muestra en la figura 1.9, tiene una entrada X a parte del reloj y dos salidas A y B.

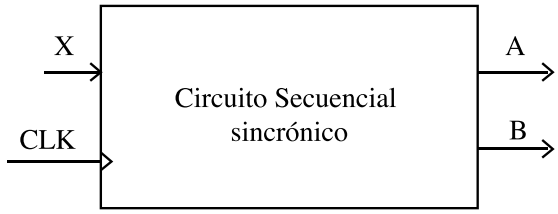


Figura 1.9. Diagrama de bloque

Paso 3. La relación que existe entre las salidas y entradas es la siguiente: cuando la entrada X es verdadera, la salidas A y B deben generar el código: 00, 01, 10, 11 y si la entrada X es falsa el código en A y B es: 11, 10, 01, 00.

Paso 4. Se construye el algoritmo de funcionamiento del circuito mediante un diagrama de estados.

El diagrama de estados se construye partiendo de un estado cualesquiera, por ejemplo el estado "a", que se muestra en la figura 1.10.

De acuerdo a las condiciones del ejercicio solo en este estado se pregunta por el valor de la entrada X, si X=1 se genera el código 00, 01, 10,11 en A y B, de lo contrario se genera el código 11, 10, 01, 00, en A y B, como se ve en la figura 1.10.

Note en la figura 1.10 que se utiliza el símbolo "Φ "para indicar la condición no importa, es decir, puede ser un "1" lógico o un "0" lógico, de acuerdo con las condiciones del diseño solo se debe preguntar por la entrada x en el primer estado; de ahí que, en el resto de los estados, la entrada "X" es una condición no importa, X= Φ simbólicamente.

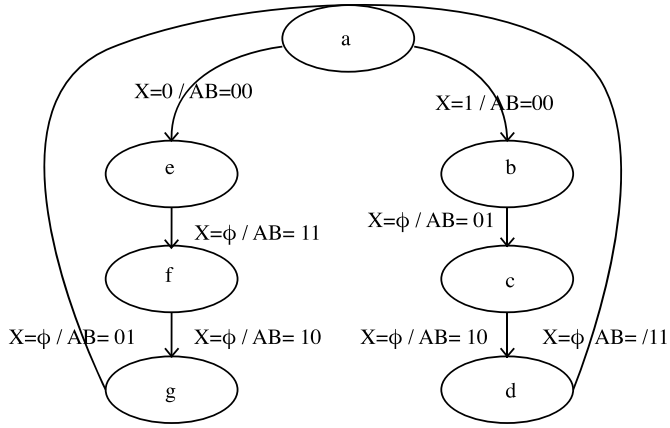


Figura 1.10. Diagrama de estados del ejemplo 1.1

Ejemplo 1.2

Diseñe un contador binario de tres bits. Realice solo el diagrama de estados.

Paso 1. Se analizan las especificaciones que debe cumplir el circuito que se va a diseñar.

Se pide diseñar un contador binario de tres bits. Con tres bits, el número de combinaciones posibles es siete. El contador debe contar del cero al siete. No hay ninguna señal de entrada aparte del reloj; por lo tanto, la cuenta debe ser solo ascendente y cambiar con cada flanco de subida del reloj.

Paso 2. El diagrama de bloques. El contador es de y tres bits y se necesitan 3 salidas una por cada bit (X, Y, Z); no hay ninguna entrada (excepto el CLK). Se debe definir con claridad qué tipo de máquina se va a diseñar. Se pide una tipo Mealy o Moore.

La figura 1.11 a) muestra el diagrama de bloques general del contador. En este caso como no hay señales de entrada al bloque de la lógica de las salidas condicionales, la máquina tiene que ser una máquina tipo Moore, en la figura 1.11 b) se muestra el diagrama general de una máquina Moore y en la figura 1.11 c) el diagrama de bloques del contador como una máquina Moore. Las salidas del contador deben ser encendidas en cada estado. Como el código generado por el contador no se repite, se puede escribir las salidas del contador como el código de cada estado, simplificando de esta manera el bloque de salidas condicionales.

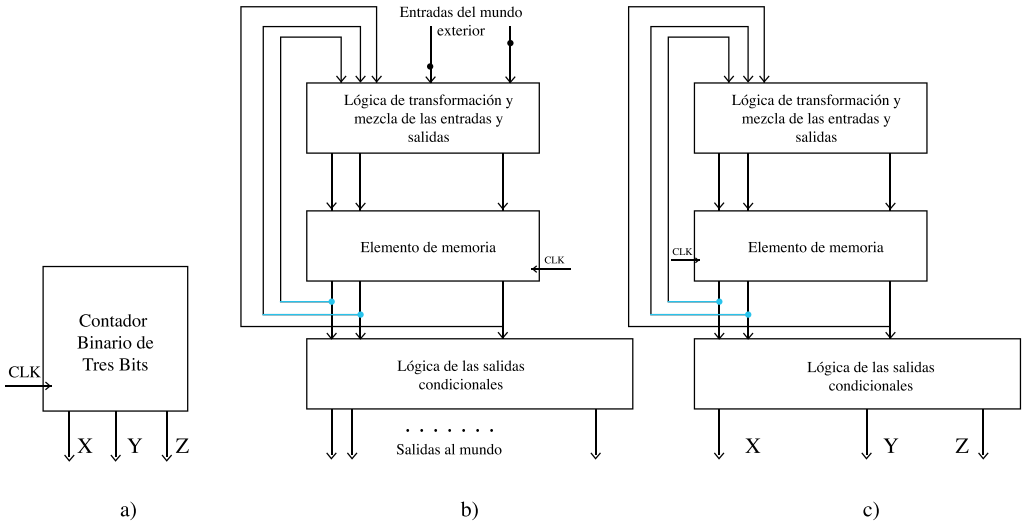


Figura 1.11. Diagrama de bloques a) del contador, b) de una máquina Moore, c) el contador como una máquina Moore.

Paso 3. Del diagrama de estados, figura 1.11 c), se puede concluir que las salidas X, Y y Z del contador dependen solo del estado de la máquina. Además, si el código de las salidas es igual al código de los estados, entonces, se puede eliminar el bloque de lógica de las salidas condicionales. La máquina, por consiguiente, es Moore sin decodificador.

El diagrama de bloques del contador, muestra en la figura 1.12. En esta se puede ver que las salidas del contador son las mismas salidas Q de los *flip-flops*.

El diagrama de estados se inicia en el estado "a" como muestra la figura 1.13. Como no hay entradas, desde el mundo exterior y el código que debe generarse no se repite, se puede asignar el mismo código de cada estado, Qx, Qy, Qz, a las salidas del mundo exterior X, Y, Z. Son necesarios tres *flip-flops* para tener ocho estados diferentes (23=8), como muestra la figura 1.13.

Se asume que el contador es sensible al flanco de subida del reloj. Con el primer flanco del reloj, nace el estado "a". Las salidas del contador son: XYZ=000= QxQyQz. Con el siguiente flanco, del estado "a", se pasa al estado "e" y XYZ=001= QxQyQz. Con el siguiente flanco, del estado "e", se pasa al estado "f", donde XYZ=010= QxQyQz. Con el siguiente flanco, del estado "f", se pasa al estado "g" donde XYZ=011= QxQyQz. Con el siguiente flanco, del estado "g", se pasa al estado "h" donde XYZ=100= QxQyQz. Con el siguiente flanco, del estado "h", se

pasa al estado “d” donde XYZ=101= QxQyQz. Con el siguiente flanco, del estado “d” se pasa al “c” donde XYZ=110= QxQyQz. Con el siguiente flanco, del estado “c” se pasa al “b” donde XYZ=111= QxQyQz. Con el siguiente flanco, del estado “b” se pasa al estado “a” donde XYZ=000= QxQyQz y, de aquí, el ciclo se vuelve a repetir en forma indefinida.

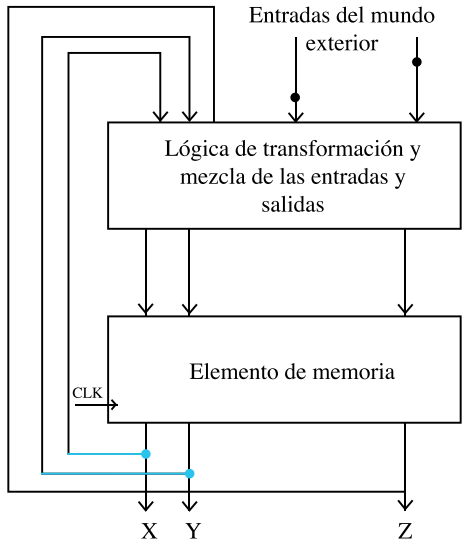


Figura 1.12. Diagrama de bloques del contador

Paso 4. Implementación. Para implementar el circuito, se debe definir el tipo de *flip-flops* que se va a utilizar como elemento de memoria. Para este ejemplo, se elige *flip-flops* D. El diagrama de la figura 1.12 debe incluir tres *flip-flops* tipo D; por lo tanto, debe ser modificado. La figura 1.14 muestra el diagrama de bloques modificado para tres *flip-flops*.

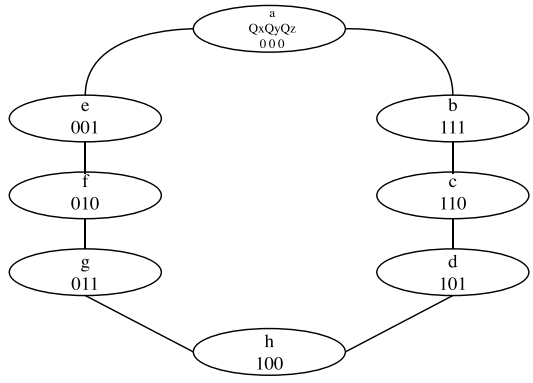


Figura 1.13. Diagrama de estados del contador de tres bits

La figura 1.14 muestra que el único bloque que hay que diseñar es el de la lógica de transformación y mezcla de las entradas y salidas. Como es un bloque combinacional, se utiliza una tabla de verdad. La tabla debe tener las entradas: Qx, Qy, Qz, que van del lado izquierdo de la tabla, las salidas del bloque son: Dx, Dy, Dz.

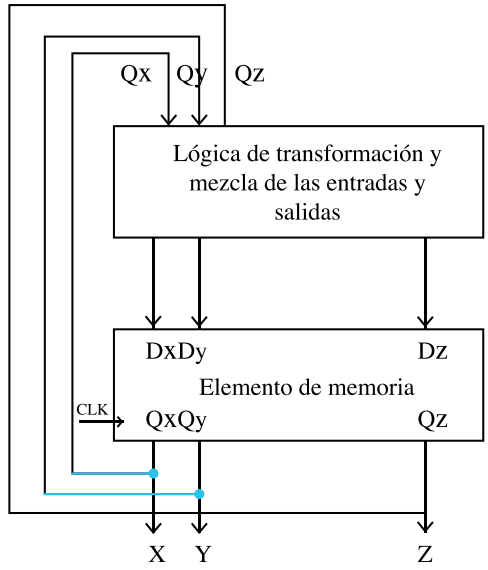


Figura 1.14. Diagrama de bloques del contador con *flip-flops* D

Con estas consideraciones, se construye la tabla de verdad 1.1. La tabla se llena primero con todas las combinaciones posibles para el estado presente; luego, el estado siguiente se llena a partir del diagrama de estados, de la siguiente manera. Cuando el circuito está en el estado presente "a", QxQyQz=000 el estado siguiente debe ser el "e", QxQyQz=001. Con el siguiente flanco de subida del reloj, del estado presente "g", se pasa al estado siguiente "h", donde QxQyQz=100. Con el siguiente flanco de subida del reloj, del estado presente "h", se pasa al estado siguiente "d", donde QxQyQz=010. Con el siguiente flanco de subida del reloj, del estado presente "d", se pasa al estado siguiente "c", donde QxQyQz=010.

Estado presente			Estado siguiente			SALIDAS		
Q_x	Q_y	Q_z	Q_{x+1}	Q_{y+1}	Q_{z+1}	D_x	D_x	D_Z
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0

Tabla 1.1. Código del estado presente y estado siguiente

Como la entrada D de un *flip-flops* es igual al estado siguiente, entones: Dx= Qx+1, Dy=Qy+1 y y Dz=Qz+1.

Las salidas D son: Dx= Qx /Qz + Qx/Qy+ /QxQy/Qz. Dy=Qx /Qz+ /Qy/Qz. y, Dz=/Qz.

Ejemplo 1.3

Se requiere diseñar una máquina secuencial que reciba una secuencia infinita de bits, un bit a la vez con cada flanco de subida del reloj. Cada tres bits de la secuencia forman un número. El dispositivo debe devolver el complemento a dos bits de cada número de tres. Elabore el diagrama de estados. El dispositivo debe enviar por una salida bit a bit y, con cada flanco, el complemento, a dos bits del número.

La figura 1.15 muestra el diagrama de bloques del circuito que se va a diseñar. Los bits que deben complementarse, ingresan uno a uno con cada ciclo del reloj y, en la salida, está el valor complementado; x es la entrada y z es la salida. El algoritmo mediante el cual se va a obtener el complemento a dos de un número es el siguiente: se examina el número desde el bit menos significativo, se busca el primer uno. Este primer uno se deja como uno y el resto de los bits se complementan. Así por ejemplo, si el número es 011, su complemento a dos es 101.

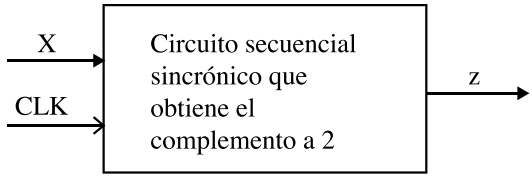


Figura 1.15 Diagrama de bloques para el ejemplo 1.3.

La figura 1.16 muestra el diagrama de estados. Como se puede ver, se parte de un estado llamado "a". En este estado, se consulta el valor de la entrada de datos; si el bit es 0, se sigue un camino y, si es 1, se elige otro.

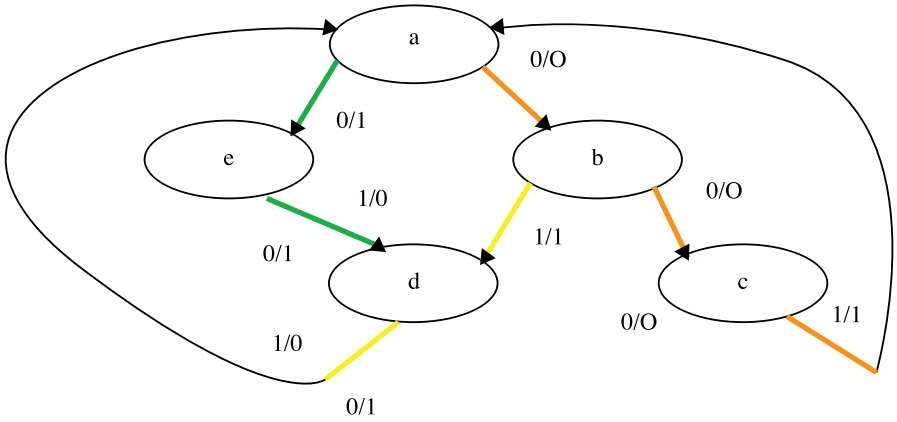


Figura 1.16. Diagrama de estados del complemento a dos

Para elaborar este diagrama se sigue la siguiente estrategia. Se supone primero que los tres bits que se reciben son cero, por lo tanto el "diagrama de estados" pasa del estado a al estado b, del estado b al estado c y finalmente del estado c retorna al estado a. Este camino está indicado en color rojo en el "diagrama de estados". Todas las salidas son cero porque no se ha recibido ningún uno.

Luego el análisis se continua desde el estado C (el camino en color amarrillo), si la entrada es cero ya está considerada, pero, falta el camino que se debe seguir si la entrada es uno, si la entrada es uno, este sería el primer uno que se recibe por lo tanto la salida en ese estado es uno.

El análisis se sigue con el estado b. Falta considerar que sucede si la entrada vale uno. De ser así, este sería el primer uno que recibe el circuito en el estado b; por lo tanto, la salida vale uno, y se va a otro estado que se le llama d.

Al llegar al estado d, ya se recibió el primer uno; por lo tanto, cualquier bit que se reciba debe enviarse complementado, y eso es lo que se realiza en el estado d.

A continuación, se ve que falta considerar qué pasa si, en el estado a, se recibe un uno. Este uno sería el primer uno que se recibe y, por lo tanto, se envía a la salida como uno y el siguiente estado es el e. En este estado, cualquier bit que se reciba debe ser enviado a la salida complementado; de ahí, el siguiente estado es el d, ya que, en este estado, cualquier bit que se reciba es enviado complementado.

Por otro lado, del estado e, podría irse a otro estado, por ejemplo, al h Y, de este estado h, se debería retornar al estado a y cualquier bit que se reciba en este estado se enviaría a la salida complementado. Como se puede concluir, este estado h seria idéntico al estado d. De ahí que es mejor no crear otro estado sino dirigirse al estado d.

Ejemplo 1.4

Elabore el "diagrama de estados" de un circuito secuencial sincrónico que detecte tres unos seguidos, sin traslape, en una secuencia de bits infinita.

El diagrama de bloques se indica en la figura 1.16. Tiene una entrada denominada X aparte del reloj y una salida llamada Y.

Para claridad en la figura 1.16, la relación entre la entrada X y salida Y se expresa como F y es igual a X/Y.

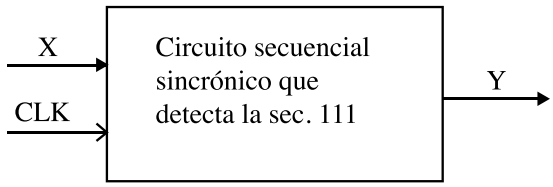


Figura 1.16. Diagrama de bloques del detector de secuencia 111

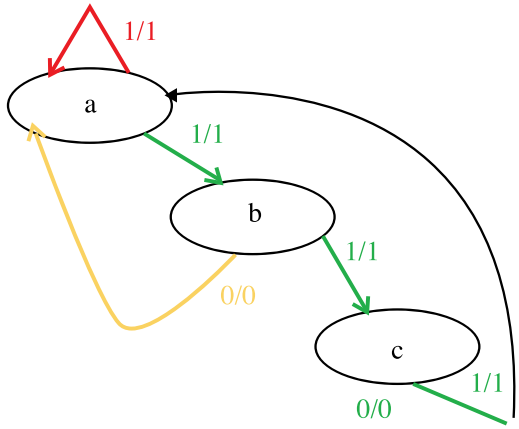


Figura 1.17. Diagrama de estados para detectar la secuencia 111

Se supone que los bits van a ingresar por la entrada X al circuito, en serie y con cada flanco del reloj. El circuito tiene una salida Y que debe encenderse cuando la secuencia 111 ha sido detectada.

Como siempre, se inicia en un estado, para este caso el estado a. Una buena estrategia para detectar un código es asumir que se recibe el código que se está buscando y dibujar el camino o trayectoria directa.

En la figura 1.17, el camino está en color verde. Luego, se va analizando cada uno de los estados desde abajo hacia arriba.

Si, en el estado c, se recibe un cero, se pierde la secuencia. La salida debe ser cero y regresa al estado a.

Si, en el estado b, se recibe un cero, la secuencia se pierde y se retorna al estado a. Esta trayectoria está en la figura 1.17 en color amarrillo.

Si, en el estado a, se recibe un cero, la secuencia se pierde y se debe quedar en el mismo estado hasta recibir un uno. La trayectoria en la figura 1.17 está en color rojo.

Ejemplo 1.5

Diseñe un circuito secuencial sincrónico que compare dos números de tres bits. Los números ingresan bit a bit y con cada flanco del reloj. El circuito debe tener tres salidas que indiquen cuando el número es igual, mayor o menor.

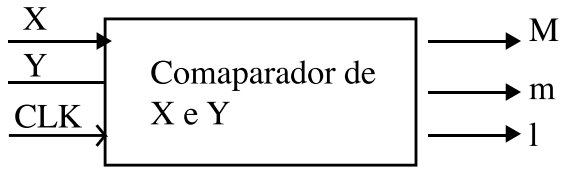


Figura 1.18. Diagrama de bloques del comparador

El circuito que se va a diseñar debe tener dos entradas, aparte del reloj, que se van a denominar X e Y y, tres salidas, una denominada M, que se encenderá cuando el número X sea mayor que el número Y; otra denominada m, que se encenderá cuando el número X sea menor que el Y; y una última denominada I, que se encenderá cuando los números X e Y sean iguales.

La relación entre las entradas y las salidas se denomina F=XY/MmI, el diagrama de bloques del circuito se muestra en la figura 1.18. Esto significa que el primer bit en diagrama de estados representa a X, el segundo a Y y M a X>Y,m a X<Y e I a X=Y. En la figura 1.19, se muestra el diagrama de estados del comparador.

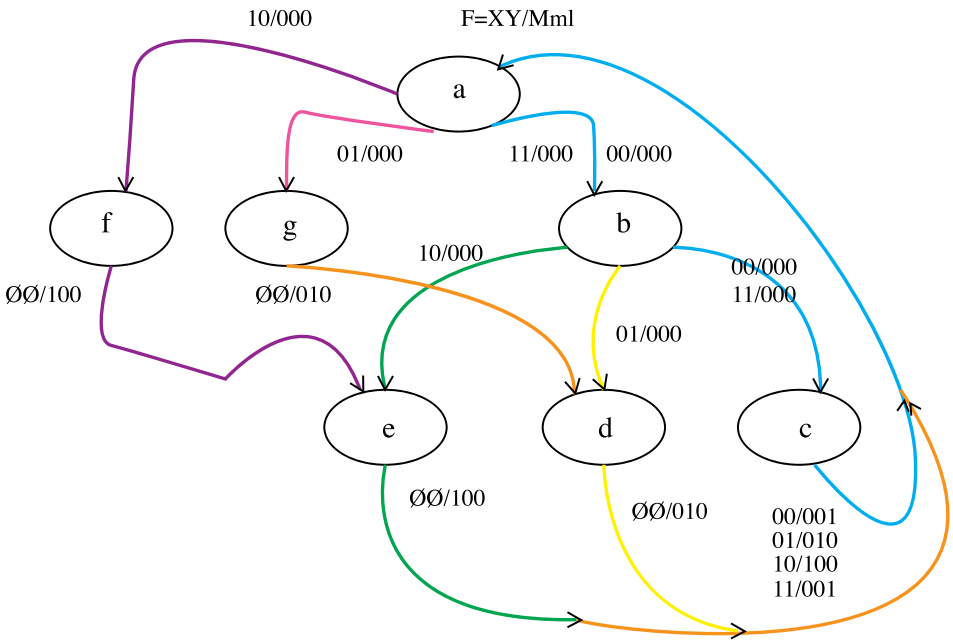


Figura 1.19 Diagrama de estados del comparador

El diagrama de estados de la figura 1.19 se inicia en el estado a. Se asume al inicio que los números X e Y que se reciben son todos ceros, trayectoria de color azul, en el estado a XY=00 y MnI=000. Del estado a, se pasa al estado b. En este estado XY=00 y MnI=000. Del estado b, el siguiente estado es el c y allí XY=00 y MnI=001.

El análisis continúa desde el estado c. Allí se deben analizar todas las combinaciones posibles de las entradas. Se ha analizado solo para XY=00 y faltan XY=01, XY=10 y XY=11.

Si XY=01, entonces X<Y y la salida que se enciende es m que representa a X<Y. Esto se escribe como: XY=01/MmI=010.

Si XY=10, entonces X>Y y la salida que se enciende es M que representa a X>Y. Esto se escribe como: XY=01/MmI=100.

Si XY=11, entonces X=Y y la salida que se enciende es I que representa a X=Y, Esto se escribe como: XY=11/MmI=100.

De esta manera, el estado c queda completamente cubierto. Es decir, se han considerado todas las trayectorias posibles desde el estado c, debido a todas las combinaciones de las entradas X e Y.

En el estado b, solo se ha considerado la trayectoria para XY=00. Falta considerar el resto de las combinaciones: XY=01, XY=10 y XY=11.

Si XY=01 en el estado b, entonces X<Y, ya que, para llegar del estado a al estado b, se cumplió que XY=00; es decir, hasta ahí los números son iguales. Por lo tanto, si XY=01, obligatoriamente X<Y y se dirige del estado b al estado siguiente que es el d, donde, independiente de los valores de X e Y, se cumple que X<Y. En el estado d, se enciende la salida m que representa a X<Y, y se escribe como: XY=ØØ/MmI=010. Debido a que XY=ØØ en el estado d, se han considerado todas las trayectorias posibles para todas las combinaciones de las entradas X e Y y el estado queda completamente cubierto.

En el estado b, para XY=10, X>Y ya que, para llegar del estado a al estado b, se cumplió que XY=00; es decir, hasta ahí los números son iguales. Por lo tanto, si XY=10 obligatoriamente X>Y, y se dirige del estado b al estado siguiente que es el e, donde, independiente de los valores de X e Y, se cumple que X>Y. En el estado e, se enciende la salida M que representa a X>Y, y se escribe como: XY=ØØ/MmI=100. Debido a que XY=ØØ en el estado e se han considerado todas las trayectorias posibles para todas las combinaciones de las entradas X e Y y el estado queda completamente cubierto.

En el estado b, para la combinación de las entradas XY=11, hasta aquí X=Y, ya que para llegar del estado a al estado b, se cumplió que XY=00; es decir, hasta aquí los números son iguales. Por lo tanto, el siguiente bit, de la combinación XY definirán si X<Y o X>Y o si X=Y, pero, precisamente esas combinaciones son realizadas en el estado c, por lo que, del estado b, el siguiente estado es el c.

De esta manera, el estado b queda complemente cubierto y falta analizar el estado a.

En el estado a solo se ha considerado la trayectoria para XY=00, faltan las otras tres posibilidades XY=01, XY=10 y XY=11.

En el estado a, cuando XY=11, si X<Y o X>Y o X=Y dependerá de los dos siguientes bits; pero los estados b y c realizan precisamente ese análisis. Por lo tanto, del estado a, el estado siguiente es b para XY=11.

En el estado a, cuando XY=10, definitivamente X>Y (en el estado a está el bit más significativo) y, por lo tanto, el *reset* o de los valores de XY no importa. De ahí que, del estado a, el siguiente estado es el f y, del estado f, el siguiente estado es el e, porque, en ese estado, se enciende la salida M que significa X>Y.

En el estado a cuando XY=01, definitivamente X<Y (en el estado a esta el bit más significativo) y, por lo tanto, el *reset* o de los valores de XY no importa. De ahí que, del estado a, el siguiente estado es el g y, del estado g, el siguiente estado es el d, porque en ese estado se enciende la salida m que significa X<Y.

1.3.3 Implementación de diagramas de estados

Para implementar un diagrama de estados se debe definir a qué tipo de máquina representa y esto está íntimamente ligado con la relación que hay o no entre las entradas y salidas.

Hay que recordar que, si las salidas dependen tanto del estado de la máquina así como también de las entradas la máquina es una maquina Mealy; si, por el contrario, las salidas no dependen de las entradas sino solo del estado de la máquina, la máquina es una máquina Moore.

En todo caso, como se indicó en el capítulo 1 existen varios tipos de máquinas y son:

- Clase A o máquina Mealy
- Clase B o máquina Moore
- Clase C o máquina Moore sin decodificador de salidas
- Clase D o memoria *look -up*
- Clase C

Los circuitos secuenciales sincrónicos pueden ser diseñados como una máquina Mealy, una Moore o a una Moore sin decodificador de salidas.

Definir a qué máquina representa el diagrama de estados es simple. Sí hay presencia de entradas y salidas y las salidas dependen del estado de la máquina (las salidas Q de los *flip-flops*) así como también de las entradas externas, la máquina es de tipo Mealy. Si las salidas dependen solo del estado (código) de esta, la máquina se denomina Moore.

Las máquinas clase A (Mealy) o B (Moore) tienen tres bloques que son: el decodificador de estado siguiente, el decodificador de salidas y el bloque de memoria.

El bloque de memoria está conformado por algún tipo de *flip-flops*, así, en realidad se debe diseñar solo dos de los tres bloques puesto que los *flip-flops* son circuitos cuya estructura es conocida. Es más, todavía existen en forma de circuitos integrados y lo único que hay que hacer es elegir un tipo de ellos, los dos bloques que deben ser diseñados son de tipo combinacional y la herramienta de diseño es la tabla de verdad.

En realidad, la tarea más difícil en el diseño de circuitos secuenciales es el desarrollo del algoritmo que describa la función o funcionamiento del circuito, algoritmo que es representado mediante símbolos especiales, en este caso los símbolos de un diagrama de estados.

Como se va constatar más a delante, la implementación de un diagrama de estados es en realidad una tarea mecánica, pues solo hay que elaborar la tabla característica para cada bloque, encontrar las expresiones booleanas e implementar.

A continuación se presentan varios ejemplos de diseño e implementación de circuitos secuenciales sincrónicos.

Ejemplo 1.6

Diseñe un circuito secuencial sincrónico que detecte el código: 110 en tres muestras consecutivas. Use *flip-flops* tipo D para implementar el bloque de memoria.

El diagrama de bloques del detector de secuencia se indica en la figura 1.20,. El diagrama de estados está indicado en la figura 1.21. y el tipo de máquina en la figura 1.22.

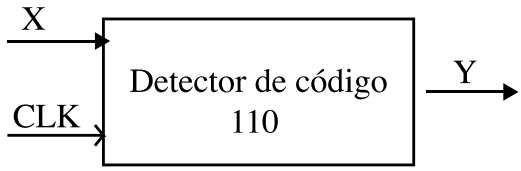


Figura 1.20. Diagrama de bloques del detector de código

En la figura 1.20, se supone que los bits ingresan, por la entrada X, uno a uno y con cada flanco de subida del reloj. El circuito analiza la cadena de bits y debe encender la salida Y cada vez que la secuencia pedida es detectada.

Una buena estrategia para diseñar detectores de código es asumir que el circuito recibe directamente el código buscado. Así, en el estado a, recibe el primer uno; luego, en el estado b, recibe el segundo uno y, en el tercer estado, c, recibe un cero.

Luego se va analizando cada estado. Se inicia en el último estado, en este caso el c. Falta analizar si la entrada recibe un uno. Si este es el caso, la secuencia se perdió, pero, este uno puede ser parte de la secuencia así que hay que conservarlo, y la forma de conservar este uno es regresando al estado c. De esta forma queda cubierto completamente el estado c.

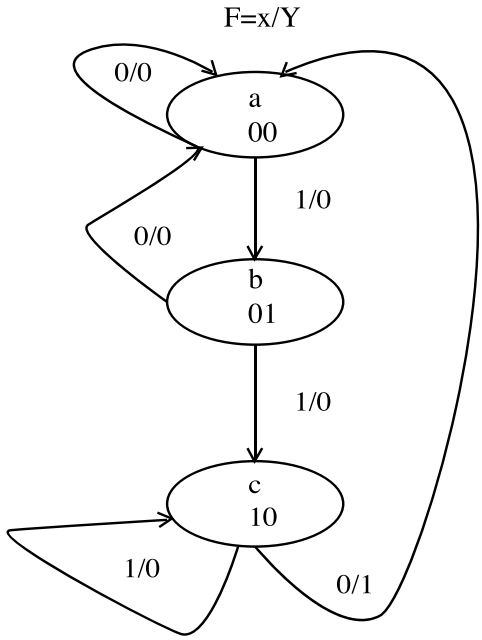


Figura 1.21. Diagrama de estados del detector de código 110

En el estado b, falta analizar qué pasa si la entrada es cero. Con este bit, se pierde la secuencia y el circuito debe retornar al estado a. Esto cubre completamente el estado b.

En el estado a, falta analizar si la entrada recibe un cero. La secuencia se pierde y el circuito debe quedarse en ese estado hasta que reciba un uno que es el primer bit de la secuencia.

En el diagrama de estados de la figura 1.21, se ve claramente que la salida Y depende del valor de la entrada X; por lo tanto, este diagrama representa a una máquina tipo Mealy.

En la figura 1.22, se muestra el diagrama de bloques de la máquina tipo Mealy y el diagrama de bloques general del detector de secuencia.

Como se puede concluir, dos bloques deben ser diseñados: el decodificador de estado siguiente y el decodificador de salidas.

El diagrama de estados tiene tres estados diferentes por lo que se necesitan dos *flip-flops*, al uno se le denomina a y al otro b. Sus entradas son Da y Db y sus salidas, Qa y Qb.

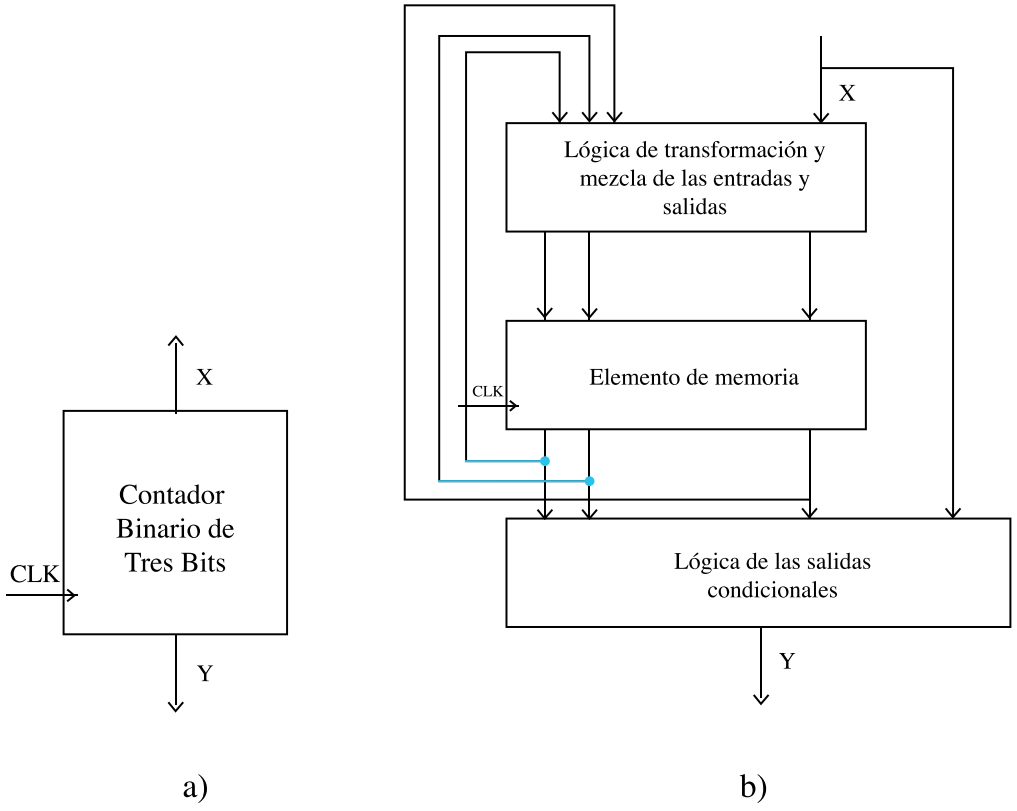


Figura 1.22. Diagrama de bloques del detector a) general, b) máquina Mealy

En la figura 1.22 se ve el diagrama de bloque del detector de secuencia como una máquina tipo Mealy, el decodificador de estado siguiente o bloque de lógica de transformación de las entradas y salidas tiene tres entradas Qa, Qb y X, y dos salidas Da y Db.

El bloque de la lógica de las salidas condicionales tiene tres entradas Qa, Qb, X y la salida Y.

La tabla de verdad para el diseño simultáneo de los dos bloques es la 1.2.

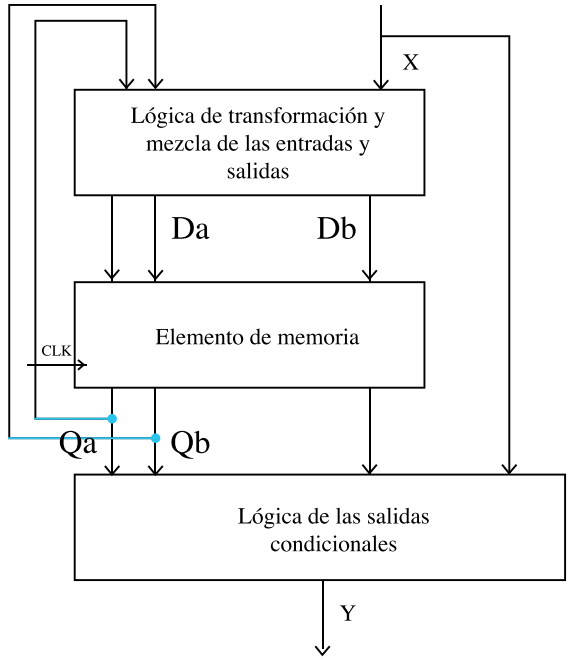


Figura 1. 23. Diagrama de bloques del detector como máquina Mealy.

De la tabla caracteristica 1.2 se encuentran las ecuaciones booleanas para DA, Db e Y.

Da= Qb X +Qa X

Db= /Qa /Qb.X

Y= Qa./X

Qa	Qb	x	Q a+1	Q b+1	Da	Db	y
0	0	0	0	0	0	0	1
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	0
0	1	1	1	0	1	0	0
1	0	0	0	0	0	0	**1**
1	0	1	1	0	1	0	0
1	1	0					
1	1	1					

Tabla 1.2. Tabla para el diseño del detector de secuencia

Ejemplo 1.7

Obtenga la tabla de verdad para diseñar un circuito secuencial sincrónico que divida un número de tres bits para dos. La división debe ser solo entre números que den una división exacta. Haga los diagramas de bloques que se requieran, la tabla de verdad para la implementación de los decodificadores de estado siguiente y de salidas.

Para los números que son divisibles para dos, el circuito debe enviar a su salida el resultado de la división para dos. Para los números que no son divisibles para dos, el circuito debe enviar a su salida el mismo número.

Con tres bits se pueden tener números en el rango de cero a siete. Los únicos números que divididos para dos tienen un residuo de cero son, el seis, el cuatro y el dos.

El diagrama de bloques del circuito que debe diseñarse se muestra en la figura 1.24. El circuito tiene tres entradas X, Y, Z y tres salidas S1, S2, S3.

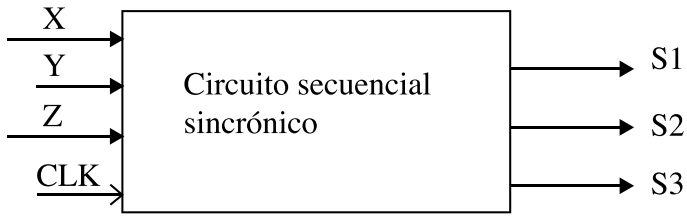


Figura 1. 24. Diagrama de bloque del circuito que divide para dos

El diagrama de estados se muestra en la figura 1.25.

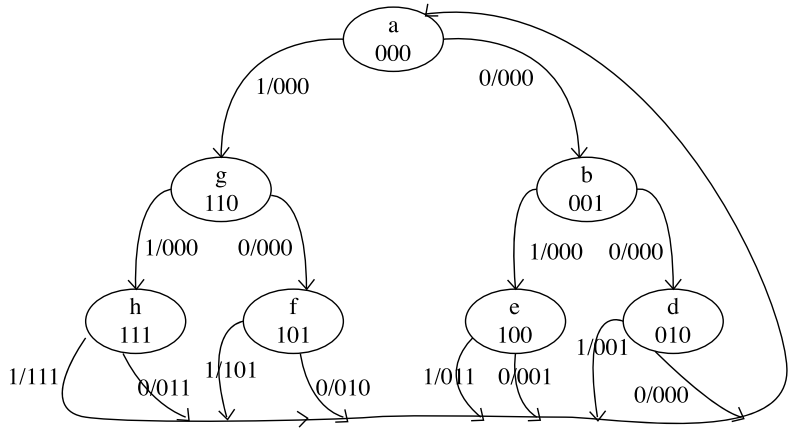


Figura 1.25. Diagrama de estados del circuito que divide para 2

De la figura 1.25, la tabla de verdad para el diseño del decodificador de estado siguiente y de salidas es como se indica en la tabla 1.3.

Qa	Qb	Qc	x	Qa+1	Qb+1	Qc+1	Da	Db	Dc	S1	S2	S3
0	0	0	0	0	0	1	0	0	1	0	0	0
0	0	0	1	1	1	0	1	1	0	0	0	0
0	0	1	0	0	1	0	0	1	0	0	0	0
0	0	1	1	1	0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	1
0	1	1	0	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	0	0	0
0	1	1	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	1	0
1	0	1	1	0	0	0	0	0	0	1	0	1
1	1	0	0	1	0	1	1	0	1	0	0	0
1	1	0	1	1	1	1	1	1	1	0	0	0
1	1	1	0	0	0	0	0	0	0	0	1	1
1	1	1	1	0	0	0	0	0	0	1	1	1

Tabla 1.3 Combinaciones posibles para los decodificadores.

De la tabla 1.3, se encuentran las ecuaciones booleanas y se implementa el circuito. Se pueden utilizar dispositivos como multiplexores y/o decodificadores para implementar esta tabla.

Ejemplo 1.8

Diseñe un circuito secuencial que divida la frecuencia del CLK para 2.

Tomando como referencia el diagrama de bloques de un *flip-flops* tipo D, se puede ver que, si se realimenta la señal /Q hacia la entrada D del *flip-flops*, la salida Q va en forma alternante de 0 a 1 y de 1 a 0 con cada flanco del CLK. El diagrama en el dominio del tiempo se puede apreciar en la figura 1.27.

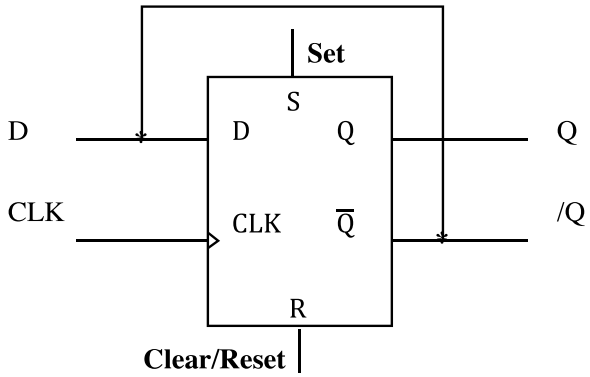


Figura 1.26. Diagrama del F/F D como divisor de la frecuencia del reloj

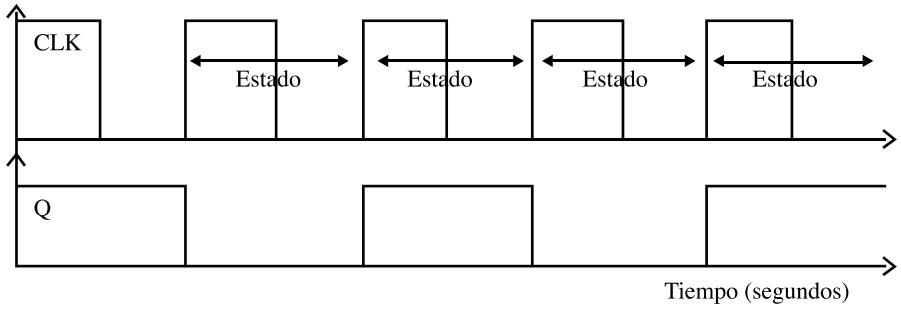


Figura 1.27. La salida Q del *flip-flops* D.

Ejemplo 1.9

Diseñe un circuito secuencial, que divida la frecuencia del CLK para 4

Tomando como referencia el diagrama de bloques de la figura 1.26, se puede ver que, si se realimenta la señal /Q hacia la entrada D, la frecuencia de la salida Q se divide para 2, por tanto, si la salida Q se conecta a la entrada del CLK del otro bloque, la frecuencia de la salida Q2 será $\frac{fo}{2}/2$ es decir $\frac{fo}{4}$. Si se añade otro bloque en cascada, la frecuencia se seguirá dividiendo para 2 figura 1.28.

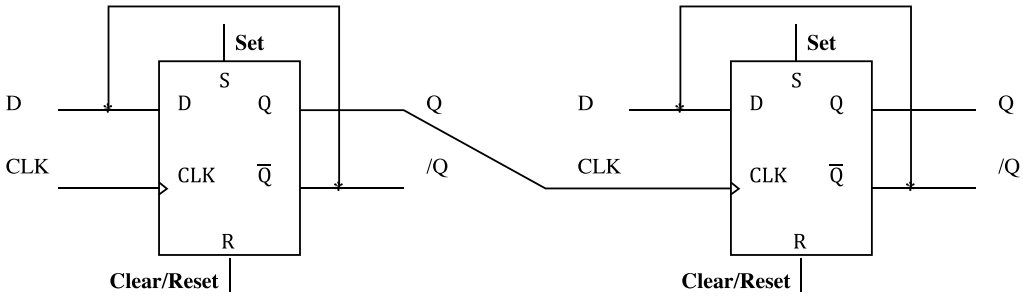


Figura 1.28. Dividiendo la frecuencia del reloj para 2 y para 4.

Ejemplo 1.10

Para el ejercicio anterior encuentre la relación matemática entre la frecuencia y el número de *flip-flops*.

Número de F/F en cascada	1	2	3	4	5	6	7	8	9
Frecuencia de la salida Q de cada Flip-Flop	$\frac{fo}{2}$	$\frac{fo}{4}$	$\frac{fo}{8}$	$\frac{fo}{16}$	$\frac{fo}{32}$	$\frac{fo}{64}$	$\frac{fo}{128}$	$\frac{fo}{256}$	$\frac{fo}{512}$

Tabla 1.4. Número de *flip-flops* en cascada y frecuencia

Cada salida Q de cada *flip-flops* en cascada divide la frecuencia de su reloj por dos. La tabla 1.4 muestra la relación entre la cantidad de *flip-flops* y la frecuencia de la salida Q. La tabla 2.4 se puede escribir como la tabla 1.5.

De la tabla anterior, se deduce que la frecuencia de la salida Q del Nsimo *flip-flops* está dado por:

$$fQN = \frac{fo}{(2)^N}$$

Número de Flip-Flops cascada	1	2	3	4	5	6	7	.... .	N
Frecuencia de la salida Q de cada Flip-Flop	$\frac{fo}{(2)^1}$	$\frac{fo}{(2)^2}$	$\frac{fo}{(2)^3}$	$\frac{fo}{(2)^4}$	$\frac{fo}{(2)^5}$	$\frac{fo}{(2)^6}$	$\frac{fo}{(2)^7}$	... ...	$\frac{fo}{(2)^N}$

Tabla 1.5. Relación entre *flip-flops* en cascada y la frecuencia del reloj

Ejemplo 1.11

¿Se puede decir que el circuito del ejercicio anterior es secuencial sincrónico?

Es un circuito secuencial, pero debido a que el CLK no es común a todos los *flip-flops* se podría decir que, desde el punto de vista del CLK es asincrónico.

Ejemplo 1.12

¿Se puede aplicar la técnica de diagramas de estado para analizar o diseñar circuitos como el del ejercicio anterior?

No, porque la técnica que se está estudiando solo es aplicable cuando los relojes de los *flip-flops* están todos conectados a un solo reloj que es el reloj del sistema.

1.4 Contadores

Un contador es un circuito secuencial sincrónico que pasa por un número de estados predeterminados.

Los contadores son utilizados para:

1. Generar un código en sus salidas con cada ciclo del reloj.
2. Contar y almacenar el número de veces que una señal de entrada se ha hecho verdadera.
3. Generar retardos de tiempo.

Una forma de clasificar a los contadores es tomando en cuenta los siguientes parámetros:

1. El número de bits de salida. Ejemplo, contador de dos bits. Este tiene dos líneas de salida una para cada bit.
2. El número de estados por los que atraviesa. Por ejemplo, si el diagrama de estados del contador tiene seis estados es un contador módulo 6.
3. La secuencia que genera.
4. Sincrónico o asincrónico.
5. Por el modo de operación: modo simple o multimodo.

Los contadores suelen **especificarse** por los siguientes parámetros:

1. Modo de operación
2. Número de bits de salida
3. Número de módulo
4. Tipo de código

Por ejemplo se puede tener un contador de modo simple, tres bits de salida, módulo 6, o seis estados, código binario.

El diagrama general de bloques de un contador de modo simple es el que se indica en la figura 1.29.

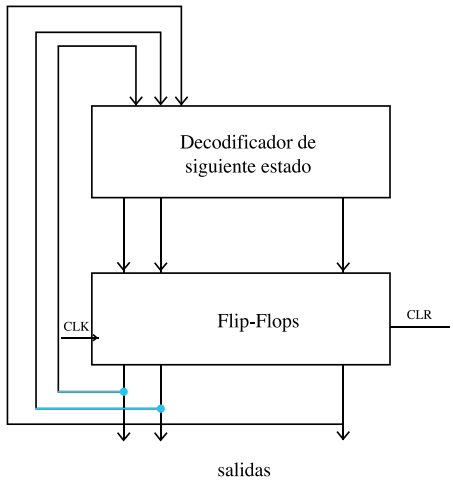


Figura 1.29. Diagrama de un contador de modo simple

Como se puede observar, el diagrama es parecido al de una máquina clase C sin entradas externas en el decodificador de estado siguiente.

En un contador de modo simple, el código y el número de estados por los que debe pasar están plenamente definidos.

Los contadores son las máquinas secuenciales sincrónicas más fáciles de diseñar, porque el número de estados están bien definidos.

Ejemplo 1.13

Diseñe un contador binario que tenga las siguientes características: modo simple, tres bits de salida, código de distancia unitaria y módulo 6.

Con tres bits de salida, el contador puede generar un código de hasta ocho valores diferentes. Como el contador es módulo seis, de las ocho combinaciones posibles solo debe utilizar seis y, para que las salidas tengan distancia unitaria solo un bit de salida debe cambiar.

El código que se va a generar es el siguiente: 000 – 001 – 011 – 010 – 110 – 100.

El diagrama de bloques del contador se muestra en la figura 1.30.

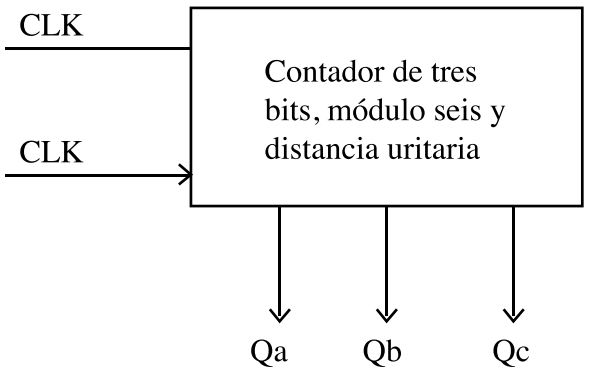


Figura 1.30. Contador modulo seis de tres bits

El diagrama de estados se muestra en la figura 1.31.

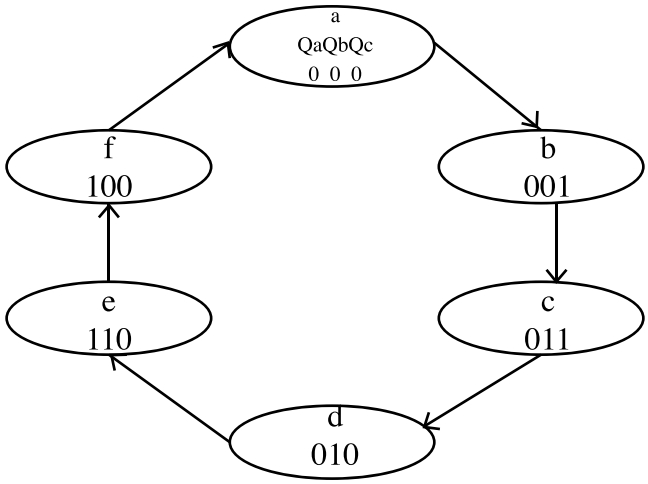


Figura 1.31. Diagrama de estados para el contador de tres bits.

Como es un contador de modo simple, las salidas del contador son las salidas Q de los *flip-flops* del elemento de memoria. El contador tiene una señal de *clear* (CLR) que está directamente conectada a los *flip-flops* y la función es poner a cero las salidas de los *flip-flops´s*, y en este caso también a las salidas del contador, cuando esta señal es verdadera. El contador podría utilizar cualquier tipo de *flip-flops*, el tipo D, el tipo T, el JK. Lo más común es que los contadores se diseñen con *flip-flops* tipo D por la simplicidad de estos dispositivos.

La tabla para el diseño del decodificador de estado siguiente es la 1.6.

Qa	Qb	Qc	Qa+1	Qb+1	Qc+1	Da	Db	Dc
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	1
0	1	0	1	1	0	1	1	0
0	1	1	0	1	0	0	1	0
1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0
1	1	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

Tabla 1.6. Diseño del decodificador de estado siguiente

Las ecuaciones para las entradas de los *flip-flops* son: Da= Qb/Qc; Db= /Qa / Qb; Dc= Qb /Qc. La figura 1.32 muestra la implementación.

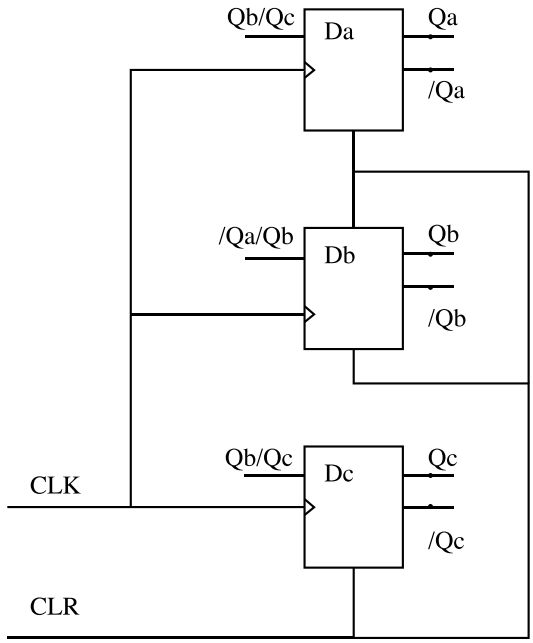


Figura 1.32. Implementación del contador

Ejemplo 1.14

Diseñe un contador Johnson de cuatro bits, con un *clear* sincrónico. Utilice *flip-flops* tipo D en la implementación. El código que generan estos contadores es el siguiente:

0000
0001
0011
0111
1111
1110
1100
1000

La señal de *clear* debe ser sincrónica, es decir, debe estar sincronizada con el reloj del contador. Así, si la señal de *clear* es verdadera y el reloj es verdadero el contador debe poner en cero todas sus salidas; de lo contrario, el contador no se resetea. Síncrono significa que trabaja en concordancia con el reloj. Las señales asíncronas, como su nombre indica, no trabajan en sincronía con el reloj. Cuando una señal asíncrona es verdadera, esta se ejecuta independientemente del estado en el que se encuentre el reloj. El diagrama del contador muestra la figura 1.33.

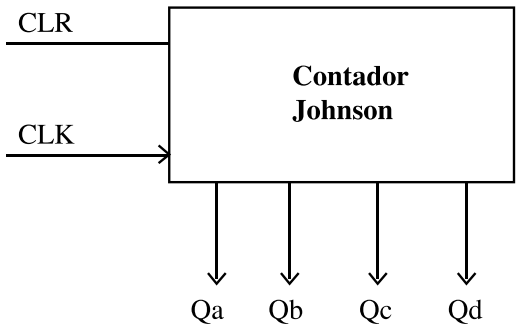


Figura 1. 33. Contador Johnson de cuatro bits

El diagrama de estados para este contador se muestra en la figura 1.34.

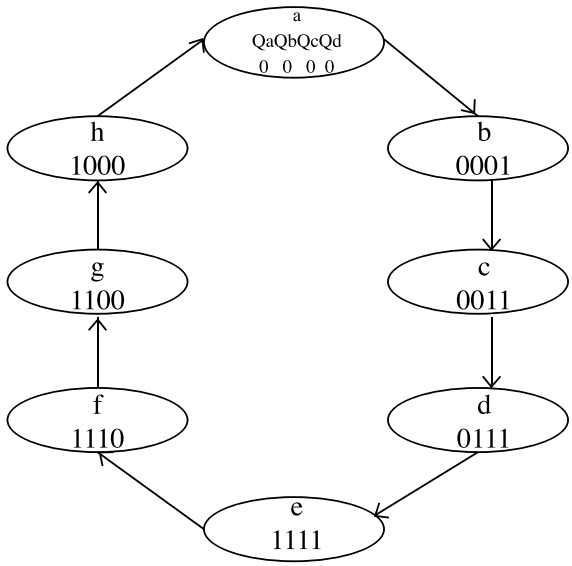


Figura 1.34. Diagrama de estados del contador Johnson

El diagrama de estados se inicia en el estado a. En este estado, debe generar las salidas Qa, Qb, Qc, Qd = 0000 respectivamente. Con el flanco de subida del reloj las salidas deben cambiar a Qa, Qb, Qc, Qd = 0001 respectivamente, y así sucesivamente hasta completar el código, no hay señales de entrada y tampoco un decodificador de salidas. Las salidas del contador son las salidas de los cuatro *flip-flops*. La tabla 1.7 muestra la relación entre las entradas y salidas.

Qa	Qb	Qc	Qd	Qa+1	Qb+1	Qc+1	Qd+1	Da	Db	Dc	Dd
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	1	0	0	1	1
0	0	1	0	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
0	0	1	1	0	1	1	1	0	1	1	1
0	1	0	0	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
0	1	0	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
0	1	1	0	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
0	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
1	0	1	0	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
1	0	1	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
1	1	0	0	1	0	0	0	1	0	0	0
1	1	0	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
1	1	1	0	1	1	0	0	1	1	0	0
1	1	1	1	1	1	1	0	1	1	1	0

Tabla 1.7. Para el diseño del contador Johnson

1.5 Contadores multimodo

Los contadores multimodo, a diferencia de los contadores monomodo, tienen una señal de control que precisamente controla el modo de operación del dispositivo. Por ejemplo, la señal de control puede hacer que el contador detenga su cuenta o invierta el sentido de la cuenta.

Ejemplo 1.15

Diseñe un contador ascendente/descendente, módulo tres, distancia unitaria, con un *clear* sincronizado con el reloj negado. El diagrama de bloques del contador muestra la figura 1.35. Tiene una señal de control A/D que permite que el contador incremente o decremento su cuenta. Si A/D=1, la cuenta debe incrementarse y si A/D=0 la cuenta debe decrementar.

El diagrama de estados se muestra en la figura 1.36. Parte de un estado a en donde genera el código 000. Si A/D=1, el contador va al siguiente estado que es el b y genera el código 001. Si, en el estado a, la entrada A/D es cero, el contador va al estado f y genera el código 111 y así sucesivamente.

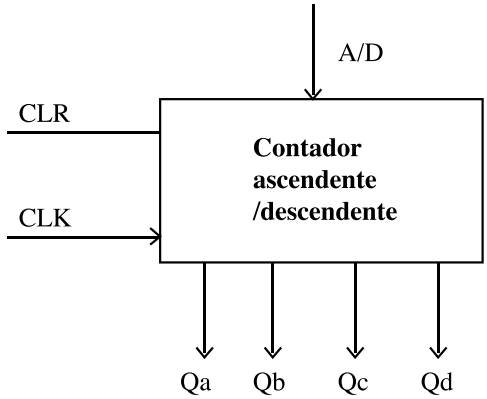


Figura 1.35. Diagrama de bloques del contador ascendente /descendente.

Hay que diseñar el decodificador de estado siguiente. Las salidas del contador son las mismas salidas de los *flip-flops*. La tabla 1.8 muestra el paso de estado a estado de este contador.

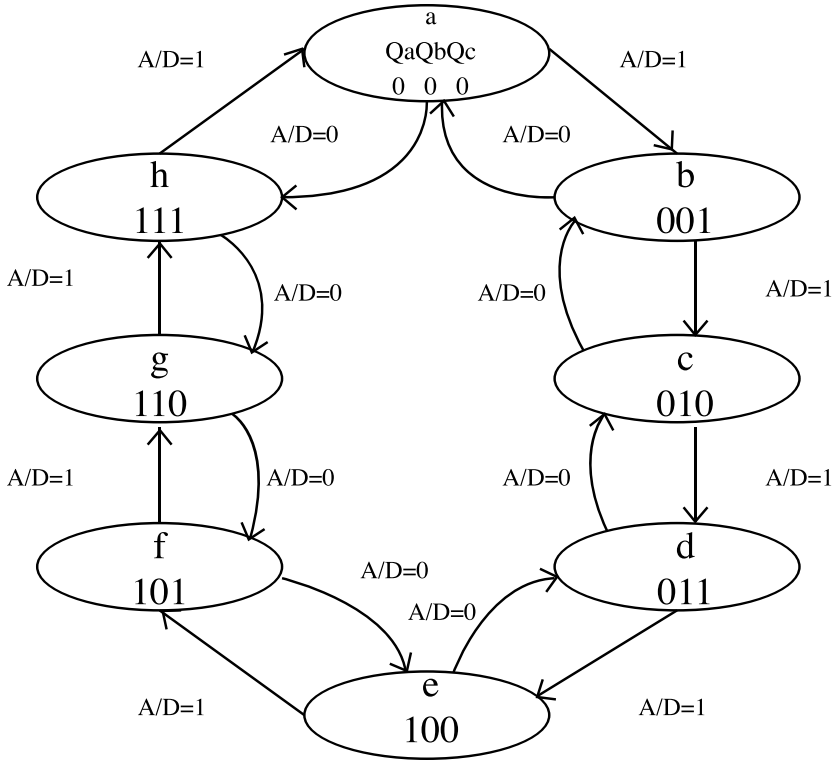


Figura 1.36. Diagrama de estados del contador ascendente/descendente

A/D	Qa	Qb	Qc	Qa+1	Qb+1	Qc+1	Da	Db	Dc
0	0	0	0	1	1	1	1	1	1
0	0	0	1	1	1	0	1	1	0
0	0	1	0	1	0	1	1	0	1
0	0	1	1	1	0	0	1	0	0
0	1	0	0	0	1	1	0	1	1
0	1	0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	0	0	1
0	1	1	1	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	0	1	0
1	0	1	0	0	1	1	0	1	1
1	0	1	1	1	0	0	1	0	0

1	1	0	0	1	0	1	1	0	1
1	1	0	1	1	1	0	1	1	0
1	1	1	0	1	1	1	1	1	1
1	1	1	1	0	0	0	0	0	1

Tabla 1.8. Tabla del contador de subida/bajada

Las ecuaciones booleanas son:

Da= /Qa /(A/D) + Qa/Qc/(A/D)+ /Qa QbQc

Db= /AD/Qa/Qb + /Qb(A/D)Qc + Qb/Qc(A/D)

Dc= /Qc

En la figura 1.37, se muestra la implementación del circuito, no se han dibujado las puertas lógicas, en su lugar, se han escrito las ecuaciones booleanas para cada entrada D de los *flip-flops*.

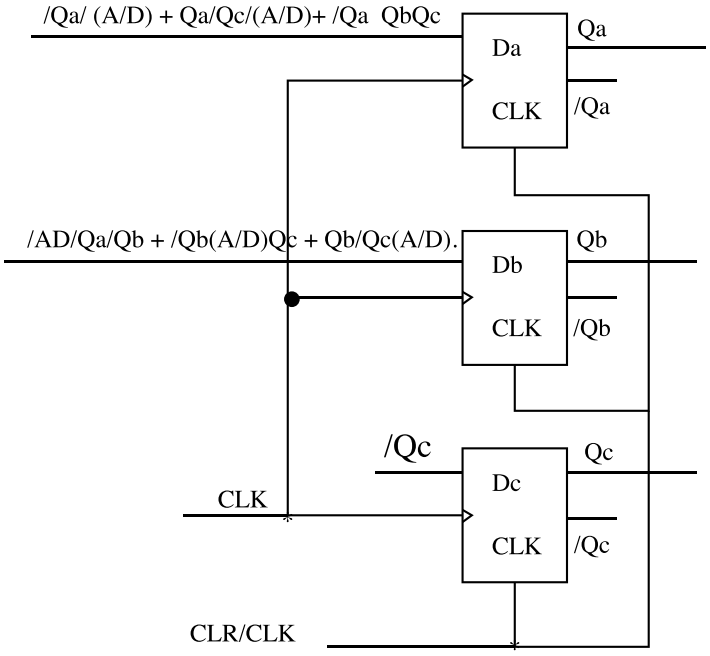


Figura 1.37. Implementación del contador ascendente/descendente

La señal *clear* (CLR) se sincroniza con el reloj realizando la operación lógica AND con el reloj de los *flip-flops*. Nótese que el CLR se activa con el reloj negado.

1.6 Contadores asincrónicos

Las técnicas de diseño de circuitos secuenciales sincrónicos estudiados en este capítulo no son aplicables a circuitos que no tienen un reloj común (asincrónicos); por lo tanto, se debe aplicar otro método para diseñar este tipo de circuitos.

El diseño de contadores asincrónicos es fácil realizarlo con *flip-flops* tipo T, debido a su forma particular de funcionamiento, ya que mientras la entrada T sea igual a 1 y el reloj sea verdadero la salida Q cambia, conmuta continuamente y con cada flanco del reloj, o visto de otra manera, la salida Q tiene la mitad de la frecuencia del reloj, y si este reloj se aplica a una segunda etapa entonces la salida Q de esta segunda etapa tendrá una frecuencia que es la cuarta parte de la frecuencia del reloj, y si a su vez la salida Q de esta segunda etapa se aplica al reloj de la tercera etapa entonces la salida Q de esta tercera etapa tendrá la octava parte de la frecuencia del reloj.

También es factible conectar las salida /Q de cada *flip-flops* al reloj de la siguiente etapa, a continuación se analizan las dos estructuras.

Un contador descendente asincrónico se muestra en la figura 1.38. Como se puede ver la salida Q0 del primer *flip-flops* es el reloj del siguiente *flip-flops*, a su vez la salida Q1 de este último *flip-flops* es el reloj del siguiente *flip-flops*.

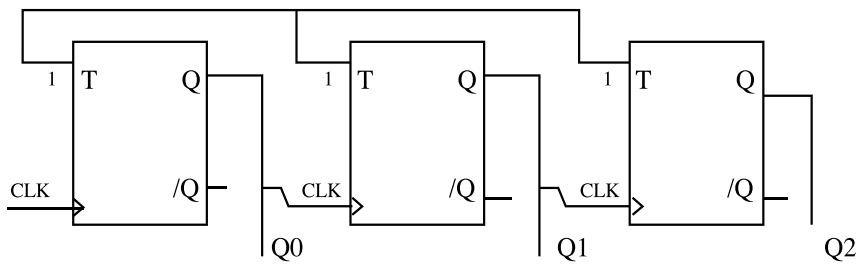


Figura 1.38. Contador descendente de tres bits

El funcionamiento de este contador se analiza etapa por etapa; primero se analiza la etapa a la que está conectada la señal de reloj, en este caso, el *flip-flops* que tiene la salida Q0, el diagrama de tiempo de esta etapa se indica en la figura 1.40 con la señal que está en color rojo.

Esta señal Q0 que está en color rojo se aplica al reloj de la siguiente etapa y se obtiene la señal Q1 de color azul en la figura 1.39. Esta señal de color azul se aplica al reloj de la última etapa y se obtiene la señal de color verde.

Debe notarse que cada *flip-flops* cambia de estado siempre que el reloj estéen su transición o flanco de subida.

De la figura 1.39, se puede ver que este contador formado con *flip-flops* tipo T genera una cuenta que va desde el número siete al número cero y repite esta secuencia en forma indefinida. Como la cuenta es en forma descendente, a este contador se lo llama contador descendente.

El contador que muestra la figura 1.40 es similar al contador de la figura 1.38 con la diferencia de que la salida /Q de cada etapa se conecta al reloj de la siguiente etapa.

Se analiza en la misma forma que se hizo para el contador descendente y el diagrama de tiempo resultante se muestra en la figura 1.41.

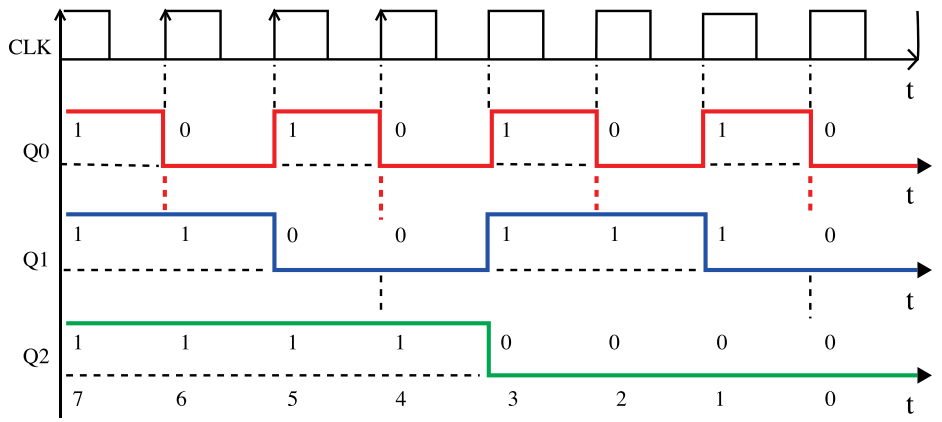


Figura 1.39. Diagrama de tiempo del contador descendente

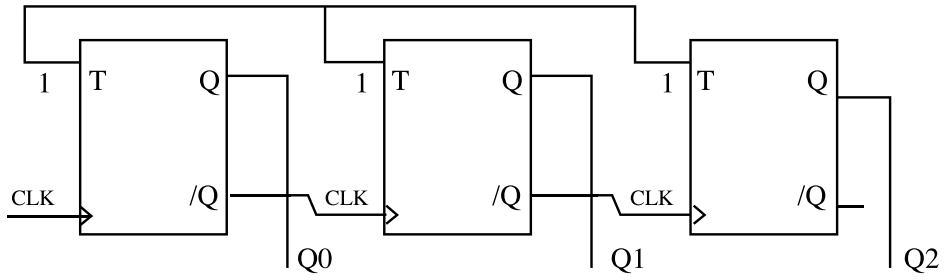


Figura 1.40. Contador ascendente asíncrono de tres bits

Como se puede ver, la cuenta de este contador va desde cero hasta siete. Es importante notar que cada *flip-flops* de este contador se activa con la señal Q negada, esto significa que cuando Q esté en el flanco de bajada, el *flip-flops* se activa (cuenta) como muestra la figura 1.41.

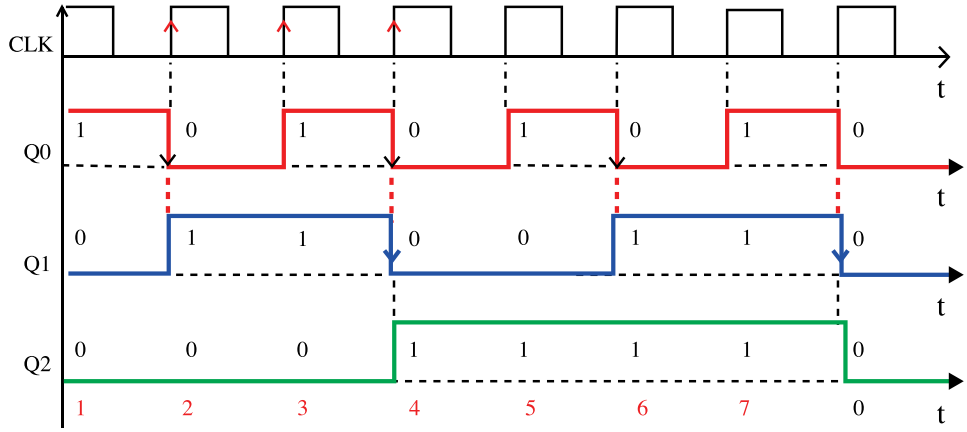


Figura 1.41. Diagrama de tiempo del contador ascendente

Ejemplo 1.16

Elabore el diagrama de estados y la tabla característica tanto para *flip-flops* tipo D y T de un contador ascendente de tres bits. El contador debe contar desde cero (000) hasta siete (111).

El código que debe generar el contador es el siguiente: 000-001-010-011-100-101-110-111. El diagrama de estados para este contador muestra la figura 1.42.

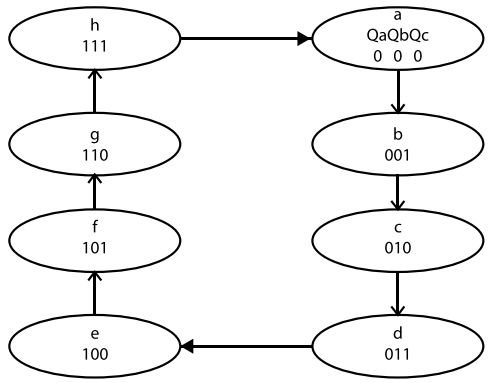


Figura 1.42. Diagrama de estados del contador ascendente de tres bits

La tabla característica 2.9 contiene las salidas para *flip-flops* tipo D y T. De esta tabla se puede implementar el contador tanto con *flip-flops* tipo D así como también con tipo T.

Una vez que un diagrama de estados se ha implementado con algún tipo de *flip-flops*, por ejemplo con el tipo T, se puede implementar ese mismo diagrama de estados con otro tipo de *flip-flops*, por ejemplo con los tipo JK. Para esto, lo único que hay que hacer es convertir los *flip-flops* tipo T en flip-flops JK.

El proceso de conversión es idéntico a como se realizó la conversión de *latch* asincrónicos; lo único que hay que hacer es cambiar los *latch* asincrónicos por *flip-flops*, pero, vale insistir, la técnica del proceso de conversión es exactamente la misma.

Qa	Qb	Qc	Qa+1	Qb+1	Qc+1	Da	Db	Dc	Ta	Tb	Tc
0	0	0	0	0	1	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0	0	1	1
0	1	0	0	1	1	0	1	1	0	0	1
0	1	1	1	0	0	1	0	0	1	1	1
1	0	0	1	0	1	1	0	1	0	0	1
1	0	1	1	1	0	1	1	0	0	1	1
1	1	0	1	1	1	1	1	1	0	0	1
1	1	1	0	0	0	0	0	0	1	1	1

Tabla 1.9. Entradas de los *flip-flops* D y T

Ejemplo 1.17

Explique cómo analizaría un circuito secuencial sincrónico a partir de su diagrama de circuito.

Con el diagrama de circuitos, el proceso es exactamente contrario al proceso de diseño. En el proceso de diseño, se parte de las especificaciones que debe cumplir el circuito, se realiza un diagrama de bloques, se realiza el diagrama de estados; del diagrama de estados, se elabora la tabla característica; de la tabla característica, se encuentran las ecuaciones booleanas y se implementan esta ecuaciones con compuertas lógicas; la tabla característica también se puede implementar directamente con multiplexores o decodificadores.

En el proceso de análisis se parte del circuito. Por lo tanto, lo primero que hay que hacer es identificar las entradas de los *flip-flops* que son las salidas del decodificador de estado siguiente, y las salidas del circuito que son las salidas del decodificador de salidas.

Luego se escriben las ecuaciones booleanas de cada entrada de cada *flip-flops*, así como las ecuaciones booleanas de cada salida; de estas ecuaciones booleanas, se elabora la tabla característica; de la tabla característica, se elabora el diagrama de estados; y es este diagrama de estados precisamente el que describe el funcionamiento del circuito.

Ejemplo 1.18

Analice el circuito que se muestra en la figura 1.43.

Lo primero que se debe hacer es comparar el bloque de la arquitectura de una máquina secuencial sincrónica (máquina Mealy) con el circuito que se va a analizar. De este análisis se identifican los bloques que contiene el circuito, y cuáles son las entradas y salidas de cada bloque, es decir, del bloque del decodificador de estado siguiente y del bloque de decodificación de las salidas.

Una vez identificado cada bloque, se procede a encontrar las ecuaciones booleanas para las entradas de cada bloque.

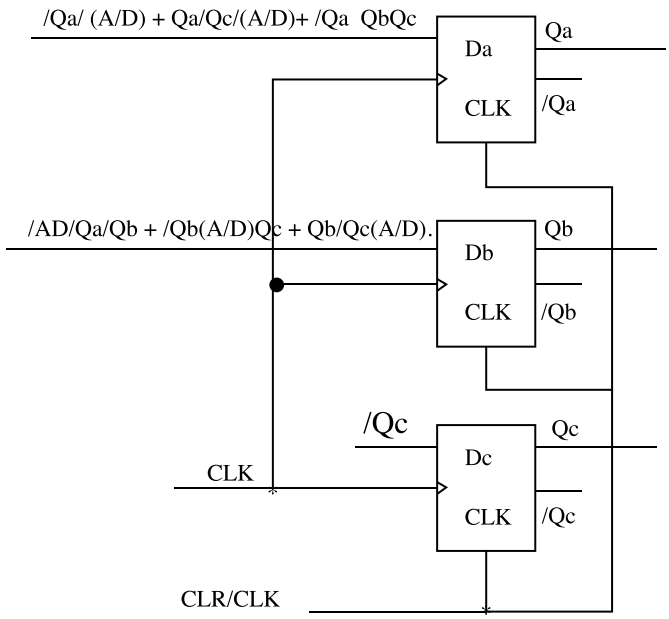


Figura 1.43. Circuito propuesto para analizar

De la figura 1.43, las entradas al decodificador de siguiente estado son Da, Db y Dc Al decodificador de siguiente estado llegan las señales del mundo exterior así como las salidas de los *flip-flops*.

Las entradas a los *flip-flops* son Da, Db y Dc, las salidas del circuito son las mismas salidas de los *flip-flops*, por lo tanto, el decodificador de salidas no existe y no hay que escribir las ecuaciones booleanas para este.

Las ecuaciones booleanas para cada entrada son:

Da= /Qa /(A/D) + Qa/Qc/(A/D)+ /Qa QbQc

Db= /AD/Qa/Qb + /Qb(A/D)Qc + Qb/Qc(A/D)

Dc= /Qc

Examinandas las ecuaciones anteriores, se puede concluir que la señal A/D debe ser la entrada desde el mundo exterior, puesto que, a las entradas del decodificador de siguiente estado llegan las entradas desde el mundo exterior así como las salidas de los *flip-flops*, para este caso, las salidas de los *flip-flops* son Qa, Qb, Qc y A/D debe ser la entrada desde el mundo exterior.

A partir de estas ecuaciones se elabora la tabla característica y es la que se muestra en la tabla 2.10. De la tabla 2.10, se elabora el diagrama de estados. Para llevar a cabo esta tarea, se va leyendo fila por fila la tabla y dibujando cada estado. Por ejemplo, si A/D= 0 y Qa, Qb y Qc son 000 respectivamente, entonces el estado siguiente es Qa+1 Qb+1 Qc+1= 111 respectivamente.

A/D	Qa	Qb	Qc	Qa+x	Qb+1	Qc+1	Da	Db	Dc
0	0	0	0	1	1	1	1	1	1
0	0	0	1	1	1	0	1	1	0
0	0	1	0	1	0	1	1	0	1
0	0	1	1	1	0	0	1	0	0
0	1	0	0	0	1	1	0	1	1
0	1	0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	0	0	1

0	1	1	1	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	0	1	0
1	0	1	0	0	1	1	0	1	1
1	0	1	1	1	0	0	1	0	0
1	1	0	0	1	0	1	1	0	1
1	1	0	1	1	1	0	1	1	0
1	1	1	0	1	1	1	1	1	1
1	1	1	1	0	0	0	0	0	0

Tabla 1.10. Entradas a los *flip-flops*

Si A/D= 0 y Qa, Qb y Qc son 001 respectivamente, entonces el estado siguiente es Qa+1 Qb+1 Qc+1= 000 respectivamente.

Si A/D= 0 y Qa, Qb y Qc son 010 respectivamente, entonces el estado siguiente es Qa+1 Qb+1 Qc+1= 001 respectivamente.

Si A/D= 1 y Qa, Qb y Qc son 001 respectivamente, entonces el estado siguiente es Qa+1 Qb+1 Qc+1= 010 respectivamente.

Si A/D= 1 y Qa, Qb y Qc son 111 respectivamente, entonces el estado siguiente es Qa+1 Qb+1 Qc+1= 000 respectivamente.

Este proceso se realiza con todas las filas de la tabla. El gráfico del diagrama de estados es el que muestra la figura 1.44.

Es necesario recalcar que este proceso de análisis solo se puede aplicar a circuitos secuenciales sincrónicos.

La forma más simple de reconocer a un circuito secuencial sincrónico es observando si en el diagrama de circuito hay un banco de *flip-flops* y si todos estos *flip-flops* tienen un reloj común.

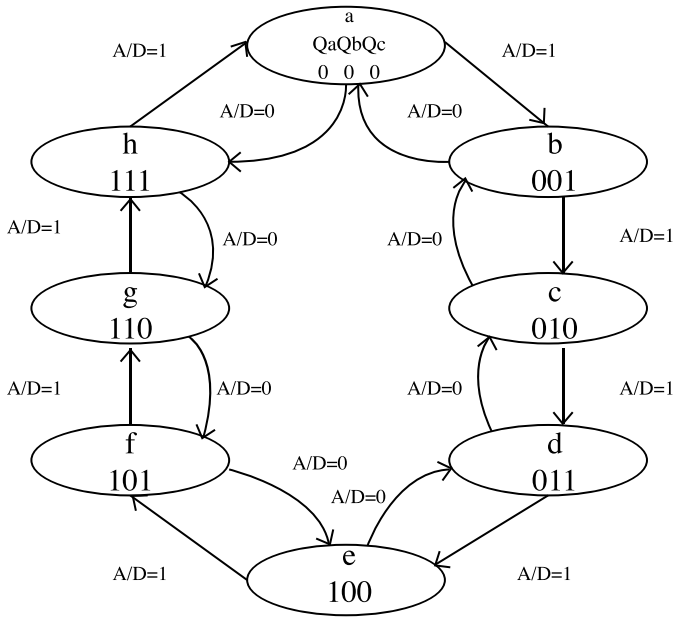


Figura 1.44. Diagrama de estados del circuito analizado

Analizando el diagrama de estados de la figura 1.44 se puede concluir que, cuando A/D = 1, las salidas generan el código: 000-001-010-011-100-101-110-111.

Cuando A/D=0, el código que se genera es el 111-110-101-100-011-010-001-000.

En cada estado, se verifica la señal A/D, si esta es igual a uno, el circuito pasa a un estado siguiente cuyo código es uno más el valor del código del estado presente; si A/D es cero, el circuito va a un estado siguiente cuyo código es uno menos el código del estado presente; por lo tanto, este dispositivo incrementa o decrementa el valor de sus salidas dependiendo de la señal A/D. En definitiva, el circuito es un contador de incremento y decremento controlado por la señal A/D.

1.7 Ejercicios propuestos

1. ¿Qué es un “diagrama de estados?
2. Explique cada parte que conforma un estado.

3. ¿Cómo se identifica temporalmente un estado?

4. ¿Cómo se identifica un estado en un elemento de memoria?

5. Diseñe un circuito que habrá una cerradura eléctrica cuando el código 1, 3, 4, 5, ha sido detectado, si un código erróneo se introduce el circuito debe ir a un estado de falla y permanecer en ese estado hasta que una señal de *reset* sea presionada por el usuario.

6. Un diagrama de estados utiliza tres bits en su código, con tres bits se pueden tener hasta ocho estados diferentes; si solo se utilizan seis de los ocho códigos, explique ¿qué podría pasar si el circuito cae en alguno de los dos códigos que no se utilizan?

7. Proponga una solución para que el circuito del ejercicio anterior se recupere automáticamente cuando caiga en un estado que no esté definido.

8. Diseñe un circuito secuencial sincrónico que obtenga el complemento a dos de un número de cuatro bits. Los bits llegan uno a uno con cada ciclo del reloj. Utilice*flip-flops* JK para implementar el circuito.

9. Diseñe un circuito secuencial sincrónico que genere cuatro ondas cuadradas desfasadas 90 grados una de la otra.

10. Diseñe un circuito secuencial sincrónico que detecte la secuencia 01110 sin traslapamiento. Utilice *flip-flops* tipo T para implementar el circuito.

11. Diseñe un circuito que genere en forma indefinida la secuencia: 0000-0001-1110-1111- 1100. Utilice *flip-flops* tipo D para implementar el circuito.

12. Las señales que enciende en sus salidas un circuito secuencial sincrónico tienen una duración de un ciclo del reloj. Explique cómo conseguiría que las salidas solo duren la mitad del período del reloj.

13. Si se conectan dos *flip-flops* tipo D en una estructura maestro-esclavo ¿qué se obtiene?

1.8 Bibliografía

Baldeón, W. (2010). Problemas resueltos de sistemas digitales. Riobamba:Ecopycenter.

Baldeón, W. (2010). Análisis de señales. Riobamba: Ecopycenter.

Maxinez, D. (2014) Programación de sistemas digitales con VHDL. México DF: Patria.

Sánchez, L. (2018). Problemas de electrónica digital. Valencia: Universidad Politécnica de Valencia.

Dorf, R. y Bishp, R., (2005), Sistemas de Control Moderno. Madrid. España: Pearson.

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

Morris, M. (2003). Diseño digital. México DF: Mc Graw Hill.

Wakerly, J. (2001). Diseño digital principios y prácticas. México DF: Prentice Hall.

2. INTRODUCCIÓN A QUARTUS II

2.1 Introducción

Quartus II es una aplicación informática desarrollada por Altera Corporation, mediante la cual se programan dispositivos lógicos programables como FPGA, CPLD, entre otros. Pertenece al grupo de herramientas denominadas Computer Aided Design (CAD). Altera desarrolló Quartus II para programar los dispositivos lógicos programables que fabrica. Quartus II es un *software* que además permite analizar y sintetizar programas escritos en un lenguaje de descripción de *hardware* (HDL).

Quartus II se encuentra en la actualidad (año 2018) en la versión 15 (V15.0). Las diferentes versiones se encuentran en la página de Altera y están disponibles desde la v2.2 hasta la v15.0.

2.2 Como crear un nuevo proyecto en Quatrus II

Antes de crear un nuevo proyecto, se requiere tener alguna de las versiones de Quartus II, para lo cual hay que acceder a la página web de Altera (www.altera.com), crear una cuenta y descargar alguna versión de Quartus II.

Hay dos ediciones de Quartus II. La una es la versión web que es gratuita y la otra es la de suscripción.

Para cualquiera de las ediciones, la ventana de descarga es tal como se indica en la figura 2.1. Allí se elige una y la descarga inicia presionando el botón *download*.

Figura 2.1. Ventana de descarga de Quartus II

Una vez instalado este *software*, la primera pantalla de Quartus II que aparece es la que se indica en la figura 2.2. Desde aquí, se puede crear un nuevo proyecto. En la parte central, aparece la versión de descarga y en el lado superior izquierdo el *Software* de diseño. Quartus tiene dos opciones el asistente para crear un nuevo proyecto (*new Project wizard*) y abrir un proyecto (*open Project*).

Figura 2.2. Ventana inicial de Quartus II

Para crear un nuevo proyecto, se hace clic sobre *new Project wizard* o se puede también hacer clic sobre: *file* que está en la barra del menú y elegir: *new Project wizard*, con lo que aparece la ventana que se indica en la figura 2.3.

Esta ventana indica que el asistente le permite configurar algunas opciones para el nuevo proyecto, entre estas se tiene:

Figura 2.3. Ventana con las opciones de configuración

Crear un directorio y asignar un nombre al proyecto, nombrar a la entidad de más alto nivel, incluir archivos de algún otro proyecto y librerías, la familia del dispositivo y el dispositivo programable elegido.

En la ventana demostrada en la figura 2.3 se hace clic sobre *next*. Aparece la ventana 2.4, aquí se debe crear un directorio de trabajo que contendrá el proyecto. Para el caso de la figura 2.4 el directorio es: ejemplos qII, el nombre del proyecto es: mi_primer_proyecto. Este nombre automáticamente se escribe en la pestaña que está inmediatamente abajo, esto es, automáticamente se escribió el nombre de la entidad de más alto nivel.

Sin embargo, el nombre de esta entidad puede ser modificado, y puede ser distinto del nombre del proyecto, pero hay que tener especial cuidado en elegir el nombre de la entidad ya que este debe ser el mismo nombre que tenga la entidad de más alto nivel.

Figura 2.4. Ventana para crear un directorio de trabajo

La entidad de más alto nivel es la primera entidad que se declare en el programa. Cuando se instala Quartus II, se crea un directorio por obligación y se llama: “qdesigns”.

Al hacer clic sobre *next* en la ventana de la figura 2.4, aparece la ventana que se muestra en la figura 2.5. Aquí hay dos alternativas: se puede elegir un proyecto modelo o uno vacío. Para el primer caso, Quartus II elige automáticamente un proyecto modelo y lo único que se debe hacer es hacer clic sobre el botón *next* hasta que aparezca una ventana: *summary* en donde se encuentre activo solo el botón *finish*.

Figura 2.5. Ventana de elección del proyecto

Para la segunda alternativa, el proceso es idéntico. Se debe hacer clic sobre el botón *Next* hasta que aparezca la ventana de resumen (*summary*) que tiene activo solo el botón *Finish*, figura 2.5. En este caso, se ha dejado también que Quartus II elija la configuración por *Default*. Si se va a trabajar con alguna familia en particular y algún dispositivo específico o se desea incluir alguna otra herramienta EDA o algún archivo, se debe elegir la alternativa primera y configurarar cada ventana que va apareciendo.

En cualquier caso una vez que se llegó a la ventana que se muestra en la figura 2.5 se debe hacer clic sobre el botón *Finish* y aparece la ventana 2.6 en la que se puede empezar a desarrollar el proyecto.

Figura 2.6. Ventana de resumen

Figura 2.7. Ventana de trabajo de Quartus II

2.3 Diseño con Quartus II

Hay dos formas de ingresar un diseño en Quartus II. La primera se llama captura esquemática y la segunda, mediante la escritura de código (un programa).

2.3.1 Captura esquemática

La captura esquemática es un proceso muy parecido al que se desarrolla con un simulador. El diseño se inicia con el ingreso del gráfico del circuito en la ventana de trabajo. A este circuito se aplican señales en sus entradas, se simula y esta simulación hace que el circuito responda generando las respectivas señales de salida.

Con Quartus II (QII), para ingresar un circuito diseñado, se parte de la ventana de la figura 2.7, allí se elige *file*/*new* como se indica en la figura 2.8 y aparece la ventana que se indica en la figura 2.9.

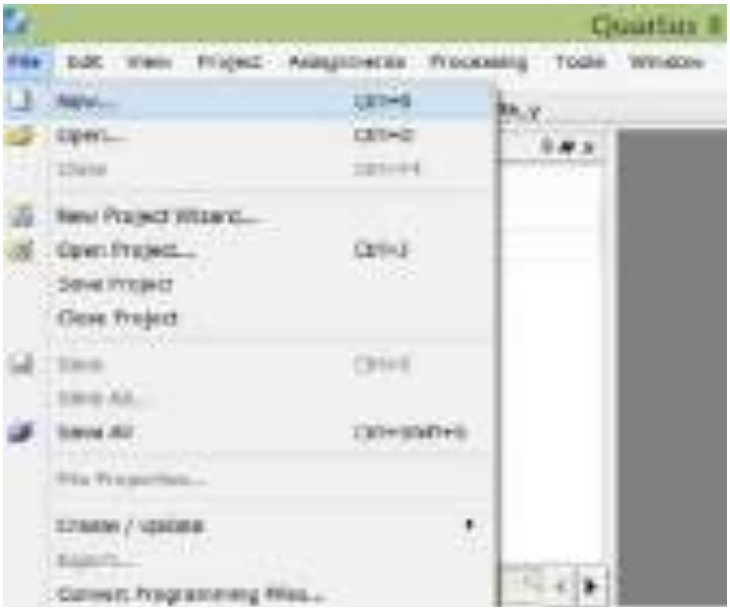


Figura 2.8. Ventana para crear un nuevo archivo

En esta ventana, se tienen varias opciones y son:

- *Design files* (archivos de diseño)
- *Memory files* (archivos de memoria)
- *Verification/debugging* (archivos de verificación y depuración)
- *Others file* (otros archivos)

Para utilizar la captura esquemática, se selecciona la alternativa: *Design files*, dentro de esta opción se encuentra *Block Diagram/Schematic File*. Al seleccionar esta opción y hacer clic sobre el botón *ok* aparece la ventana que se muestra en la figura 2.10.

Esta ventana tiene tres áreas. La superior izquierda tiene información sobre las características del circuito que se va a diseñar como el nombre del proyecto, el FPGA elegido, entre otra información. El área superior derecha es la de trabajo. Allí es donde se construye el circuito (se dibuja) elemento por elemento. El área inferior es el lugar donde aparecen los mensajes de error y avisos.

Figura 2.9. Ventana para seleccionar la captura esquemática

Figura 2.10. Ventana en donde se ingresa el diseño

2.3.2 Ingresando un circuito con captura esquemática

La captura esquemática se expone mediante un ejemplo. Este consiste en el diseño de un circuito que contiene una única puerta AND. La captura esquemática se inicia con la apertura de la ventana de la figura 2.10.

El ejercicio es simple, pero pretende ser muy ilustrativo. Vale indicar que, para la captura esquemática de circuitos mucho más complicados, solo hay que repetir estos pasos.

Para ingresar el diseño (circuito) en el área de trabajo, se puede elegir una de las siguientes tres opciones:

- Hacer clic sobre el icono de la puerta AND en la barra que se encuentra justo sobre el área de trabajo (área con cuadrícula).
- Hacer clic dos veces sobre el área de trabajo con el botón izquierdo.
- Hacer clic sobre el área de trabajo con el botón derecho y seleccionar *insert*/*symbol*.

Independientemente del método elegido se abre la ventana *symbol* que se indica en la figura 2.11. Allí se debe hacer clic en la flecha del icono de la carpeta.

Figura 2.11. Ventana symbol

En la figura 2.12, se muestra la carpeta desplegada, allí se abre la carpeta *primitives* y se selecciona la carpeta *logic*. Dentro de esta carpeta, se encuentran todos los circuitos lógicos que están a disposición para crear un circuito. Para este caso se elige la puerta AND de dos entradas y aparece la ventana *symbol* de la figura 2.12. Se hace clic sobre *ok* y se tiene la ventana 2.13. La puerta AND se puede colocar en cualquier parte del área de trabajo con solo hacer clic en el lugar elegido.

Si el circuito contiene muchas puertas, el proceso se repite para cada circuito. Una vez que se tengan todos los circuitos en el área de trabajo se procede a unir las entradas y salidas de cada uno de ellos hasta que se haya dibujado el circuito completo deseado.

Figura 2.12. Selección del circuito lógico

Figura 2.13. Puerta AND ubicada en un lugar del área de trabajo

El paso siguiente es añadir los pines de entrada y salida del circuito. Para este caso, como el circuito está compuesto solo por una compuerta de dos entradas y una salida, se deben poner dos pines de entrada y un pin de salida.

Se activan las ventanas de las figuras 2.12 y 2.13 como se indicó antes. Ahora se elige *pin*/*input* de la carpeta *primitives* como se indica en la figura 2.14. Se hace clic en *ok* y se tiene la ventana 2.15.

Este procedimiento se repite para cada pin. Lógicamente, para la salida, se debe elegir *pin*/*output*.

Figura 2.14. Ventana *symbol* para seleccionar un pin de entrada

Se debe seleccionar la ubicación donde se pondrá el pin de entrada. Para esto, se hace clic sobre el lugar seleccionado.

Figura 2.15. Ventana de trabajo con un pin de entrada

El circuito con sus pines de entrada y salida se ve en la figura 2.16. Se unen los pines de entrada con cada entrada al circuito y lo mismo se hace con las salidas. El resultado se visualiza en la ventana de la figura 2.17.

En la figura 2.17, el circuito está listo para ser compilado y simulado.

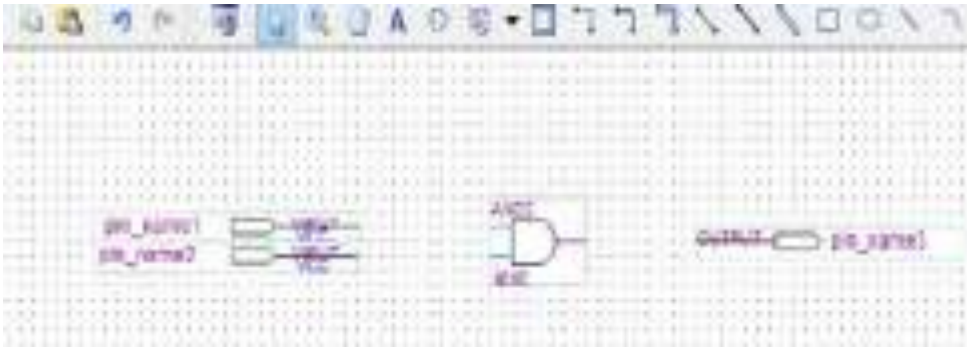


Figura 2.16. Circuito con sus entradas y salidas

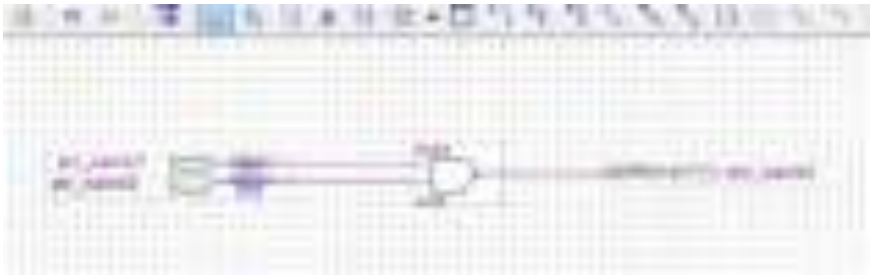


Figura 2.17. Circuito con sus entradas y salidas conectadas

2.3.3 Compilación del circuito diseñado

El circuito de la figura 2.17 debe ser compilado para detectar errores de sintaxis en el código. Se indica tres formas de compilar.

Figura 2.18. Ventana para la compilación

- Primera: se selecciona *processing/star/star Analysis & Synthesis*.
- Segunda: en esta misma ventana, se puede seleccionar *Start Compilation*.
- Tercera: se puede hacer clic sobre el botón de color azul en forma de triángulo. El círculo verde en la figura 2.18 rodea este triangulo.

Ejecutada la simulación, pregunta si desea guardar los cambios hechos. Se debe contestar sí. Luego se debe guardar el proyecto y finalmente se abrirá la ventana 2.19 si no hay errores. Si es así, debe aceptar. Si hay errores, debe corregir y volver a compilar.

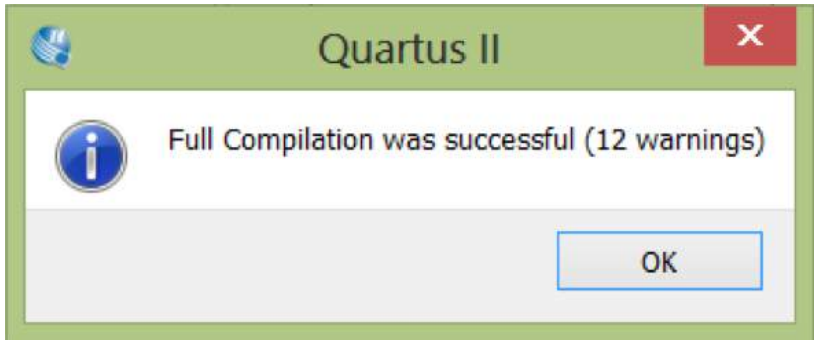


Figura 2.19. Ventana indicando que la compilación fue exitosa

2.3.4 Ingreso de las señales de entrada

Como en toda simulación, las señales de entrada al circuito son los estímulos que permiten verificar la respuesta del circuito. Para esto, se hace clic en *file/new* y aparece la ventana de la figura 2.19. Allí se selecciona *University Program VWF* que se encuentra en *Verification/debugging files*. Se hace clic sobre *ok* y aparece la figura 2.20.

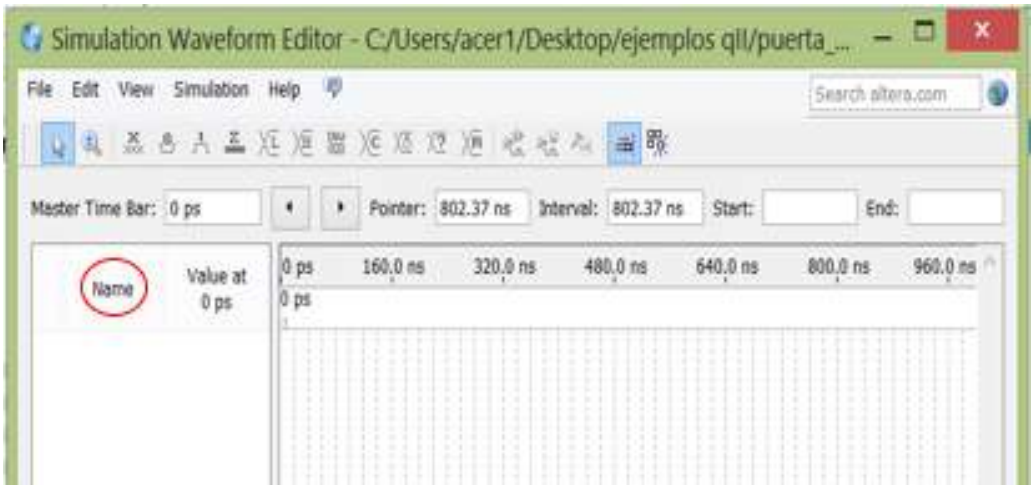


Figura 2.20. Ventana del VWF

En esta ventana se elige *Edit/Insert/Inser node or bus*. Aparece la ventana 2.21. Otra forma de llegar al menú de la figura 2.20 es hacer clic dos veces con el botón izquierdo del *mouse* en la región en blanco debajo de *name*.

Se lista los nodos; se hace clic sobre *node finder*, y aparece la ventana 2.22. En esta ventana la pestaña *filter* debe estar seleccionando *pins: all*. Finalmente, se hace clic sobre el botón que tiene el símbolo >>.

Figura 2.21. Ventana para insertar las señales de entrada

Figura 2.22. Ventana con los pines de entrada seleccionados

A continuación, se hace clic sobre *ok*. En la ventana que aparece, se vuelve a hacer clic en *ok* y se obtiene la ventana 2.23.

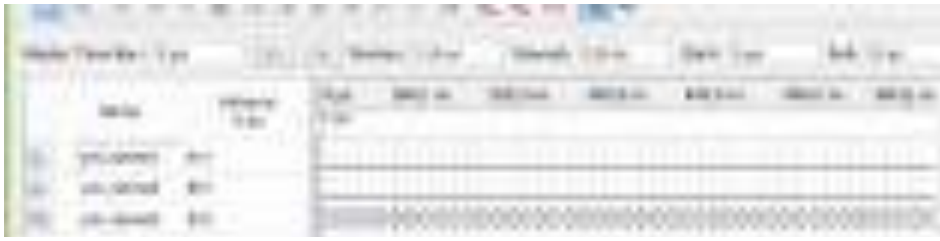


Figura 2.23. Ventana con las entradas y salidas del circuito

En eje del tiempo, las líneas llenas representan a las entradas (pin_name1, B0, pin_name2, B0) y las salidas tienen la forma de malla, en este caso pin_name3 (B X). Los nombres de los pines de entrada y salida (pin_name1, pin_name2, pin_name3) que se muestran en la figura 2.23 son los nombres que Quartus II pone por

default y se pueden cambiar. Para cambiar los nombres de las señales, en la que se indica en la figura 2.17, se hacer clic con el botón derecho sobre la señal de interés, por ejemplo pin_name1, y, en la ventana que aparece se cambia el nombre. Por ejemplo, por a. Se repite este procedimiento para cada entrada y salida.

2.3.5 Configuración del eje del tiempo

Para establecer el tiempo de duración de la simulación, en la ventana de la figura 2.23, se debe seleccionar *Edit/Set end time,* y aparece la ventana que muestra la figura 2.24. Como ejemplo, se selecciona *End Time* igual a 160 nanosegundos (ns).

Figura 2.24. Configuración del tiempo de simulación

Finalmente, se obtiene la ventana de la figura 2.25. Allí se puede notar que el eje del tiempo va de 0 a 160 ns.

Figura 2.25. Configuración del tiempo a 160 ns

2.3.6 Poniendo niveles lógicos en las señales de entrada

Hay diferentes formas de poner valores lógicos (1 o 0) en los pines de entrada. Por ejemplo, una forma manual es seleccionando el área donde se va a poner el valor lógico.

Se presiona el botón izquierdo y se arrastra el ratón. El área seleccionada cambia de color como muestra la ventana en la figura 2.26. A continuación, se

hace clic sobre el icono que tiene la forma de un nivel alto (1), y este rango del tiempo queda con ese valor como muestra la figura 2.27.

Figura 226. Asignando valores lógicos en las entradas

Figura 2.27. Asignando un uno lógico en una señal de entrada

En esta misma figura, se puede ver que hay otras opciones de niveles lógicos como: x que simboliza un valor desconocido; z, alta impedancia, etc.

Para verificar el correcto funcionamiento de un circuito, se debe verificar la respuesta del circuito para todas las combinaciones posibles de los niveles lógicos en sus entradas.

Para dos entradas hay cuatro combinaciones posibles y son: 00, 01, 10 y 11. Para n entradas hay n combinaciones posibles. En general, si es posible, hay que simular el circuito para todas las combinaciones posibles de sus entradas. Si alguna o algunas combinaciones no son probadas no habrá la completa certeza de que, para esas combinaciones, funcione bien el circuito.

Para el caso de la puerta AND, esta debe cumplir con las cuatro combinaciones de la tabla de verdad 2.1. Las señales de entrada deben tener esas combinaciones.

Hay formas de poner niveles lógicos automáticamente. Una de ellas es mediante un contador. Un contador, en sus salidas, genera todas las combinaciones posibles. Por ejemplo un contador de dos bits (dos salidas) pone en sus dos salidas las combinaciones 00, 01, 10, 11.

Para utilizar un contador como generador de señales, es conveniente primero agrupar las señales de interés. Para llevar a cabo esto, se seleccionan las señales con el botón izquierdo y se hace clic sobre ellas con el botón derecho y se selecciona *grouping*/*group*, y aparece la ventana de agrupamiento que muestra la figura 2.28. Allí se escribe un nombre en: *group name* y se presiona el botón OK.

Figura 2.28. Ventana de agrupamiento

A continuación, se seleccionan las señales agrupadas y se escoge desde la barra de menú *edit*/*value*/*count value*, y aparece la ventana que se muestra en la figura 2.29.

Figura 2.29. Ventana para colocar los valores de la cuenta

Se colocan los valores adecuados en la ventana de la figura 2.30. Por ejemplo, la cuenta es binaria, el valor inicial de la cuenta es cero (00), el incremento

en la cuenta es de una unidad, y la cuenta es cada 250 ns (*count every*), porque, el eje del tiempo tiene 1000 ns y en ese tiempo deben aparecer las cuatro combinaciones de la cuenta, una cada 250 ns.

Se presiona el botón OK en la ventana mostrada en la figura 2.29 y la figura 2.30 muestra las señales de entrada agrupadas y ponderadas.

Figura 2.30. Señales de entrada agrupadas y ponderadas

Pin:name1	Pin:name2	Pin_name3
0	0	0
0	1	0
1	0	0
1	1	1

2.3.7 Simulación del circuito diseñado

Finalmente el último paso es la simulación. Hay dos tipos de simulación: la funcional y la temporal. La funcional es una simulación ideal en que los retardos de propagación son cero. En cambio, en la simulación temporal, se incluyen los retardos. Para simular el circuito, se hace clic sobre el botón *run functional simulation* (botón encerrado en un círculo de color rojo en la figura 2.27), y se obtiene la ventana que se muestra en la figura 2.31. Esta es una ventana de simulación que contiene la forma de onda de la salida del circuito.

De la forma de onda de la salida se concluye que el circuito funciona en forma adecuada, ya que responde adecuadamente a las cuatro combinaciones posibles de las entradas. Mientras se va ejecutando la simulación, la aplicación pide que se guarde el proyecto.

Figura 2.31. Ventana de simulación

En las versiones de QII inferiores a la versión 13, el proceso de compilación y simulación es un poco diferente y más largo. Por lo tanto, si se está trabajando con alguna de estas versiones, se debe consultar la forma de compilar y simular.

2.4 Ingreso de código VHDL en Quartus II

El procedimiento es similar al de captura esquemática. Se debe, primero, crear un proyecto. Una vez creado, en la barra de menú seleccionar *File/New* y aparece la ventana *New*. Allí se debe seleccionar: *VHDL FILE* de la opción *Design Files*. Hecho esto, aparece la ventana de la figura 2.32, donde se puede ver parte del código de un programa en VHDL que se ha escrito. La compilación, ingreso de formas de onda y simulación es exactamente igual a como se realizó para el caso de captura esquemática.

Figura 2.32. Ventana para ingresar el código HDL

La diferencia con la captura esquemática está en que no se dibuja un circuito sino más bien se escribe un programa en código HDL (lenguaje de descripción de *hardware*), que describe el funcionamiento de un circuito o sistema. Luego, esta codificación se compila y se simula de forma idéntica a como se hizo en la captura esquemática; es decir, para compilar y simular, no hay diferencia.

2.5 La plantilla de Quartus II

Quartus II permite insertar plantillas de lenguajes como: VERILOG HDL, SystemVerilog, VHDL, AHDL, Quartus II Tcl, Tcl, TimeQuest design constraints, o Megafunction template. Estas plantillas se pueden insertar en cualquier parte de un texto, y facilitan la escritura de código de cualquiera de los lenguajes indicados.

La figura 2.33 muestra el icono *insert template* (se parece a un pergamino) sobre el que se debe hacer clic para desplegar el contenido de la plantilla.

Figura 2.33. Icono de *insert template*

Figura 2.34. Ventana *insert template*

La figura 2.34 muestra el contenido de la plantilla. Como se puede apreciar, permite elegir ocho plantillas de ocho lenguajes.

Al desplegar VHDL de la ventana *insert template*, se tiene la ventana que se indica en la figura 2.35. Dentro de VHDL, se puede elegir *Constructs*, *Design Units* y allí están las plantillas para declarar cada uno de los módulos de VHDL. A continuación, se muestra el contenido de *Library Clause*.

Figura 2.35. Despliegue de VHDL

Contenido de *Library Clause*:

-- A library clause declares a name as a library. It

-- does not create the library; it simply forward declares

-- it.

library <library_name>;

Este contenido se puede insertar en el área de trabajo de la ventana en donde se escribe el código de VHDL al hacer clic en el botón insertar que se muestra en la figura 2.36.

Figura 2.36. Ventana con la plantilla de Library

El Contenido devUse Clause es:

-- Use clauses import declarations into the current scope.

-- If more than one use clause imports the same name into the

-- the same scope, none of the names are imported.

-- Import all the declarations in a package

use <library_name>.<package_name>.all;

-- Import a specific declaration from a package

use <library_name>.<package_name>.<object_name>;

-- Import a specific entity from a library

use <library_name>.<entity_name>;

-- Import from the work library. The work library is an alias

-- for the library containing the current design unit.

use work.<package_name>.all;

-- Commonly imported packages:

-- STD_LOGIC and STD_LOGIC_VECTOR types, and relevant functions

use ieee.std_logic_1164.all;

```
-- SIGNED and UNSIGNED types, and relevant functions
use ieee.numeric_std.all;
-- Basic sequential functions and concurrent procedures
use ieee.VITAL_Primitives.all;
-- Library of Parameterized Modules:
-- customizable, device-independent logic functions
use lpm.lpm_components.all;
-- Altera Megafunctions
use altera_mf.altera_mf_components.all;
```

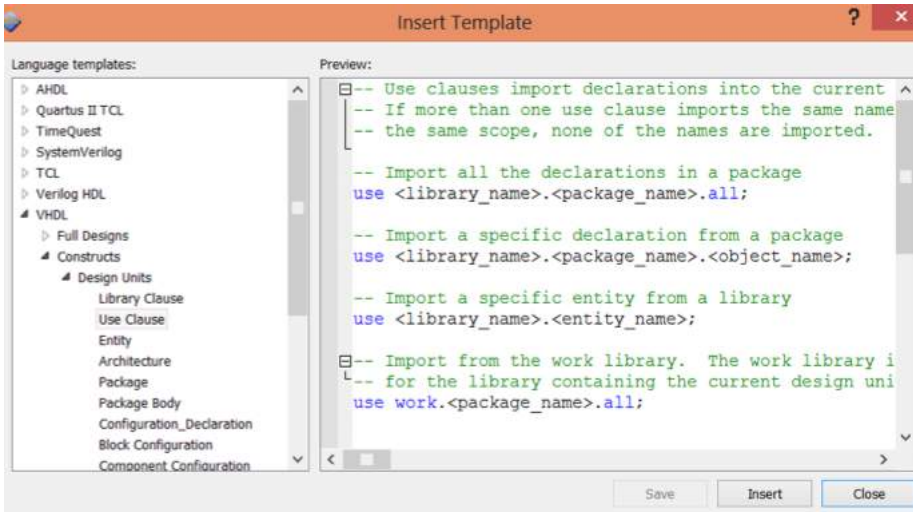


Figura 2. 37 Contenido de *Use Clause*

A continuación se muestra el contenido de *Entity* de la opción *Design Units*. Como se puede ver esta opción permite insertar la plantilla con la sintaxis de declaración de entidad y adaptarla a los requerimientos de cada diseño en particular.

```
entity <entity_name> is
    generic
    (
            <name>          : <type>  :=     <default_value>;
```

```
        ...
        <name>          : <type>  :=     <default_value>
    );

    port
    (
        -- Input ports
        <name>          : in  <type>;
        <name>          : in  <type> := <default_value>;

        -- Inout ports
        <name>          : inout <type>;

        -- Output ports
        <name>          : out <type>;
        <name>          : out <type> := <default_value>
    );
end <entity_name>;
```

La figura 2.38 muestra la ventana cuyo contenido es la declaración de entidad. La plantilla para la declaración de la arquitectura se muestra en la figura 2.39, igual que para el *reset* o de las opciones esta plantilla se puede adaptar a cualquier requerimiento de diseño.

Figura 2.38. Ventana con el contenido de declaración de entidad

El contenido de la plantilla de declaración de arquitectura se muestra a continuación:

```
architecture <arch_name> of <entity_name> is
    -- Declarations (optional)
begin
    -- Process Statement (optional)
    -- Concurrent Procedure Call (optional)
    -- Concurrent Signal Assignment (optional)
    -- Conditional Signal Assignment (optional)
    -- Selected Signal Assignment (optional)
    -- Component Instantiation Statement (optional)


    -- Generate Statement (optional)
end <arch_name>;
```

La ventana con la plantilla de declaración de paquete se muestra en la figura 2.40. Esta plantilla puede ser insertada dentro del código VHDL y adaptada a cualquier diseño de paquete

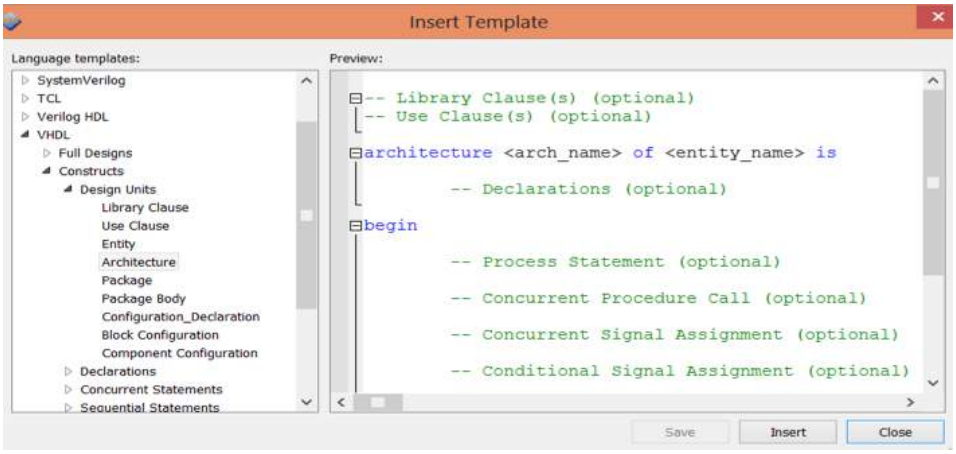


Figura 2.39. Ventana con la plantilla de declaración de arquitectura

La plantilla para la declaración de paquete tiene las características que se muestran a continuación:

```
-- Library Clause(s) (optional)
-- Use Clause(s) (optional)
package <package_name> is
    -- Type Declaration (optional)
    -- Subtype Declaration (optional)
    -- Constant Declaration (optional)
    -- Signal Declaration (optional)
    -- Component Declaration (optional)
end <package_name>;
```

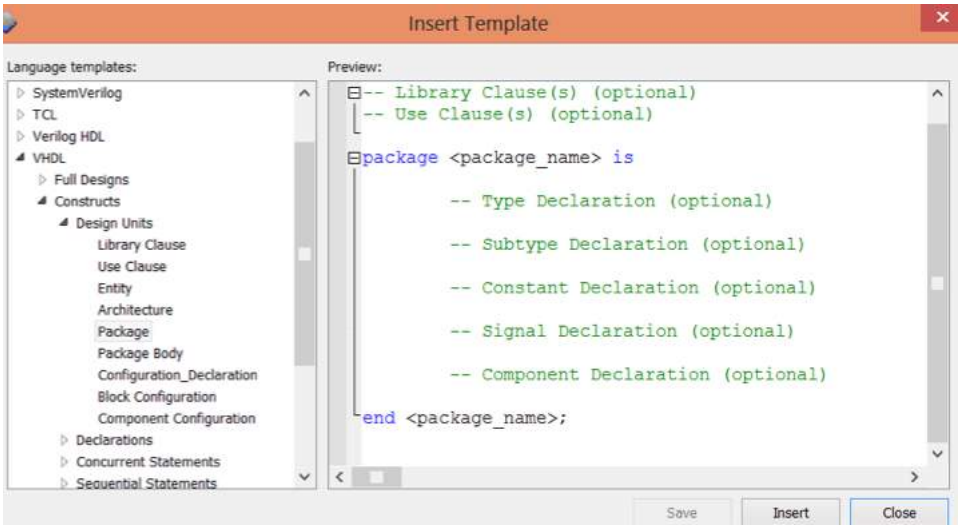


Figura 2.40. Ventana con la plantilla de declaración de paquete

El contenido de la plantilla de declaración de señal es:

-- Signal with no default value. Your design should assign an explicit

-- value to such a signal using an assignment statement. You assign

-- to a signal with the "<=" operator.

signal <name> : <type>;

-- Signal with a default value. If you do not assign a value to the

-- signal with an assignment, Quartus II Integrated Synthesis will

-- initialize it with the default value. Integrated Synthesis also

-- derives power-up conditions for memories and registers from the

-- default value.

signal <name> : <type> := <default_value>;

-- Commonly declared signals

signal <name> : std_logic;

signal <name> : std_logic_vector(<msb_index> downto <lsb_index>);

signal <name> : integer;

signal <name> : integer range <low> to <high>;

La figura 2.41 muestra la ventana con el contenido de declaración de plantilla de declaración de señal.

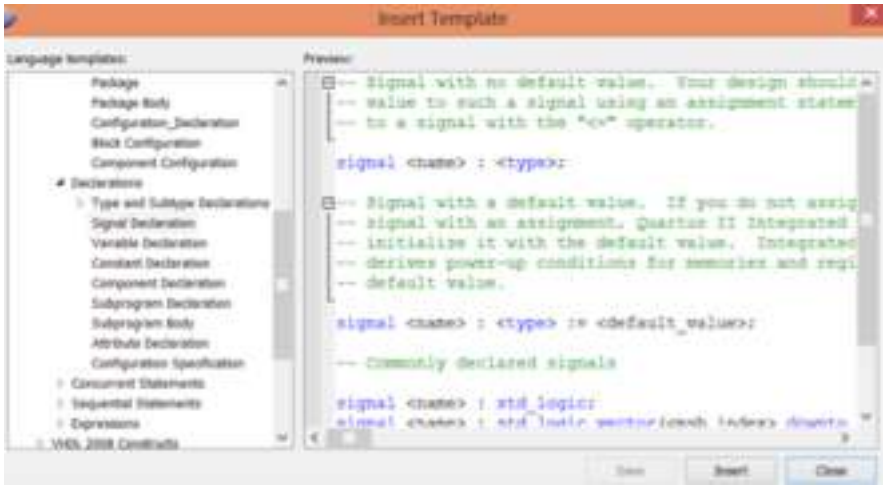


Figura 2.41. Ventana con la plantilla de declaración de señal

A continuación se escribe la sintaxis de declaración de una variable y que, igual que para las plantillas anteriores, puede ser adaptada a cualquier declaración de variable.

-- Variables should be declared in a process statement or subprogram.

-- They are useful for storing intermediate calculations. You assign

-- to a variable with the ":=" operator.

-- Variable with no default value. Your design should assign an

-- explicit value to this variable before referring to it in a

-- statement or an expression

variable <name> : <type>;

-- Variable with a default value.

variable <name> : <type> := <default_value>;

-- Commonly declared variables

variable <name> : std_logic;

variable <name> : std_logic_vector(<msb_index> downto <lsb_index>);

variable <name> : integer;

variable <name> : integer range <low> to <high>;

La figura 2.42 muestra la ventana que contiene la plantilla de declaración de variable.

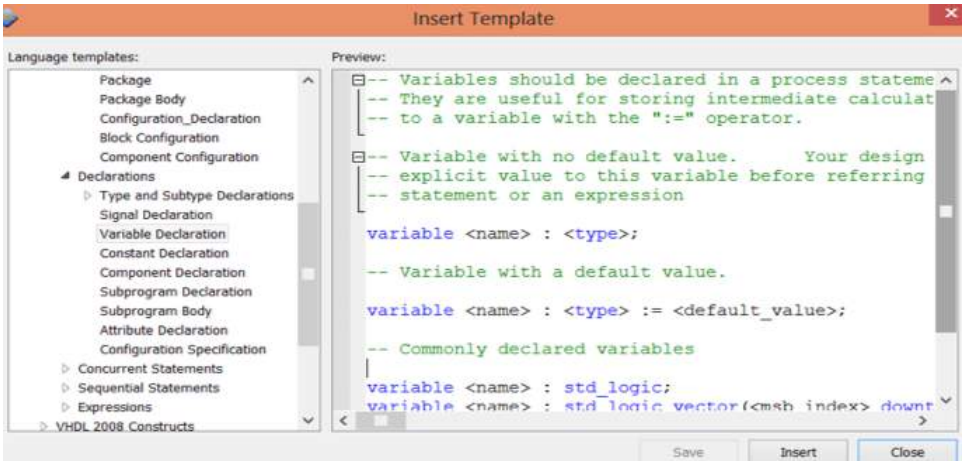


Figura 2.42. Ventana con la plantilla de declaración de variable

La plantilla para la declaración de una constante se muestra a continuación.
constant <constant_name> : <type> := <constant_value>;

La figura 2.43 muestra la ventana que contiene la plantilla indicada, este contenido igual que para el *reset* o de las plantillas puede ser modificada según las necesidades del diseño.

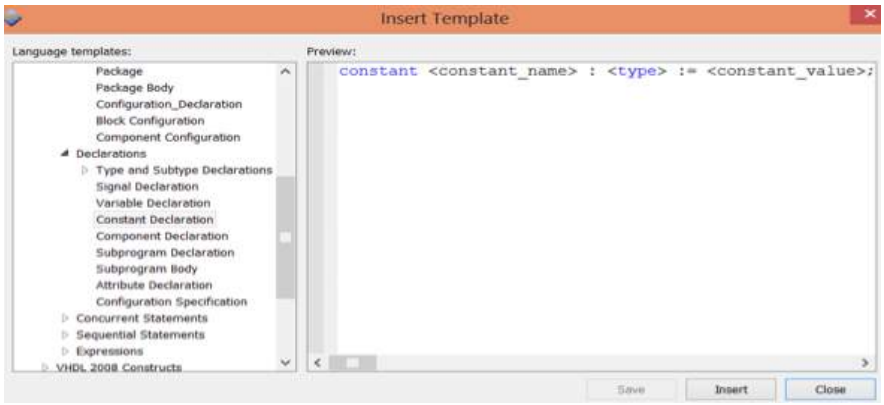


Figura 2.43. Ventana con la plantilla de declaración de constante

2.6 El entrenador DE2 de Altera

El tablero de entrenamiento DE2 de Altera está desarrollado con el FPGA que Altera llama Cyclone II. Todos los componentes que contiene el DE2 están conectados a los pines del Cyclon II.

Este entrenador tiene conectados al Cyclon II los siguientes dispositivos: *switch*, pulsadores, *leds*, *display* de siete segmentos, SDRAM, SRAM, *flash memory*, un *display* de 16x2, el procesador NIOS II, interfaces estándares (RS-232, PS/2), puertos USB, un puerto *ethernet*, un puerto para conectar una tarjeta de memoria SD y conectores para conectar el DE2 a otros dispositivos externos.

2.6.1 Conexiones del entrenador DE2 de Altera

El DE2 se conecta a un computador mediante un cable de datos USB. La energía la recibe a través de una fuente de corriente continua de nueve voltios.

La figura 2.44 muestra una fotografía del DE2. Allí se pueden observar todos los dispositivos que están en este entrenador. Como se puede ver, el cable USB se debe conectar a uno de los dos puertos USB (estos dos puertos están juntos) al puerto que está junto al conector de color negro de energía eléctrica de 9 voltios.

Figura 2.44. El tablero de entrenamiento DE2 de Altera

2.6.2 Instalación del *driver* USB-Blaster

Para conectar físicamente el DE2 con un computador se requiere un cable de datos USB. El DE2 debe ser energizado mediante una fuente de corriente directa de nueve voltios. Para instalar el USB-BLASTER debe seguir los siguientes pasos:

1. Conecte el DE2 al computador a través del cable USB.
2. Conecte el cable USB al conector del DE2 que está más cerca de la fuente de alimentación del tablero.
3. Conecte el DE2 a la fuente de nueve voltios.
4. Presione el botón rojo de energización del DE2.

Al conectar el DE2 al computador, este reconocerá al DE2 rápidamente, pero, si en el computador no está instalado el driver USB-Blaster no será reconocido. Si este es el caso, se debe recurrir al asistente de instalación de nuevo *hardware*.

En el computador, con el botón derecho, se hace clic sobre el icono del equipo y aparece la ventana que muestra la figura 2.44. Se selecciona propiedades y se presenta la ventana de la figura 2.45. En esta ventana, se selecciona administrador de dispositivos y aparece la ventana que se indica en la figura 2.46. En esta se busca el USB_Blaster que está en el icono denominado controlador de bus serie universal o en el icono con la leyenda otros dispositivos, como se muestra en la figura 2. 46.

Figura 2.44 Ventana de propiedades del *equ*

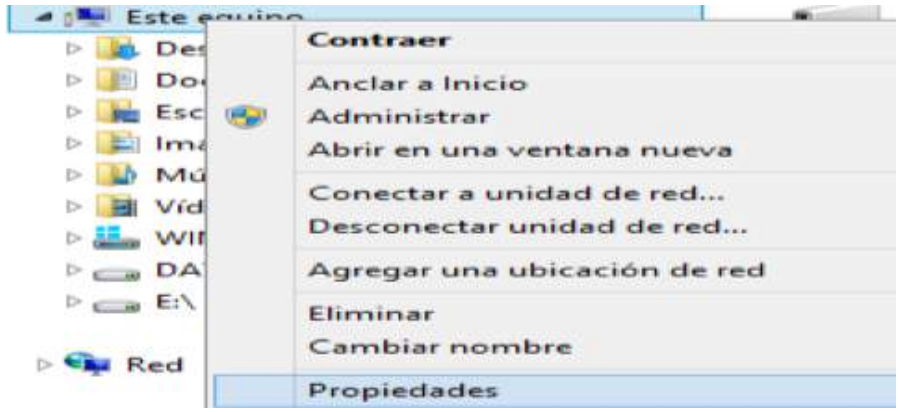


Figura 2.45. Ventana de sistema

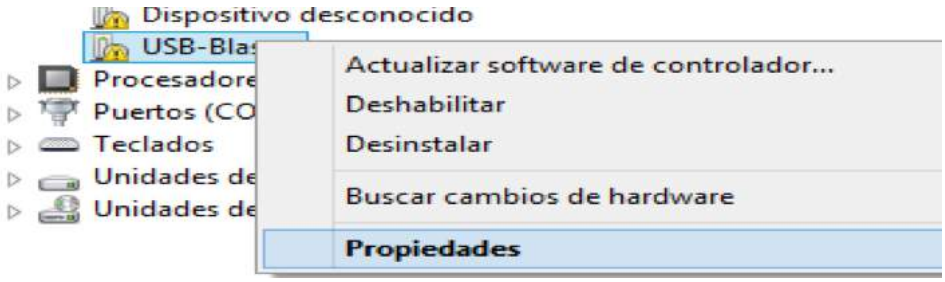


Figura 2.46. Ventana del sistema que contiene el USB-Blaster

En esta ventana, se encuentra el USB-Blaster. Para instalar, se hacer clic sobre este icono con el botón derecho como muestra la figura 2.47 y aparece la ventana que se indica en la figura 2.48.

Figura 2.47. Ventana con el USB-Blaster

En la ventana de la figura 2.48 se selecciona: controlador y allí se hacer clic sobre actualizar controlador y aparece la venta que muestra la figura 2.49.

Figura 2.48. Ventana de selección de propiedades de USB-Blaster

Esta ventana presenta dos opciones: se selecciona buscar *software* de controlador en el equipo y aparece la ventana que muestra la figura 2.50.

Figura 2.49. Ventana de búsqueda de *software*

En la ventana de la figura 2.50, se hace clic sobre el botón examinar para buscar la carpeta Altera, que contiene los drivers USB-Blaster. La figura 2.51 muestra la ubicación de esta carpeta.

Figura 2.50. Ventana de actualización de controlador USB-Blaster

Figura 2.51. Ventana de búsqueda de la carpeta Altera

Se hace clic sobre la carpeta Altera cuyo contenido se muestra en la figura 2.52. Dentro de ella se busca la carpeta Quartus, se la abre y se selecciona la carpeta *drivers* como muestra la ventana de la figura 2.53.

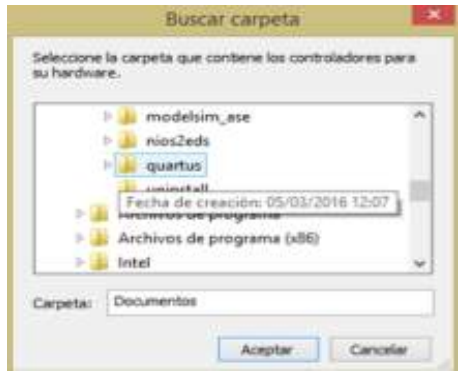


Figura 2.52. Ventana de selección de la carpeta Quartus

Figura 2.53. Ventana de selección de la carpeta *drivers*

Dentro de la carpeta *drivers* se encuentra la carpeta *USB-Blaster* como muestra la figura 2.54. Se selecciona esta carpeta y se preciona el boton aceptar de esta ventana y aparece la ventana que muestra la figura 2.55.

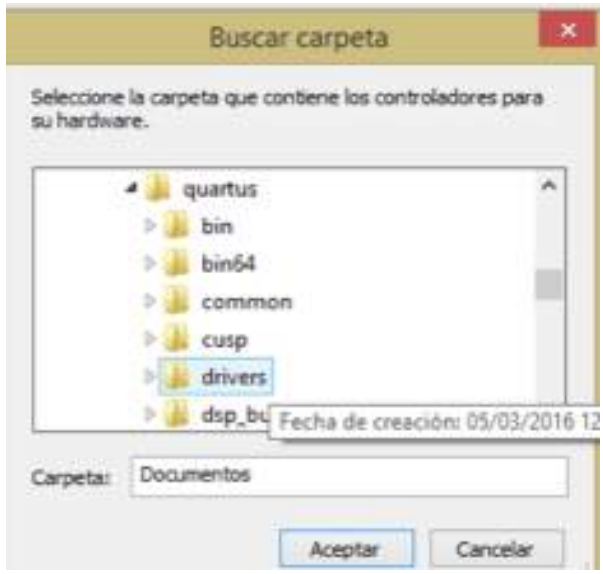


Figura 2.54. Ventana donde se selecciona la carpeta *USB-Blaster*

En la ventana de la figura 2.55, se presiona instalar.

Figura 2.55. Ventana de seguridad de Windows

Finalmente, en la figura 2.56, se muetra la ventana en la que se indica que se finalizó la instalacion del *software* y el USB_Blaster está listo para ser utilizado.

Figura 2.56. Ventana que muestra la instalación correcta del USB-Blaster

2.6.3 Asignación de los pines del Cyclon II en el DE2

A continuación, se muestra la tabla 2.2 de la asignación de los pines del Cyclon II a los diferentes elementos del entrenador DE2 de Altera.

Como puede observarse en la tabla, a cada pin del Cyclon II le corresponde un elemento. Por ejemplo, el PIN_25 está signado o conectado al elemento denominado SW[0], *switch* cero, del entrenador DE2.

Quartus II Version 5.1 Internal Build 160 09/19/2005 TO Full Version

File: D:\de2_pins\de2_pins.csv

Generated on: Wed Sep 28 09:40:34 2005

Note: The column header names should not be changed if you wish to import this .csv file into the Quartus II software.

To	Location
SW[0]	PIN_N25
SW[1]	PIN_N26
SW[2]	PIN_P25
SW[3]	PIN_AE14
SW[4]	PIN_AF14
SW[5]	PIN_AD13
SW[6]	PIN_AC13
SW[7]	PIN_C13
SW[8]	PIN_B13
SW[9]	PIN_A13
SW[10]	PIN_N1
SW[11]	PIN_P1
SW[12]	PIN_P2
SW[13]	PIN_T7
SW[14]	PIN_U3
SW[15]	PIN_U4
SW[16]	PIN_V1
SW[17]	PIN_V2
DRAM_ADDR[0]	PIN_T6
DRAM_ADDR[1]	PIN_V4
DRAM_ADDR[2]	PIN_V3
DRAM_ADDR[3]	PIN_W2
DRAM_ADDR[4]	PIN_W1
DRAM_ADDR[5]	PIN_U6
DRAM_ADDR[6]	PIN_U7
DRAM_ADDR[7]	PIN_U5

DRAM_ADDR[8]	PIN_W4
DRAM_ADDR[9]	PIN_W3
DRAM_ADDR[10]	PIN_Y1
DRAM_ADDR[11]	PIN_V5
DRAM_BA_0	PIN_AE2
DRAM_BA_1	PIN_AE3
DRAM_CAS_N	PIN_AB3
DRAM_CKE	PIN_AA6
DRAM_CLK	PIN_AA7
DRAM_CS_N	PIN_AC3
DRAM_DQ[0]	PIN_V6
DRAM_DQ[1]	PIN_AA2
DRAM_DQ[2]	PIN_AA1
DRAM_DQ[3]	PIN_Y3
DRAM_DQ[4]	PIN_Y4
DRAM_DQ[5]	PIN_R8
DRAM_DQ[6]	PIN_T8
DRAM_DQ[7]	PIN_V7
DRAM_DQ[8]	PIN_W6
DRAM_DQ[9]	PIN_AB2
DRAM_DQ[10]	PIN_AB1
DRAM_DQ[11]	PIN_AA4
DRAM_DQ[12]	PIN_AA3
DRAM_DQ[13]	PIN_AC2
DRAM_DQ[14]	PIN_AC1
DRAM_DQ[15]	PIN_AA5
DRAM_LDQM	PIN_AD2
DRAM_UDQM	PIN_Y5
DRAM_RAS_N	PIN_AB4
DRAM_WE_N	PIN_AD3
FL_ADDR[0]	PIN_AC18
FL_ADDR[1]	PIN_AB18
FL_ADDR[2]	PIN_AE19
FL_ADDR[3]	PIN_AF19

FL_ADDR[4]	PIN_AE18
FL_ADDR[5]	PIN_AF18
FL_ADDR[6]	PIN_Y16
FL_ADDR[7]	PIN_AA16
FL_ADDR[8]	PIN_AD17
FL_ADDR[9]	PIN_AC17
FL_ADDR[10]	PIN_AE17
FL_ADDR[11]	PIN_AF17
FL_ADDR[12]	PIN_W16
FL_ADDR[13]	PIN_W15
FL_ADDR[14]	PIN_AC16
FL_ADDR[15]	PIN_AD16
FL_ADDR[16]	PIN_AE16
FL_ADDR[17]	PIN_AC15
FL_ADDR[18]	PIN_AB15
FL_ADDR[19]	PIN_AA15
FL_ADDR[20]	PIN_Y15
FL_ADDR[21]	PIN_Y14
FL_CE_N	PIN_V17
FL_OE_N	PIN_W17
FL_DQ[0]	PIN_AD19
FL_DQ[1]	PIN_AC19
FL_DQ[2]	PIN_AF20
FL_DQ[3]	PIN_AE20
FL_DQ[4]	PIN_AB20
FL_DQ[5]	PIN_AC20
FL_DQ[6]	PIN_AF21
FL_DQ[7]	PIN_AE21
FL_RST_N	PIN_AA18
FL_WE_N	PIN_AA17
HEX0[0]	PIN_AF10
HEX0[1]	PIN_AB12
HEX0[2]	PIN_AC12
HEX0[3]	PIN_AD11

HEX0[4]	PIN_AE11
HEX0[5]	PIN_V14
HEX0[6]	PIN_V13
HEX1[0]	PIN_V20
HEX1[1]	PIN_V21
HEX1[2]	PIN_W21
HEX1[3]	PIN_Y22
HEX1[4]	PIN_AA24
HEX1[5]	PIN_AA23
HEX1[6]	PIN_AB24
HEX2[0]	PIN_AB23
HEX2[1]	PIN_V22
HEX2[2]	PIN_AC25
HEX2[3]	PIN_AC26
HEX2[4]	PIN_AB26
HEX2[5]	PIN_AB25
HEX2[6]	PIN_Y24
HEX3[0]	PIN_Y23
HEX3[1]	PIN_AA25
HEX3[2]	PIN_AA26
HEX3[3]	PIN_Y26
HEX3[4]	PIN_Y25
HEX3[5]	PIN_U22
HEX3[6]	PIN_W24
HEX4[0]	PIN_U9
HEX4[1]	PIN_U1
HEX4[2]	PIN_U2
HEX4[3]	PIN_T4
HEX4[4]	PIN_R7
HEX4[5]	PIN_R6
HEX4[6]	PIN_T3
HEX5[0]	PIN_T2
HEX5[1]	PIN_P6
HEX5[2]	PIN_P7

HEX5[3]	PIN_T9
HEX5[4]	PIN_R5
HEX5[5]	PIN_R4
HEX5[6]	PIN_R3
HEX6[0]	PIN_R2
HEX6[1]	PIN_P4
HEX6[2]	PIN_P3
HEX6[3]	PIN_M2
HEX6[4]	PIN_M3
HEX6[5]	PIN_M5
HEX6[6]	PIN_M4
HEX7[0]	PIN_L3
HEX7[1]	PIN_L2
HEX7[2]	PIN_L9
HEX7[3]	PIN_L6
HEX7[4]	PIN_L7
HEX7[5]	PIN_P9
HEX7[6]	PIN_N9
KEY[0]	PIN_G26
KEY[1]	PIN_N23
KEY[2]	PIN_P23
KEY[3]	PIN_W26
LEDR[0]	PIN_AE23
LEDR[1]	PIN_AF23
LEDR[2]	PIN_AB21
LEDR[3]	PIN_AC22
LEDR[4]	PIN_AD22
LEDR[5]	PIN_AD23
LEDR[6]	PIN_AD21
LEDR[7]	PIN_AC21
LEDR[8]	PIN_AA14
LEDR[9]	PIN_Y13
LEDR[10]	PIN_AA13
LEDR[11]	PIN_AC14
LEDR[12]	PIN_AD15

LEDR[13]	PIN_AE15
LEDR[14]	PIN_AF13
LEDR[15]	PIN_AE13
LEDR[16]	PIN_AE12
LEDR[17]	PIN_AD12
LEDG[0]	PIN_AE22
LEDG[1]	PIN_AF22
LEDG[2]	PIN_W19
LEDG[3]	PIN_V18
LEDG[4]	PIN_U18
LEDG[5]	PIN_U17
LEDG[6]	PIN_AA20
LEDG[7]	PIN_Y18
LEDG[8]	PIN_Y12
CLOCK_27	PIN_D13
CLOCK_50	PIN_N2
EXT_CLOCK	PIN_P26
PS2_CLK	PIN_D26
PS2_DAT	PIN_C24
UART_RXD	PIN_C25
UART_TXD	PIN_B25
LCD_RW	PIN_K4
LCD_EN	PIN_K3
LCD_RS	PIN_K1
LCD_DATA[0]	PIN_J1
LCD_DATA[1]	PIN_J2
LCD_DATA[2]	PIN_H1
LCD_DATA[3]	PIN_H2
LCD_DATA[4]	PIN_J4
LCD_DATA[5]	PIN_J3
LCD_DATA[6]	PIN_H4
LCD_DATA[7]	PIN_H3
LCD_ON	PIN_L4
LCD_BLON	PIN_K2

SRAM_ADDR[0]	PIN_AE4
SRAM_ADDR[1]	PIN_AF4
SRAM_ADDR[2]	PIN_AC5
SRAM_ADDR[3]	PIN_AC6
SRAM_ADDR[4]	PIN_AD4
SRAM_ADDR[5]	PIN_AD5
SRAM_ADDR[6]	PIN_AE5
SRAM_ADDR[7]	PIN_AF5
SRAM_ADDR[8]	PIN_AD6
SRAM_ADDR[9]	PIN_AD7
SRAM_ADDR[10]	PIN_V10
SRAM_ADDR[11]	PIN_V9
SRAM_ADDR[12]	PIN_AC7
SRAM_ADDR[13]	PIN_W8
SRAM_ADDR[14]	PIN_W10
SRAM_ADDR[15]	PIN_Y10
SRAM_ADDR[16]	PIN_AB8
SRAM_ADDR[17]	PIN_AC8
SRAM_DQ[0]	PIN_AD8
SRAM_DQ[1]	PIN_AE6
SRAM_DQ[2]	PIN_AF6
SRAM_DQ[3]	PIN_AA9
SRAM_DQ[4]	PIN_AA10
SRAM_DQ[5]	PIN_AB10
SRAM_DQ[6]	PIN_AA11
SRAM_DQ[7]	PIN_Y11
SRAM_DQ[8]	PIN_AE7
SRAM_DQ[9]	PIN_AF7
SRAM_DQ[10]	PIN_AE8
SRAM_DQ[11]	PIN_AF8
SRAM_DQ[12]	PIN_W11
SRAM_DQ[13]	PIN_W12
SRAM_DQ[14]	PIN_AC9
SRAM_DQ[15]	PIN_AC10

SRAM_WE_N	PIN_AE10
SRAM_OE_N	PIN_AD10
SRAM_UB_N	PIN_AF9
SRAM_LB_N	PIN_AE9
SRAM_CE_N	PIN_AC11
OTG_ADDR[0]	PIN_K7
OTG_ADDR[1]	PIN_F2
OTG_CS_N	PIN_F1
OTG_RD_N	PIN_G2
OTG_WR_N	PIN_G1
OTG_RST_N	PIN_G5
OTG_DATA[0]	PIN_F4
OTG_DATA[1]	PIN_D2
OTG_DATA[2]	PIN_D1
OTG_DATA[3]	PIN_F7
OTG_DATA[4]	PIN_J5
OTG_DATA[5]	PIN_J8
OTG_DATA[6]	PIN_J7
OTG_DATA[7]	PIN_H6
OTG_DATA[8]	PIN_E2
OTG_DATA[9]	PIN_E1
OTG_DATA[10]	PIN_K6
OTG_DATA[11]	PIN_K5
OTG_DATA[12]	PIN_G4
OTG_DATA[13]	PIN_G3
OTG_DATA[14]	PIN_J6
OTG_DATA[15]	PIN_K8
OTG_INT0	PIN_B3
OTG_INT1	PIN_C3
OTG_DACK0_N	PIN_C2
OTG_DACK1_N	PIN_B2
OTG_DREQ0	PIN_F6
OTG_DREQ1	PIN_E5
OTG_FSPEED	PIN_F3

OTG_LSPEED	PIN_G6
TDI	PIN_B14
TCS	PIN_A14
TCK	PIN_D14
TDO	PIN_F14
TD_RESET	PIN_C4
VGA_R[0]	PIN_C8
VGA_R[1]	PIN_F10
VGA_R[2]	PIN_G10
VGA_R[3]	PIN_D9
VGA_R[4]	PIN_C9
VGA_R[5]	PIN_A8
VGA_R[6]	PIN_H11
VGA_R[7]	PIN_H12
VGA_R[8]	PIN_F11
VGA_R[9]	PIN_E10
VGA_G[0]	PIN_B9
VGA_G[1]	PIN_A9
VGA_G[2]	PIN_C10
VGA_G[3]	PIN_D10
VGA_G[4]	PIN_B10
VGA_G[5]	PIN_A10
VGA_G[6]	PIN_G11
VGA_G[7]	PIN_D11
VGA_G[8]	PIN_E12
VGA_G[9]	PIN_D12
VGA_B[0]	PIN_J13
VGA_B[1]	PIN_J14
VGA_B[2]	PIN_F12
VGA_B[3]	PIN_G12
VGA_B[4]	PIN_J10
VGA_B[5]	PIN_J11
VGA_B[6]	PIN_C11
VGA_B[7]	PIN_B11

VGA_B[8]	PIN_C12
VGA_B[9]	PIN_B12
VGA_CLK	PIN_B8
VGA_BLANK	PIN_D6
VGA_HS	PIN_A7
VGA_VS	PIN_D8
VGA_SYNC	PIN_B7
I2C_SCLK	PIN_A6
I2C_SDAT	PIN_B6
TD_DATA[0]	PIN_J9
TD_DATA[1]	PIN_E8
TD_DATA[2]	PIN_H8
TD_DATA[3]	PIN_H10
TD_DATA[4]	PIN_G9
TD_DATA[5]	PIN_F9
TD_DATA[6]	PIN_D7
TD_DATA[7]	PIN_C7
TD_HS	PIN_D5
TD_VS	PIN_K9
AUD_ADCLRCK	PIN_C5
AUD_ADCDAT	PIN_B5
AUD_DACLRCK	PIN_C6
AUD_DACDAT	PIN_A4
AUD_XCK	PIN_A5
AUD_BCLK	PIN_B4
ENET_DATA[0]	PIN_D17
ENET_DATA[1]	PIN_C17
ENET_DATA[2]	PIN_B18
ENET_DATA[3]	PIN_A18
ENET_DATA[4]	PIN_B17
ENET_DATA[5]	PIN_A17
ENET_DATA[6]	PIN_B16
ENET_DATA[7]	PIN_B15
ENET_DATA[8]	PIN_B20

ENET_DATA[9]	PIN_A20
ENET_DATA[10]	PIN_C19
ENET_DATA[11]	PIN_D19
ENET_DATA[12]	PIN_B19
ENET_DATA[13]	PIN_A19
ENET_DATA[14]	PIN_E18
ENET_DATA[15]	PIN_D18
ENET_CLK	PIN_B24
ENET_CMD	PIN_A21
ENET_CS_N	PIN_A23
ENET_INT	PIN_B21
ENET_RD_N	PIN_A22
ENET_WR_N	PIN_B22
ENET_RST_N	PIN_B23
IRDA_TXD	PIN_AE24
IRDA_RXD	PIN_AE25
SD_DAT	PIN_AD24
SD_DAT3	PIN_AC23
SD_CMD	PIN_Y21
SD_CLK	PIN_AD25
GPIO_0[0]	PIN_D25
GPIO_0[1]	PIN_J22
GPIO_0[2]	PIN_E26
GPIO_0[3]	PIN_E25
GPIO_0[4]	PIN_F24
GPIO_0[5]	PIN_F23
GPIO_0[6]	PIN_J21
GPIO_0[7]	PIN_J20
GPIO_0[8]	PIN_F25
GPIO_0[9]	PIN_F26
GPIO_0[10]	PIN_N18
GPIO_0[11]	PIN_P18
GPIO_0[12]	PIN_G23
GPIO_0[13]	PIN_G24

GPIO_0[14]	PIN_K22
GPIO_0[15]	PIN_G25
GPIO_0[16]	PIN_H23
GPIO_0[17]	PIN_H24
GPIO_0[18]	PIN_J23
GPIO_0[19]	PIN_J24
GPIO_0[20]	PIN_H25
GPIO_0[21]	PIN_H26
GPIO_0[22]	PIN_H19
GPIO_0[23]	PIN_K18
GPIO_0[24]	PIN_K19
GPIO_0[25]	PIN_K21
GPIO_0[26]	PIN_K23
GPIO_0[27]	PIN_K24
GPIO_0[28]	PIN_L21
GPIO_0[29]	PIN_L20
GPIO_0[30]	PIN_J25
GPIO_0[31]	PIN_J26
GPIO_0[32]	PIN_L23
GPIO_0[33]	PIN_L24
GPIO_0[34]	PIN_L25
GPIO_0[35]	PIN_L19
GPIO_1[0]	PIN_K25
GPIO_1[1]	PIN_K26
GPIO_1[2]	PIN_M22
GPIO_1[3]	PIN_M23
GPIO_1[4]	PIN_M19
GPIO_1[5]	PIN_M20
GPIO_1[6]	PIN_N20
GPIO_1[7]	PIN_M21
GPIO_1[8]	PIN_M24
GPIO_1[9]	PIN_M25
GPIO_1[10]	PIN_N24
GPIO_1[11]	PIN_P24

GPIO_1[12]	PIN_R25
GPIO_1[13]	PIN_R24
GPIO_1[14]	PIN_R20
GPIO_1[15]	PIN_T22
GPIO_1[20]	PIN_T21
GPIO_1[21]	PIN_T20
GPIO_1[22]	PIN_U26
GPIO_1[23]	PIN_U25
GPIO_1[24]	PIN_U23
GPIO_1[25]	PIN_U24
GPIO_1[26]	PIN_R19
GPIO_1[27]	PIN_T19
GPIO_1[28]	PIN_U20
GPIO_1[29]	PIN_U21
GPIO_1[30]	PIN_V26
GPIO_1[31]	PIN_V25
GPIO_1[32]	PIN_V24
GPIO_1[33]	PIN_V23
GPIO_1[34]	PIN_W25
GPIO_1[35]	PIN_W23

Tabla 3.2 Distribución de pines

2.6.4 El planeador de pines de Quartus II

Una vez que el circuito diseñado haya superado con éxito la compilación, la síntesis y la simulación, el paso siguiente es la asignación de los pines del FPG a las entradas y salidas del circuito diseñado.

La compilación permite detectar errores en la sintaxis del programa. La síntesis permite crear el circuito diseñado en el nivel más bajo de abstracción. La simulación permite verificar si el circuito trabaja o funciona como debe funcionar; es decir, por ejemplo, si es un sumador, debe sumar.

La asignación de los pines a las entradas y salidas del circuito se debe realizar sobre la base de los siguientes pasos:

En la opción *assignment* del menú principal de Quartus II, se hace clic sobre *Pin Planner* y aparece la ventana que se indica en la figura 2.57.

Figura 2.57. Ventana para la planeación de pines del FPGA

En la parte inferior de esta ventana, se hace clic sobre la opción *location* y aparece una ventana adicional que se muestra en la figura 2.58.

Figura 2.58. Ventana con el planeador de pines desplegada

En la ventana de la figura 2.59, se muestra la porción ampliada de la ventana desplegada de la figura 2.58.

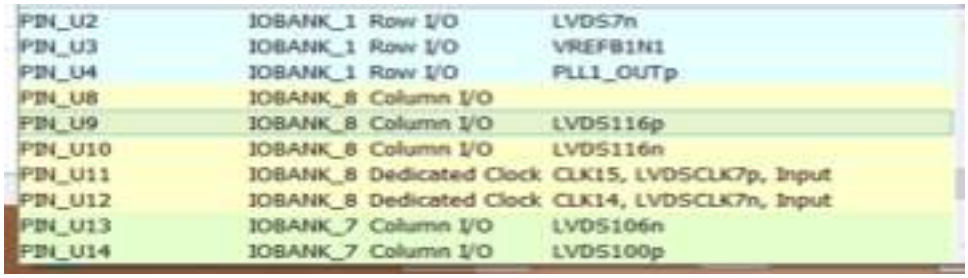


Figura 2.59. Ventana desplegada ampliada

En la ventana 3.47, se seleccionan los pines que se van asignar a cada entrada y salida del circuito diseñado. Una vez terminada esta asignación, se compila de nuevo desde la ventana principal de Quartus II y se procede a programar el FPGA.

2.6.5 Gráfico del circuito diseñado

Antes de programar el FPGA, si se quiere obtener el gráfico del circuito diseñado se procede de la siguiente manera: en la ventana principal de Quartus II se hace clic sobre la opción *Tool*, se selecciona *Netlist* y *RTL Viewer*, y aparece la ventana de la figura 2.60.

Figura 2.60. Gráfico del circuito diseñado

2.6.6 Programación del FPGA

Para programar el FPGA, se procede de la manera siguiente:

En el menú de la ventana principal de Quartus II se hace clic sobre *Tool/programmer/*, y aparece la ventana de la figura 2.61. Si en la pestaña que está al lado de la pestaña *Hardware Setup* aparece *no Hardware*, significa que el entrenador

DE2 no está conectado o la computadora no ha reconocido al USB-Blaster. Si este es el caso, se debe hacer clic sobre la pestaña *Hardware Setup* y aparece la ventana que muestra la figura 2.61. En esa ventana se hace clic sobre la pestaña *no hardware* y aparece la opción USB-Blaster, se hace clic sobre esta y está listo.

Para iniciar la programación, se hace clic sobre la pestaña *Start* de la ventana que se muestra en la figura 2.61.

Cuando la pestaña *Progres* de la misma figura ha llegado al 100 % el FPGA está programado.

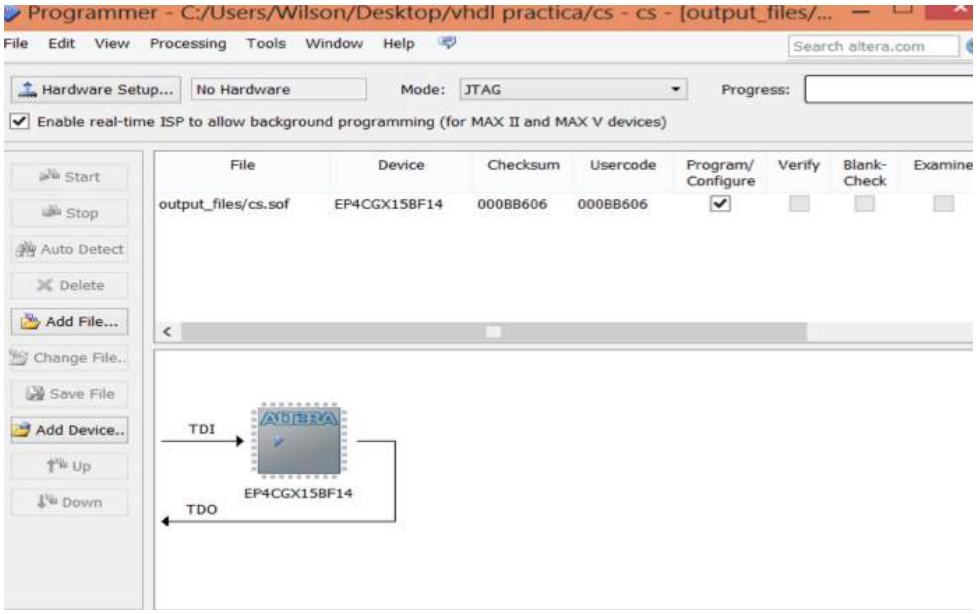


Figura 2.61. Ventana de programación del FPGA

2.6.7 Ejercicios propuestos

1. Explique dos formas diferentes para crear un nuevo proyecto en Quartus II.

2. Cuando se está creando un nuevo proyecto, es necesario especificar con claridad en nombre de la entidad de más alto nivel. Explique cuál debe ser por obligación ese nombre.

3. Para abrir un proyecto guardado, se deben abrir dos archivos. ¿Cuáles?

4. Cuando se está creando un nuevo proyecto con Quartus II, el asistente para la creación de nuevo proyecto le permite realizar cinco acciones. ¿Cuáles son esas acciones?

5. ¿El nombre del proyecto y el nombre de la entidad de más alto nivel deben ser los mismos? Explique.

6. Explique cuál es proceso para añadir algunos archivos que su proyecto requiera.

7. ¿Qué significa seleccionar la familia y el dispositivo destino?

8. ¿Quartus II puede incluir alguna otra herramienta EDA en un proyecto que se esté creando?

9. Qué es una captura esquemática dentro de Quartus II?

10. ¿Cuál es el proceso mediante el cual se crea una captura esquemática en Quartus II?

11. ¿Por qué hay que compilar un proyecto en Quartus II?

12. ¿Cuál es la función de la opción Analysis & Sinthesis?

13. En el proceso de compilación, Quartus II le indica los errores y los *Warnings*. Explique cada uno.

14. Si la compilación de un proyecto fue exitosa, entonces el proyecto está correctamente elaborado y el circuito diseñado debe trabajar adecuadamente. ¿Por qué?

15. ¿Para qué se utiliza VHDL File?

16. ¿Cómo se compila un proyecto con Quartus II?

17. ¿Cómo se simula un proyecto con Quartus II?

18. ¿Qué es una simulación temporal?

19. ¿Qué es una simulación funcional?

20. ¿Cuál es la diferencia entre una simulación temporal y una simulación funcional?

21. ¿Cuál es la utilidad del RTL Viewer?

22. Explique cada una de las cuatro ventanas que contiene la ventana principal de Quartus II.

23. ¿Qué información contiene la opción: *insert template* y cuál es su utilidad?

24. ¿Cómo se configuran los pines del FPGA que están disponibles en el entrenador DE2 de Altera?

25. Explique el proceso mediante el cual se programa el FPGA del entrenador DE2 de Altera.

26. Explique cómo se instala el USB-Blaster.

27. Si en la ventana *Programmer* no está visible el USB-Blaster, ¿qué procedimiento debe realizarse para que sea visible?

28. En la ventana *Programmer*, ¿cuál es la utilidad de *Add File*?

29. En la ventana *Programmer*, ¿cuál es la utilidad de *Add Device*?

BIBLIOGRAFÍA

Altera. (2015). DE2 Developed and Education Board. Recuperado de: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyc2/cyc2_cii51001.pdf. Recuperado de: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/cyc2/cyc2_cii51002.pdf.

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

ÍNDICE EN ORDEN ALFABÉTICO

La ciencia en la que se fundamenta los sistemas digitales, en lo relativo a sus conceptos y fundamentos, no ha sufrido un cambio radical todavía, sin embargo, el diseño y aplicación práctica de esta ciencia (la implementación) ha cambiado sustancialmente en las últimas décadas con la aparición de los lenguajes de descripción de hardware (HDL) y herramientas CAD.

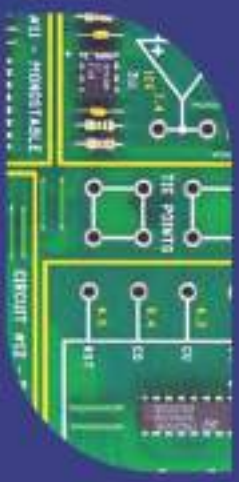
Este libro expone los conceptos fundamentales de los diagramas de estado desde una perspectiva clásica, pues su fundamento científico no ha cambiado. El diseño mediante HDL y herramientas CAD es tratado en el tercer libro de esta obra.

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 su contenido de una forma que el lector pueda desarrollar habilidad intuitiva para entender y aplicar los conceptos fundamentales, la estructura, el análisis y diseño de máquinas secuenciales sincrónicas. Se exponen ejemplos que permiten reforzar los conocimientos que el lector va adquiriendo.

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 magister 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 magister 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: 978-9942-35-648-2