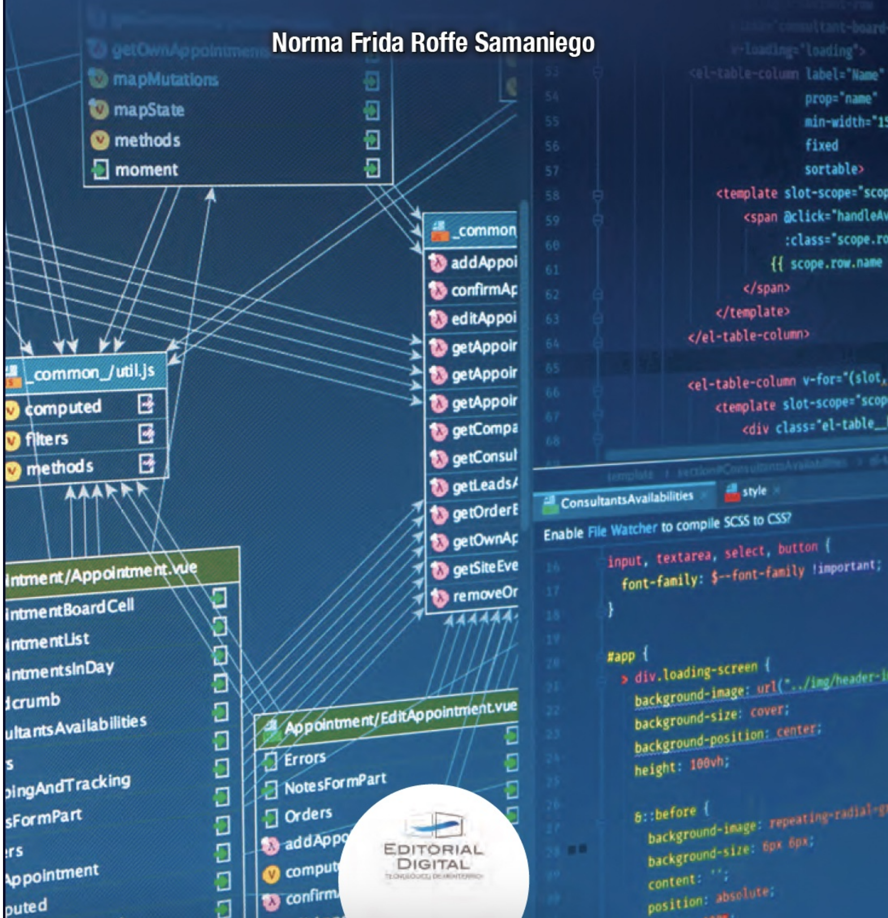
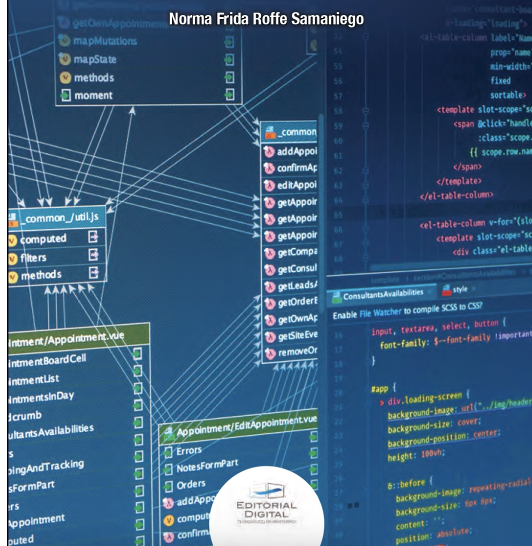
Primera edición

Sistemas digitales a través de diseños esquemáticos y VHDL



Norma Frida Roffe Samaniego

©Editorial Digital Tecnológico de Monterrey



Primera edición
De venta en: Amazon Kindle, Apple Books, Google Books y Amazon.
Fragmento editado, diseñado, publicado y distribuido por el Instituto Tecnológico y de Estudios Superiores de Monterrey.
Se prohíbe la reproducción total o parcial de esta obra por cualquier medio sin previo y expreso consentimiento por escrito del Instituto Tecnológico y de Estudios Superiores de Monterrey.
Ave. Eugenio Garza Sada 2501 Sur Col. Tecnológico C.P. 64849 |
Monterrey, Nuevo León | México.

©Editorial Digital Tecnológico de Monterrey

Página

Acerca de la autora 11

Introducción 13

Capítulo 1. Descripción de un circuito en VHDL 17

1.1 Características generales de un circuito descrito en VHDL 20

1.2 Descripción de un circuito en VHDL 22

1.3 Simulación de un circuito 28

1.4 Expresiones booleanas en VHDL 35

1.5 Más información sobre la definición de un circuito: entidad 43

1.6 Nombres 44

1.7 Tipos de datos 45

1.8 Arquitectura: niveles de descripción 47

1.9 Sintaxis de VHDL 48

1.10 Descripción de circuitos combinacionales con mayor número de elementos 52

Actividad integradora del capítulo 1 62

Conclusión del capítulo 1 68

Capítulo 2. Niveles de descripción de un circuito 71

2.1 Definiciones del mux 4:1 en diferentes niveles 73

2.2 Circuitos combinacionales diseñados con procesos 93

2.3 Verificación del if 98

2.4 Uso del case 100

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 2 102

Conclusión del capítulo 2 114

Capítulo 3. Circuitos para operaciones aritméticas y lógicas 117

3.1 Diseño de un sumador de dos números de 4 bits 118

3.2 Diseño del sumador en VHDL en el nivel de ecuaciones 126

3.3 Buses (vectores) de datos 127

3.4 Diseño del sumador en VHDL en el nivel de comportamiento 131

3.5 La resta 133

3.6 Comparador 139

3.7 Ejemplos 147

Actividad integradora del capítulo 3 158

Conclusión del capítulo 3 166

Capítulo 4. Aritmética BCD 169

4.1 Diseño de un sumador BCD 173

4.2 Diseño de un restador BCD 183

4.3 Ejemplos 186

Actividad integradora del capítulo 4 192

Conclusión del capítulo 4 198

©Editorial Digital Tecnológico de Monterrey

Capítulo 5. Circuitos secuenciales básicos: latches, flip flops y su modelación en VHDL 201

5.1 Introducción a los circuitos secuenciales 202

5.2 Estructuras en VHDL para el modelado de circuitos secuenciales 203

5.3 Circuitos almacenadores (*latches*) 211

5.4 Flip flops 223

Actividad integradora del capítulo 5 236

Conclusión del capítulo 5 240

Capítulo 6. Registros 243

6.1 ¿Qué es un registro? 244

6.2 Registros y su operación 247

6.3 Arquitectura interna de una muestra de registros 252

6.4 Modelación de registros por sus funciones 268

6.5 Ejemplos de modelación de registros por sus funciones .. 275

6.6 Definición de un circuito que contiene más de un registro 280

6.7 Ejemplos de un circuito que contiene más de un registro. .. 284

6.8 Modelación incorrecta 287

Actividad integradora del capítulo 6 302

Conclusión del capítulo 6 310

©Editorial Digital Tecnológico de Monterrey

Capítulo 7. Circuitos monoestables y astables 313

7.1 Señales de reloj 315

7.2 Eliminación de ruido eléctrico (rebotes) 323

Actividad integradora del capítulo 7 330

Conclusión del capítulo 7 332

Capítulo 8. Diseño de circuitos secuenciales basados en registros 335

8.1 Conexión maestro-esclavo de dos registros 337

8.2 Aplicación del circuito one shot 347

8.3 Análisis de un circuito 350

8.4 Uso de la función de rotación 356

8.5 Secuencia maestro-esclavo 359

8.6 División de un número entero entre un número que es una potencia de 2 361

8.7 Entrada serial salida decodificada 366

8.8 Registro de arquitectura especial 369

8.9 Bomba de gasolina 374

8.10 Juego de basketball para niños 378

8.11 Diseño de una alarma 381

8.12 Horno de microondas 385

8.13 Captura de botones 389

8.14 Marcador de futbol 392

8.15 Calculadora 400

Actividad integradora del capítulo 8 408

Conclusión del capítulo 8 412

©Editorial Digital Tecnológico de Monterrey

Capítulo 9. Diseño de autómatas 415

9.1 Diseño de contadores 417

9.2 Diseño de máquinas de estados 433

Actividad integradora del capítulo 9 528

Conclusión del capítulo 9 538

Capítulo 10. Diseño de circuitos basados en unidades de control 541

10.1 Restador serial 544

10.2 Otro ejemplo de diseño de un sistema digital basado en una unidad de control 560

10.3 Ejercicio 568

10.4 Juego de TV 569

10.5 Conversión de un dato serial asincrónico a una salida paralela 574

10.6 Conversión de un dato serial sincrónico a una salida paralela 580

10.7 Conversión de un dato paralelo a uno serial 583

10.8 Banda transportadora I 587

10.9 Horno de microondas 592

10.10 Mensaje luminoso de una columna de LEDs 594

10.11 Máquina despachadora de café 600

10.12 Ensamble de juguetes 608

Actividad integradora del capítulo 10 612

Conclusión del capítulo 10 628

©Editorial Digital Tecnológico de Monterrey

Capítulo 11. Memorias 631
11.1 Introducción 632
11.2 Memorias ROM 633
11.3 Ejercicio 638
11.4 Memorias RAM 638
11.5 Más sobre memorias 642
11.6 Definición de una memoria 644
11.7 Memoria en el FPGA 646
11.8 SRAM 649
Actividad integradora del capítulo 11 656
Conclusión del capítulo 11 660
Capítulo 12. Periféricos 663
12.1 VGA 664
12.2 Teclado 669
12.3 Ratón 675
Actividad integradora del capítulo 12 686
Conclusión del capítulo 12 688
Capítulo 13. Aplicaciones aritméticas 691
13.1 Tipos de números 693
13.2 Números de punto fijo 703
13.3 Números de punto flotante 705
13.4 Operaciones aritméticas: multiplicación y división 709

©Editorial Digital Tecnológico de Monterrey

13.5 División 730

13.6 Operaciones de números en formato de punto flotante ... 741

Actividad integradora del capítulo 13 744

Conclusión del capítulo 13 750

Glosario 753

Bibliografía 771

Créditos 774

Aviso legal 775

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Acerca de la autora

Norma Frida Roffe Samaniego

Profesora del Tecnológico de Monterrey, Campus Monterrey. Es ingeniera en Sistemas Electrónicos por el Tecnológico de Monterrey, cuenta con una Maestría en Sistemas de Información y un Doctorado en Informática, también del Tecnológico de Monterrey. Gran parte de su carrera profesional la realizó en el Tecnológico y sus cargos más destacados fueron: directora de la carrera de Ingeniería en Sistemas Electrónicos, directora de Efectividad Institucional y Calidad Académica del Centro de Estudios Estratégicos y directora de Proyectos Académicos Institucionales de la Vicerrectoría Académica. Tiene dos libros publicados: *Modelo de planeación para la educación del siglo XXI: recursos y competencias y Guía para el diseño de compiladores*, el cual está publicado en Internet, además de dos libros literarios que están en proceso de publicación.

De igual manera, tiene una patente otorgada: "sistema de control para el ambiente del usuario", una solicitada y ocho publicadas, cuatro de ellas internacionales.

Cuenta con un método de creatividad y otro de planeación estratégica registrados. Programó el primer simulador de HDL que se utilizó en el posgrado de Ingeniería Eléctrica en el Tecnológico de Monterrey. Ha sido titular de la materia de Diseño de Sistemas Digitales durante diez años y ha impartido más de treinta conferencias en México, Estados Unidos y Europa.

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Introducción

Este libro se centra en el diseño de circuitos lógicos; no es una introducción, sino una lectura de nivel intermedio. Su contenido está dirigido a quienes dominan el álgebra booleana y ya han diseñado y construido circuitos combinacionales; de igual forma, se espera que los lectores tengan conocimientos de sistemas numéricos y de organización computacional.

El mayor énfasis de diseño se enfocará a sistemas digitales de registro, monitoreo y control, así como de unidades centrales de procesamiento.

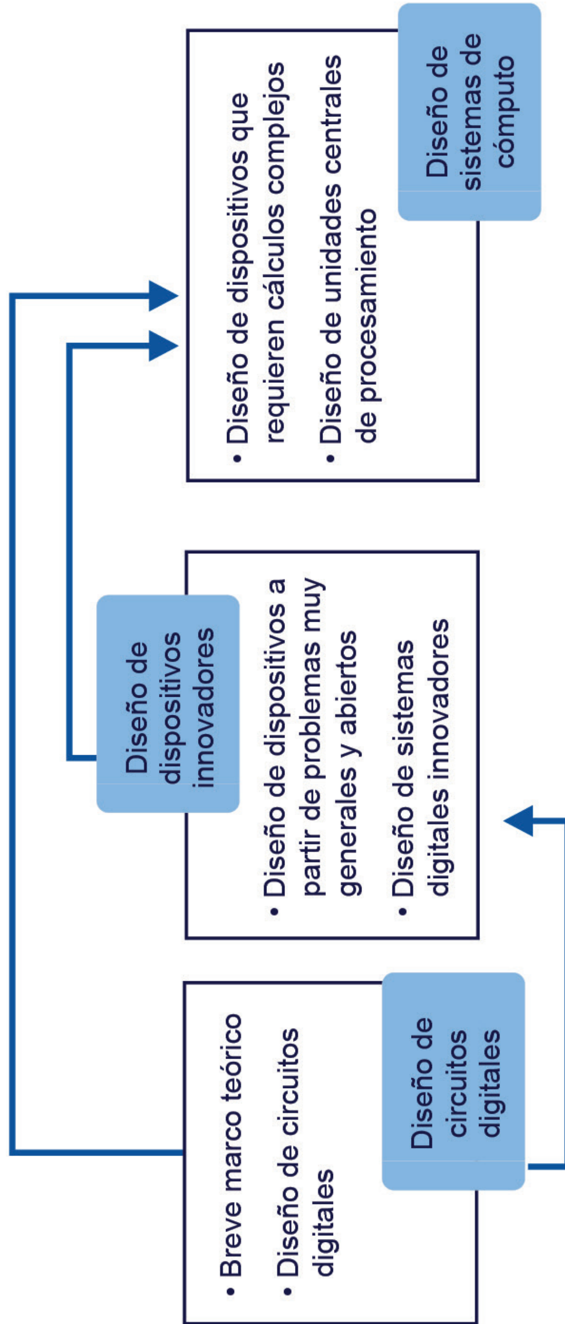
Tiene un enfoque práctico y está basado en la solución de problemas. Se plantea el diseño de componentes básicos para luego interconectarlos y crear sistemas más complejos, por lo que el método de aprendizaje implícito es inductivo e incremental.

Para el diseño de circuitos digitales se seguirán dos enfoques: el diseño basado en circuitos esquemáticos utilizando componentes TTL comerciales y el uso de una poderosa herramienta de diseño, un lenguaje descriptor de circuitos: VHDL (*VHDL = VHSIC Hardware Description Language, VHSIC = Very High Speed Integrated Circuit*), ya que para la formación de un ingeniero especializado en electrónica digital competente y competitivo internacionalmente, es necesario que se conozcan las herramientas de diseño de sistemas electrónicos más poderosas y modernas. La mejor de estas herramientas —que es la que aprenderá con este libro— es VHDL, la cual permite el diseño, simulación e implementación de sistemas digitales. La solución de los ejercicios y proyectos propuestos, y la flexibilidad de diseño que provee el VHDL contribuirán a que cada lector forme o refuerce un estilo de diseño propio.

©Editorial Digital Tecnológico de Monterrey

Actualmente, el desarrollo de un país tiene una gran relación con la producción y uso de productos de alta tecnología, ya que su elaboración y uso son de alto valor agregado. El compromiso inminente que tiene todo ingeniero egresado de una institución de educación superior en contribuir al desarrollo nacional, conlleva a impactar en los procesos productivos para que sean más eficientes. Los sistemas basados en electrónica digital juegan un papel clave en concretar este compromiso. El aspecto medular del libro es la combinación entre el bosquejo de circuitos esquemáticos, el aprendizaje de VHDL y el diseño de dispositivos de aplicación práctica.

©Editorial Digital Tecnológico de Monterrey



©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 1. Descripción de un circuito en VHDL

Descripción de un circuito en VHDL

Características generales de un circuito descrito en VHDL	Nombres
Descripción de un circuito en VHDL	Tipos de datos
Simulación de un circuito	Arquitectura: niveles de descripción
Expresiones booleanas en VHDL	Sintaxis de VHDL
Más información sobre la definición de un circuito: entidad	Descripción de circuitos combinacionales con mayor número de elementos

VHDL es la abreviación de **VHSIC** (*Very High Speed Integrated Circuit*) y **HDL** son las siglas de *Hardware Description Language*.

VHDL es un lenguaje —con una sintaxis similar a la de los lenguajes de programación, pero con una esencia sustancialmente diferente— con el que se define un circuito a partir de definir sus elementos y su interconexión. VHDL es una herramienta **EDA** (*Electronic Design Automation*) utilizada en el diseño de sistemas electrónicos que se contendrá en un circuito integrado. Al igual que el lenguaje de programación **ADA** (en memoria de Ada Byron, hija del poeta Lord Byron, matemática y compiladora del trabajo de

©Editorial Digital Tecnológico de Monterrey

Charles Babbage), su diseño VHDL inició en 1981. El primer borrador se tuvo listo en agosto de 1985 y fue diseñado por *Intermetrics*, *IBM* y *Texas Instrument*.

El desarrollo de VHDL fue patrocinado por la milicia estadounidense para estandarizar la especificación de sistemas digitales. Se requería documentar todos los **ASIC** (*Application Specific Integrated Circuit*) que se fabricaran para el Departamento de Defensa de los Estados Unidos y por esta razón se diseñó con una sintaxis similar a la de **ADA**.

En diciembre de 1987 fue aprobado como estándar del Institute of Electrical and Electronics Engineers (IEEE) y posteriormente, en 1993, fue revisado y registrado como norma IEEE Std1076−1993.

Otros lenguajes descriptores de *hardware* son:

• **ABEL** (*Advanced Boolean Expression Language*), lenguaje inicialmente creado en 1983 para programar PLD (Programmable Logic Device) por Data I/O Corporation, en Redmond, Washington.

• Verilog (*Verify logic*), creado en 1985 en Automated Integrated Design Systems Cadence, la cual tiene ahora todos los derechos sobre los simuladores lógicos de Verilog.

Los sistemas de desarrollo que cuentan con VHDL como herramienta de diseño, permiten la descripción (documentación), simulación e implementación de circuitos digitales.

Dicha implementación se realiza en dos tipos de circuitos configurables: **CPLD** (*Complex Programable Logic Device*) y **FPGA** (*Field Programable Gate Array*). Ambos tipos de circuitos permiten el ruteo (configuración) de circuitos secuenciales y son de gran densidad, ya que cuentan con miles de compuertas lógicas. Los modelos existentes tienen diferentes cantidades de compuertas y soportan distintas frecuencias de reloj. Su arquitectura básica cuenta con redes AND-OR para configurar circuitos combinacionales, y *flip flops* (típicamente D) para configurar registros. La arquitectura de un FPGA y de un CPLD se muestra en el apéndice I.

©Editorial Digital Tecnológico de Monterrey

También es posible la fabricación de un circuito a la medida, un ASIC, a partir de la especificación del circuito en VHDL, la cual cada vez tiene menos desventajas con respecto a la fabricación de un ASIC, ya que actualmente cada vez es mayor la velocidad de un circuito construido sobre un FPGA y por supuesto, el costo de fabricación es muy bajo, ya que el precio individual de un circuito es de aproximadamente 100 pesos y existen sistemas de desarrollo de uso abierto.

©Editorial Digital Tecnológico de Monterrey

1.1 Características generales de un circuito descrito en VHDL

El primer propósito de VHDL fue crear un estándar para la descripción de circuitos. El diseño esquemático de un circuito con múltiples componentes necesita mucho espacio en papel; son difíciles de analizar, no tienen principio ni fin. Si se precisa equipo computacional para elaborar el diseño esquemático, son necesarios monitores amplios para alcanzar a visualizar una parte representativa del circuito; además, se requieren impresoras especiales (*plotters*) porque los diseños no caben en una hoja simple, de otro modo habría que seccionarlos, lo cual los volvería menos comprensibles.

VHDL es una alternativa para describir circuitos utilizando solo texto, sin gráficas, que ciertamente requiere menos recursos para escribirse, desplegarse y almacenarse. Por otra parte, este lenguaje permite describir los componentes digitales de diferentes formas, de acuerdo al circuito o al gusto del diseñador. Por ejemplo, cuando se dibuja el diseño esquemático de un circuito es posible hacerlo con diferentes niveles de integración, es decir, podemos dibujar compuertas (o más bajo nivel, transistores) o bien circuitos conocidos (sumadores, decodificadores, registros, etc.) señalándolos como cajas; en VHDL ocurre algo similar, es posible diseñar un circuito por sus compuertas lógicas, o bien describirlo por su comportamiento. También es posible definir un componente para luego conectarlo con otros. Al dibujar diseños esquemáticos no se cuenta con la opción de trazar el comportamiento de un circuito; esta opción es exclusiva de los lenguajes descriptores de *hardware*.

Cualquier diseño de VHDL está compuesto por dos partes esenciales:

1. La entidad, bloque que define el puerto (o interfaz) del circuito, es decir, las entradas y las salidas con las que se comunica al exterior.

2. La arquitectura, donde se describe el diseño del circuito.

©Editorial Digital Tecnológico de Monterrey

Para mostrar cómo se describe un circuito, partiremos del ejemplo que se presenta en la sección anterior.

Actividad de repaso del tema 1.1

1. Principales fabricantes de FPGAs.

2. Principales características de la arquitectura de un FPGA.

3. ¿Qué es un ASIC?

4. ¿Cuál es la diferencia entre un ASIC y un FPGA?

5. ¿Cuál es la diferencia entre un FPGA y un microcontrolador o microprocesador?

6. Describa en forma general la arquitectura de un CLB del Spartan III de Xilinix.

7. ¿Cuál es el costo de un sistema profesional de desarrollo de Xilinix?

8. ¿Cuál es, actualmente, el FPGA con mayor capacidad en el mercado y cuál es su costo?

©Editorial Digital Tecnológico de Monterrey

1.2 Descripción de un circuito en VHDL

Posteriormente se formalizará la sintaxis. Se iniciará con circuitos combinacionales. El primer ejemplo trata de una votación electrónica. Suponga que se cuenta con tres interruptores para realizar una votación. Para votar a favor se colocaría el interruptor en posición de "on", es decir en uno lógico. La salida del circuito se conectaría a un indicador que se iluminaría en caso de ocurrir mayoría de votos (dos o tres votos a favor).

Los nombres de los tres interruptores son: **a**, **b** y **c**, la salida es **d**. La expresión mínima, en álgebra booleana para d ocurre cuando se presentan dos o tres unos en las entradas (si ocurren tres unos, los tres términos serán uno), esto es: **d = ab + ac + bc**. El diseño esquemático basado en compuertas lógicas es el que se aprecia en la figura 1.1.

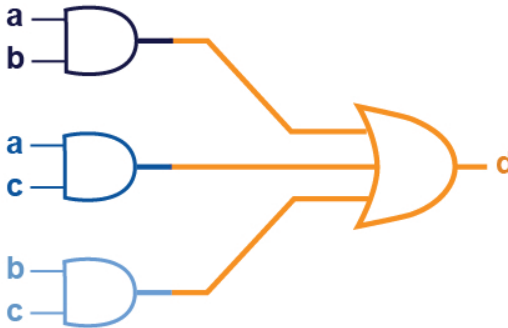


Figura 1.1 Circuito prueba

©Editorial Digital Tecnológico de Monterrey

La descripción en VHDL es:

```
entity comb1 is
    Port  ( a : in std_logic;
            b : in std_logic;
            c : in std_logic;
            d : out std_logic);
end comb1;

architecture ecuacion of comb1 is
begin
d <= (a and b) or (a and c) or (b and c);
end ecuacion;
```

A continuación, se abordará cada detalle de esta descripción.

A un circuito se le asocia un nombre, en este caso, el nombre asociado es **comb1**. Como se mencionó, la descripción consta de dos partes: entidad y arquitectura (figura 1.2).

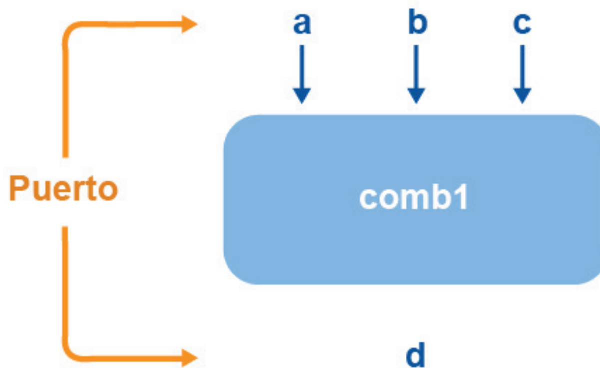


Figura 1.2 Descripción de un circuito

©Editorial Digital Tecnológico de Monterrey

La entidad define al puerto del circuito, las señales que se conectarán como entradas al circuito y las que se generarán de salida, es decir, la interfaz con el exterior del circuito. Como habrá notado, el tipo con el que se han definido las señales es **std_logic** (*standard logic*). El tipo std_logic indica que una señal puede tomar valores lógicos, como 1 o 0. Se dará mayor detalle de este tipo más adelante.

Por otra parte, la arquitectura describe al circuito, sus elementos e interconexiones. La palabra **ecuacion** (sin acento, porque VHDL no acepta acentos) en *architecture ecuacion of comb1* is es parte de la documentación del circuito; esa palabra solo tiene significado para el diseñador y para quien observe el código, pero no para el compilador.

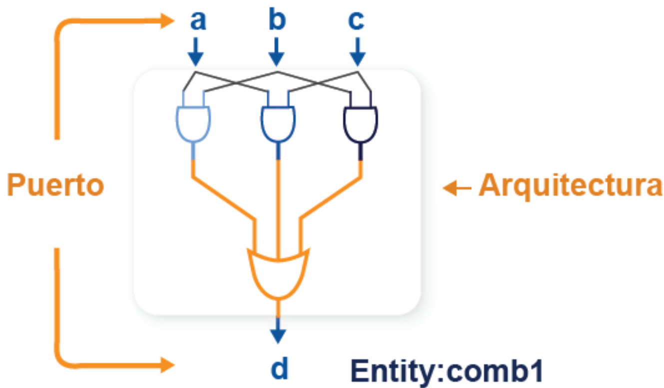


Figura 1.3 Arquitectura del circuito

En VHDL hay varias alternativas para describir un mismo circuito: por su expresión booleana, por sus elementos lógicos (pueden ser sus compuertas) o por su comportamiento (por ejemplo, su tabla de verdad).

La manera más sencilla para expresar la arquitectura de un circuito combinacional es por medio de su expresión lógica. Además, esta manera es la más conocida para usted, dados sus conocimientos en álgebra booleana. Así se hace en este ejemplo.

©Editorial Digital Tecnológico de Monterrey

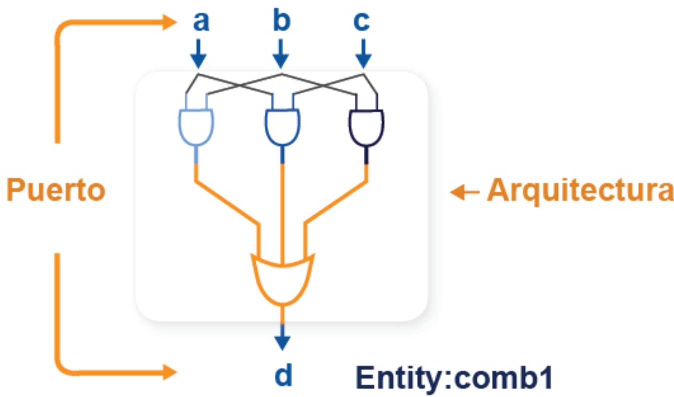


Figura 1.4 Descripción de la arquitectura de un circuito

1.2.1 Descripción por secciones

Existen más alternativas para describir este circuito; por ejemplo, por partes, en vez de utilizar solo una expresión booleana se usan varias. En la figura 1.5 se muestran alternativas para la descripción por secciones.

Es posible etiquetar las salidas intermedias de algunas o de todas las compuertas y fraccionar el circuito. Note que las salidas intermedias son entradas de otra sección del circuito. Estas señales intermedias no son parte del puerto porque no tienen contacto con el exterior del circuito.

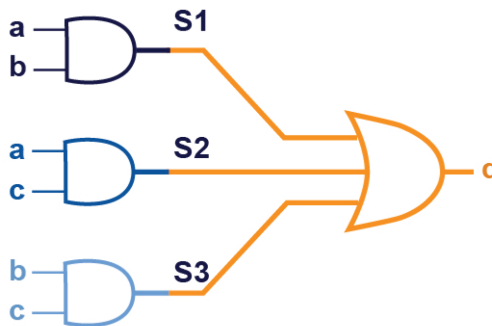


Figura 1.5 Descripción por secciones

©Editorial Digital Tecnológico de Monterrey

```
entity comb1 is
Port (a : in std_logic;
b : in std_logic;
c : in std_logic;
d : out std_logic);
end comb1;

architecture ecuaciones of comb1 is
signal S1, S2, S3 : std_logic;
begin
d <= S1 or S2 or S3;
S1 <= a and b;
S2 <= b and c;
S3 <= a and c;
end ecuaciones;
```

La definición de estas señales se efectúa en la sección **architecture**. Esta sección declarativa inicia con la palabra **signal** y se debe colocar antes de **begin**. Cabe señalar que cada lista de señales intermedias que correspondan a un tipo debe iniciar con la palabra **signal**.

Tanto **a**, **b**, **c**, **d**, como **S1**, **S2** y **S3** son señales, pero las primeras son señales que están conectadas al puerto del circuito y las segundas son señales internas (auxiliares).

El orden en que se describen las señales intermedias **S1**, **S2** y **S3**, así como la salida **d** no tiene importancia. Cualquier orden es correcto, finalmente en los circuitos esquemáticos no hay un inicio establecido por alguna convención. Es ar*bit*rario si el circuito se define desde sus entradas hasta sus salidas o si se comienza por la parte central. El diseñador puede definir el orden en el que desea describir su circuito.

La siguiente sería otra posibilidad para la descripción del circuito:

©Editorial Digital Tecnológico de Monterrey

```
entity comb1 is
Port (a : in std_logic;
      b : in std_logic;
      c : in std_logic;
      d : out std_logic);
end comb1;

architecture ecuaciones of comb1 is
signal S1, S2, S3 : std_logic;
begin
S1 <= a and b;
S2 <= b and c;
d <=S1 or S2 or S3;
S3 <= a and c;
end ecuaciones;
```

Otra posible descripción del circuito es:

```
S1 <= a and b;
S2 <= b and c;
S3 <= a and c;
d <=S1 OR S2 OR S3;
```

Se proporcionará más información sobre los operadores booleanos y la construcción de expresiones booleanas en el tema “Expresiones booleanas en VHDL”.

©Editorial Digital Tecnológico de Monterrey

1.3 Simulación de un circuito

Al tener un circuito descrito en VHDL, cualquier programador capacitado en desarrollar un compilador puede visualizar la factibilidad de programar un simulador que muestre el comportamiento de las salidas de un circuito a través del tiempo, a partir de construir estructuras de datos consistentes de tablas que evalúen las salidas cada vez que ocurren cambios en las entradas.

Antes de que existieran simuladores disponibles para estudiantes universitarios, desarrollé un simulador como parte de un curso de posgrado. Luego surgieron opciones comerciales y actualmente hay una gran variedad de sistemas de desarrollo de uso abierto que permiten simular circuitos.

Para comprender la simulación, el primer paso es examinar la respuesta de un circuito. Para ello, analicemos el retraso de una compuerta.

El retraso de una compuerta es el tiempo que se requiere para que su salida sea estable y válida (figura 1.6). Este tiempo está en el orden de picosegundos. Como es un lapso muy pequeño, se le denomina delta de tiempo.

Si en tiempo = 0 las entradas al circuito tienen un valor estable, las salidas **S1**, **S2** y **S3** tendrán salidas estables y válidas a la vez, esto será en tiempo = delta, y la salida d tendrá un valor estable en tiempo = 2 delta.

Las salidas **S1**, **S2** y **S3** son estables porque su operación eléctrica ocurre al mismo tiempo. Como no es posible que el código ejecute código en forma concurrente, el simulador se vale de estructuras de datos para emular el paralelismo de las compuertas.

©Editorial Digital Tecnológico de Monterrey

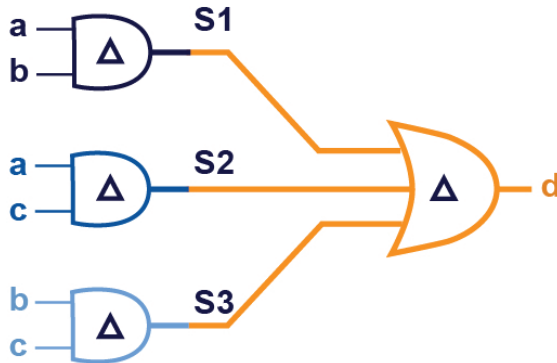


Figura 1.6 Retraso de un circuito

El sistema de desarrollo crea una tabla que registra en qué tiempo estará estable cada salida, y cuando una salida es entrada de otra parte del circuito, se hace el cálculo del retraso. La salida de cada compuerta o de cada sección del circuito se calcula de acuerdo al tiempo en que es válida su entrada, más el tiempo de retraso de esa compuerta o sección del circuito.

Por otra parte, cada vez que cambia el valor de una señal de la cual depende una señal de salida, se calcula de nuevo el valor de la salida; esto ocurre hasta que el valor de todas las salidas quede estable.

Por ejemplo, para la descripción:

```
architecture ecuacion of comb1 is
begin
d <= (a and b) or (a and c) or (b and c);
end ecuacion;
```

Suponga que en tiempo = 0 se asocian un 0 a **a**, 1 a **b** y 1 a **c**. Con esta entrada, la salida d quedaría estable después del tiempo de retraso (delta) del circuito.

©Editorial Digital Tecnológico de Monterrey

t = 0	t = 0 + Δ
a = 0	a = 0
b = 1	b = 1
c = 1	c = 1
d = U	d = 1

Tabla 1.1 Simulación del circuito

El valor **U** significa *undefined*, es decir, indefinido. Indefinido significa que puede ser 0 o 1, no que esté desconectado o sin valor. Cabe recalcar que una vez que una compuerta esté energizada tiene una salida, pero no es la salida válida, sino hasta que transcurre el tiempo de respuesta (retraso) de la compuerta.

Si el circuito se hubiera descrito fragmentado habría una diferencia en el tiempo de retraso calculado **en la simulación**, dado que el simulador relaciona un retraso delta al circuito completo asociado a una salida, no lo hace compuerta por compuerta.

Por ejemplo, para la descripción:

```
d <=S1 or S2 or S3;
S1 <= a and b;
S2 <= b and c;
S3 <= a and c;
```

O para la siguiente (que es equivalente a la anterior):

```
S1 <= a and b;
S2 <= b and c;
d <=S1 or S2 or S3;
S3 <= a and c;
```

©Editorial Digital Tecnológico de Monterrey

La simulación generaría los resultados que se muestran en la tabla 1.2.

t = 0	t = 0 + Δ	t = 0 + 2Δ
a = 0	a = 0	a = 0
b = 1	b = 1	b = 1
c = 1	c = 1	c = 1
S1 = U	S1 = 0	S1 = 0
S2 = U	S2 = 1	S2 = 1
S3 = U	S3 = 0	S3 = 0
d = U	d = U	d = 1

Tabla 1.2 Valores de señales en el tiempo

1.3.1 Definición de retrasos específicos

Como un delta de tiempo tiende a cero, cuando la simulación se observa en forma gráfica los deltas de tiempo no se distinguen. El simulador presenta un retraso de cero, es decir, muestra una respuesta instantánea.

Existe la posibilidad de definir un retraso específico para un circuito. Esto solo es posible con fines de simulación, ya que el retraso real de una compuerta no puede modificarse. De hecho, el retraso real de una compuerta no es un parámetro fijo, sino que depende de la temperatura del circuito.

Si se tiene el propósito didáctico de entender el retraso de un circuito, se definen retrasos de la siguiente manera: agregar "after X unidad de tiempo" al final de la definición de una salida y su circuito. Las unidades de tiempo válidas son: picosegundos

©Editorial Digital Tecnológico de Monterrey

(**ps**), nanosegundos (**ns**), microsegundos (**us**), milisegundos (**ms**), segundos (**s**). Por ejemplo, supongamos que se desea fijar el retraso de una compuerta **and** en **30 ns** y el de una compuerta **or** en **50 ns**.

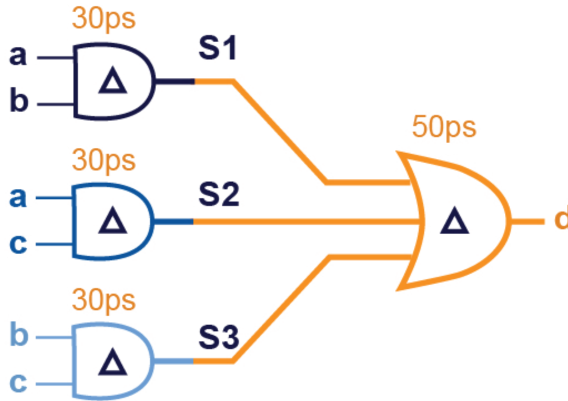


Figura 1.7 Ejemplo de retrasos de compuertas

La definición de este circuito en VHDL es:

```
architecture ecuaciones of comb_retraso is
signal S1, S2, S3: std_logic;
begin
d <= S1 or S2 or S3 after 50 ps;
S1 <= a and b after 30 ps;
S2 <= b and c after 30 ps;
S3 <= a and c after 30 ps;
end ecuaciones;
```

La simulación de este circuito, que consta en las salidas y el tiempo en que se vuelven estables, se muestra en la tabla 1.3.

©Editorial Digital Tecnológico de Monterrey

t = 0	t = 0 + 30ps	t = 0 + 80 ps
a = 0	a = 0	a = 0
b = 1	b= 1	b= 1
c = 1	c = 1	c = 1
S1 = U	S1 = 0	S1 = 0
S2 = U	S2 = 1	S2 = 1
S3 = U	S3 = 0	S3 = 0
d = U	d = U	d = 1

Tabla 1.3 Simulación con señales tipo std_logic

Existe otro tipo alterno al **std_logic**, es el tipo *bit*. La diferencia entre el tipo *bit* y el tipo std_logic reside en detalles que se presentarán en el capítulo 2; por lo pronto, se destaca la diferencia en la simulación de señales de tipo *bit*. Las señales de este tipo inician en cero, en vez de iniciar indeterminadas.

Usando este tipo, la definición del circuito es:

```
entity comb1 is
Port (a : in bit;
      b : in bit;
      c : in bit;
      d : out bit);
end comb1;

architecture ecuaciones of comb_retraso is
signal  S1, S2, S3: bit;
begin
d <= S1 or S2 or S3 after 50 ps;
S1 <= a and b after 30 ps;
S2 <= b and c after 30 ps;
S3 <= a and c after 30 ps;
end ecuaciones;
```

©Editorial Digital Tecnológico de Monterrey

La simulación reporta los valores que se aprecian en la tabla 1.4.

t = 0	t = 0 + Δ	t = 0 + 2Δ
a = 0	a = 0	a = 0
b = 1	b = 1	b = 1
c = 1	c = 1	c = 1
S1 = U	S1 = 0	S1 = 0
S2 = U	S2 = 1	S2 = 1
S3 = U	S3 = 0	S3 = 0
d = U	d = U	d = 1

Tabla 1.4 Simulación de señales tipo *bit*

La simulación de un circuito es un gran apoyo para entender su operación. Sin embargo, la estructura de datos que utiliza el simulador no es la misma que la que se utiliza para la configuración del circuito, por lo que en ocasiones ocurren inconsistencias. Esto sucede sobre todo en los circuitos secuenciales. Por ejemplo, es posible simular un registro que cuente con dos entradas de reloj y, sin embargo, no es imposible configurar este circuito, ya que los registros se consiguen conectando *flip flops* convencionales, con una sola entrada de reloj.

La regla general es: los circuitos que se pueden simular no necesariamente se pueden configurar; los circuitos que generan errores en su simulación seguramente conservarán esos errores en la implementación.

Se mostrarán otras simulaciones y más detalles acerca de las mismas a lo largo del libro.

©Editorial Digital Tecnológico de Monterrey

1.4 Expresiones booleanas en VHDL

En VHDL es común llamarle **ecuación** a una función booleana. En una ecuación se describe un circuito y se da un nombre a su salida. Una opción para describir un circuito es construir una expresión booleana (hay otras opciones que se abordarán más adelante).

El formato de una ecuación es:

Señal de salida <= expresión booleana

Los símbolos <= representan una flecha ← (este símbolo no se encuentra en el teclado, por eso no se utiliza). Se usa <= y no solo = porque en un circuito con retroalimentación la igualdad sería incorrecta. Por ejemplo, en álgebra booleana f = b + f representa la operación **or** entre la salida que también es entrada y **b**. Esto no es una igualdad matemática, es la representación de una salida lógica que también funge como entrada. Para no utilizar el símbolo de igualdad, en VHDL este circuito se describe como f <= f or a. En VHDL el operador + no representa una operación lógica, sino aritmética. Los operadores booleanos se representan con su nombre en inglés.

Una expresión está formada por operandos y operadores. Los operandos pueden ser variables booleanas o constantes. En este eBook no usaremos la palabra variable para que no haya confusión con las variables que se utilizan en programación. En vez de variable, emplearemos el término señal. Las constantes booleanas pueden ser cero, que en VHDL es '0' y uno '1'.

En cuanto a los operadores, además de los operadores básicos de la lógica **NOT**, **AND**, **OR**, en VHDL existen operadores para todas las compuertas lógicas. Así que es posible modelar un circuito con las compuertas simples que constituyen las operaciones básicas del álgebra booleana y las compuertas compuestas, es decir, que incluyen varias operaciones lógicas, como son: **XOR**, **XNOR**, **NAND**, **NOR**. Cabe aclarar que, aunque se utilicen compuertas compuestas, la expresión se construye de la misma forma que si se

©Editorial Digital Tecnológico de Monterrey

utilizaran operadores simples, por ejemplo: a NAND b NAND c. La operación XNOR corresponde a la operación de equivalencia, que es el complemento de la operación XOR.

En el lenguaje VHDL las mayúsculas y minúsculas son indistintas; la **a** y la **A** se interpretan igual. Así que se podrá escribir: **AND**, **aND**, **And**, and, **AnD**, **ANd**, **anD**, **aNd** y esto aplica a los nombres de señales, a los operadores y a las palabras reservadas.

Los nombres de los operadores son palabras reservadas del lenguaje.

En VHDL la precedencia de operadores no existe, de manera que todas las operaciones tienen la misma jerarquía. Al construir una expresión hágalo como si esta fuera a ser "evaluada" de izquierda a derecha, o bien como si el circuito fuera a ser conectado de izquierda a derecha.

Por otra parte, es posible definir expresiones booleanas en las que se usen compuertas de más de dos entradas, por ejemplo: f(a,b,-c,d) = a + b + c + d, que en VHDL es f <= a or c or d equivale al circuito que se muestra en la figura 1.8.

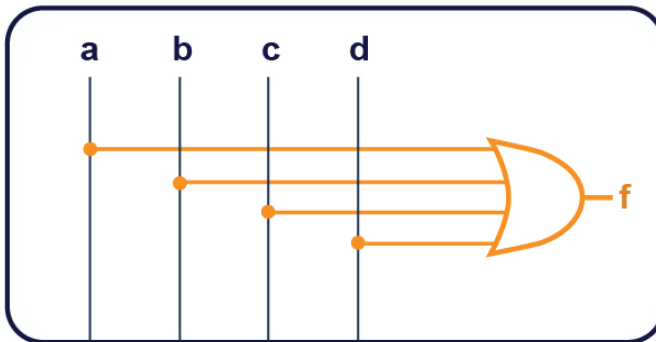


Figura 1.8 Circuito de la ecuación: f<=a or b or c or d

©Editorial Digital Tecnológico de Monterrey

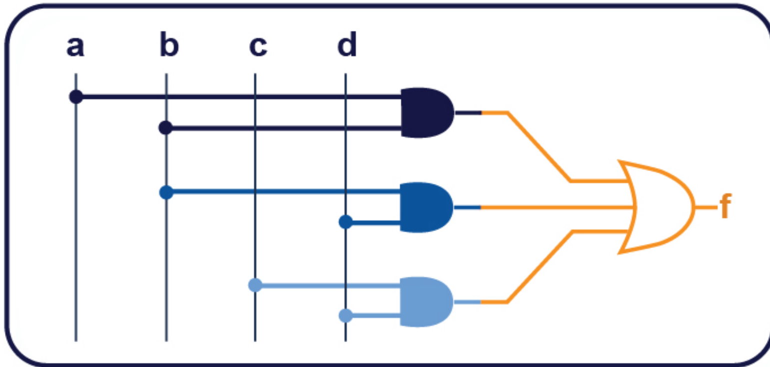


Figura 1.9 Circuito de la ecuación: f<=a and b or (b and d) or (c and d)

Es posible modelar el circuito de la figura 1.9 de la siguiente manera: **f<= a and b or (b and d) or (c and d)**. Observe que el primer **and** no requiere paréntesis por ser el primero de izquierda a derecha. Sin embargo, por presentar mayor uniformidad en la expresión, es posible utilizar paréntesis para todas las operaciones and, ya que si hay paréntesis extras no afectan a la expresión ni al circuito, de esta manera la función resulta: **f <= (a and b) or (b and d) or (c and d)** (figura 1.10). Esta expresión representa el mismo circuito que el anterior.

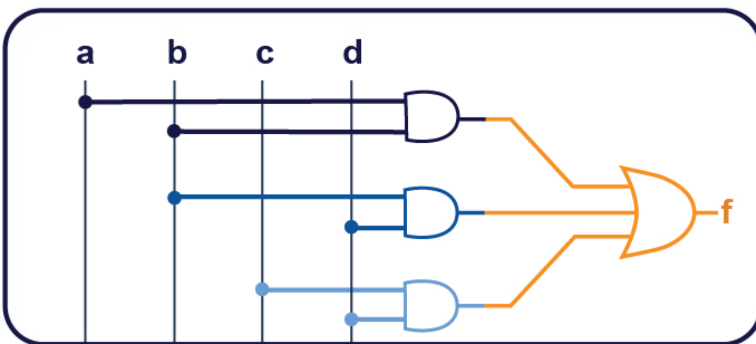


Figura 1.10 Circuito de la ecuación: f<= (a and b) or (b and d) or (c and d)

©Editorial Digital Tecnológico de Monterrey

Si la expresión se hubiera escrito sin paréntesis **f <= a and b or b and c or c and d** hubiera definido el circuito que se muestra en la figura 1.11.

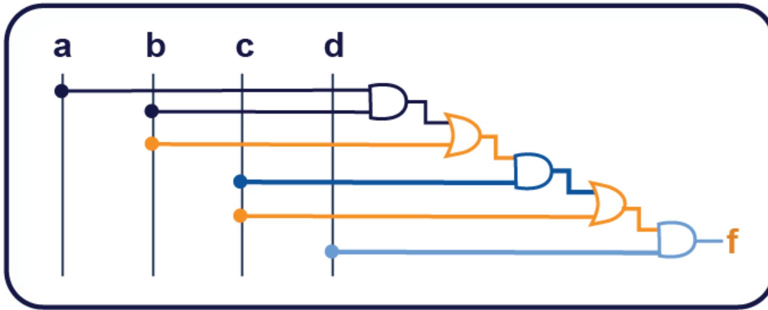


Figura 1.11 Circuito de la ecuación: f<=a and b or b and c or c and d

La ecuación equivalente con paréntesis es la siguiente: **f <=** (((((**a and b**) **or b**) **and c**) **or c**) **and d**).

El **not** es un operador unitario, es decir, es una compuerta con solo una entrada, así que en las expresiones no requiere paréntesis, por lo que **f <= not a and not b or c** representa al circuito que se aprecia en la figura 1.12.

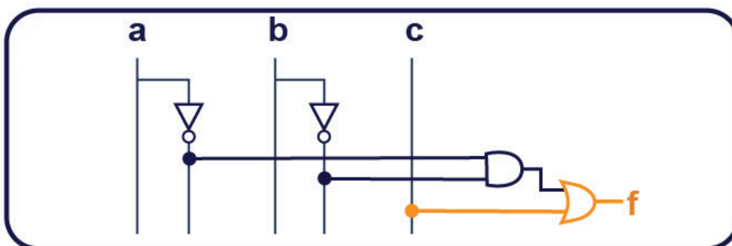


Figura 1.12 Circuito de la ecuación: F(a,b,c,d) = not a and not b or c

F <= a xnor b xnor c define el circuito que se presenta a continuación. Observe que el xnor es una compuerta de solo dos entradas, porque así está definida en lógica la operación de or exclusivo y equivalencia (figura 1.13).

©Editorial Digital Tecnológico de Monterrey

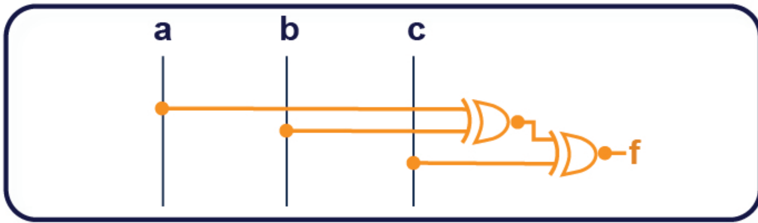


Figura 1.13 Circuito de la ecuación: f<= a xnor b xnor c

El circuito para f <= (a and b nand not c) nand not d es el que se aprecia en la figura 1.14.

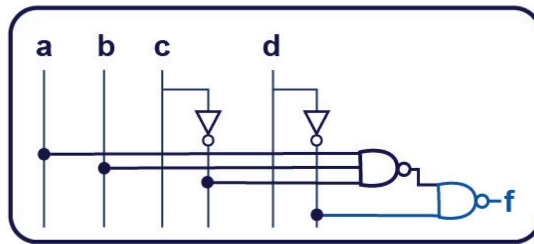


Figura 1.14 Circuito de la ecuación: f<= (a nand b nand not c) nand not d

Por último, el circuito para z <= (a and b) and not c or d es el que se observa en la figura 1.15.

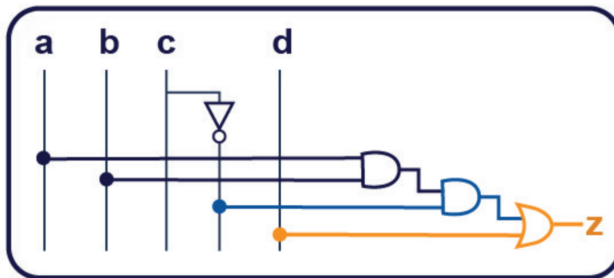


Figura 1.15 Circuito de la ecuación: z<= (a and b) and not c or d

©Editorial Digital Tecnológico de Monterrey

Actividad de repaso del tema 1.4

1. Describa en VHDL el cuerpo de la arquitectura (después de begin) del circuito de la figura 1.16. Llame **d** a la salida.

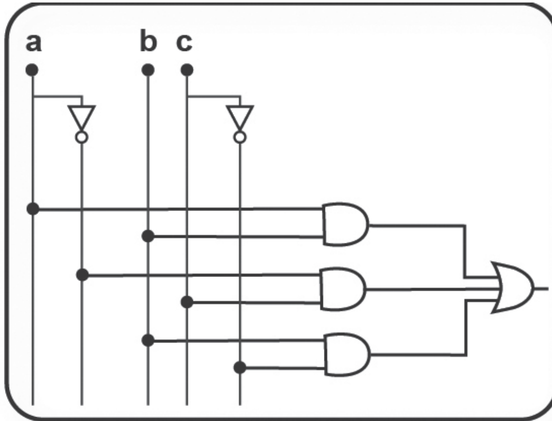


Figura 1.16 Circuito combinacional not-and-or

2. Suponga que el tiempo de respuesta (retraso) de un **not** es de **5ps**, de un **and 30ps** y de un **or 20ps**. Si en tiempo = 0 están listas las señales a, b y c.

a) ¿En qué momento, como máximo, es válida la salida del siguiente circuito (figura 1.17)?

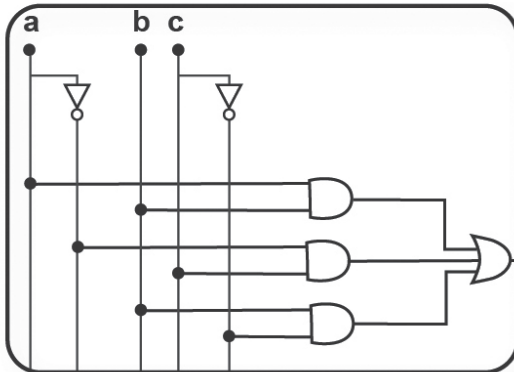


Figura 1.17

©Editorial Digital Tecnológico de Monterrey

b) ¿Y como mínimo? (tome en cuenta que al haber un 1en una compuerta or en cualquiera de sus entradas, luego de transcurrir tiempo de retraso, la salida es estable).

3. Realice las siguientes actividades.

a) Defina en VHDL el circuito secuencial que se muestra en la figura 1.18 (en VHDL es válido el operadornor).

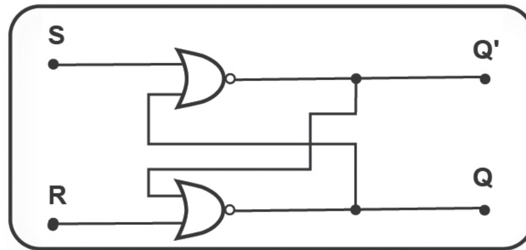


Figura 1.18

b) Analice el funcionamiento de este circuito e indique a qué circuito conocido corresponde.

c) Muestre cuál sería el resultado de la simulación de este circuito.

4. Diseñe en VHDL un dispositivo que genere dos señales oscilatorias, equivalentes a señales de reloj. Las señales se llamarán CLK1 y CLK2. CLK2 debe tener el doble de frecuencia que CLK1. El período de CLK1 debe ser de 20ns y el de CLK2 de 10ns.

5. Resuelva las siguientes actividades.

a) Describa en VHDL la arquitectura del circuito que se presenta en la figura 1.19 (no tome en cuenta los valores que están junto a los nombres de las señales).

©Editorial Digital Tecnológico de Monterrey

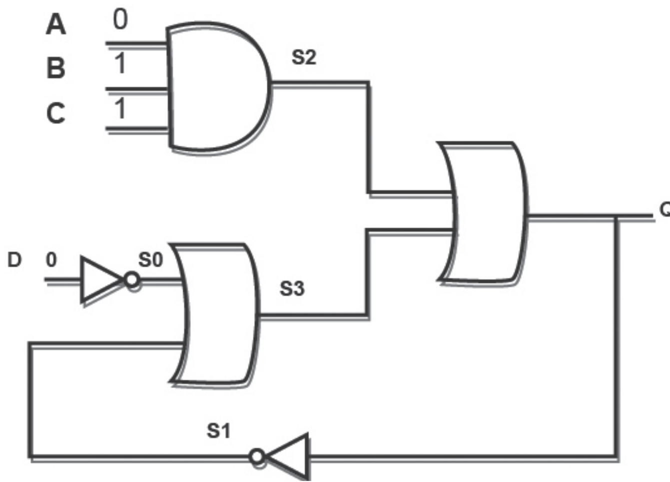


Figura 1.19

Los retrasos de las compuertas son:

Retraso not: 1ns

Retraso or: 2ns

Retraso and: 3ns

b) Construya la tabla de tiempo de retraso del circuito mostrado en el inciso a. Considere que en tiempo = 0 las señales tienen los valores que se indican.

©Editorial Digital Tecnológico de Monterrey

1.5 Más información sobre la definición de un circuito: entidad

La entidad es la sección en la que se define el nombre del circuito y su puerto. El puerto está constituido por las señales con las que el circuito se comunica con el exterior. En la figura 1.20 se describe un circuito con entradas **a**, **b**, **c** y salida **d**. Las entradas y salidas conforman el puerto, y las compuertas interconectadas son la arquitectura.

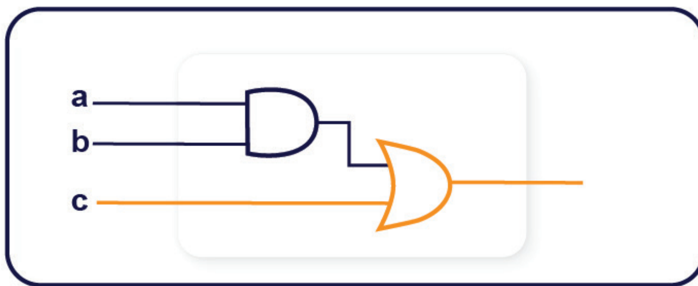


Figura 1.20 Descripción del circuito

La manera de declarar la entidad de este circuito en VHDL es:

```
Entity circuito_prueba is
      Port ( a, b, c: in std_logic;
             d: out std_logic);
end circuito_prueba;
```

Para la descripción en VHDL de una entidad es necesario que se defina un nombre para el circuito y también, al momento de definir las entradas y salidas, se le asocie a cada una un nombre, modo y tipo.

En el ejemplo anterior, una señal tiene de nombre '**a**', modo "**in**" y tipo "**std_logic**".

©Editorial Digital Tecnológico de Monterrey

1.6 Nombres

Si el circuito fuera construido en un *protoboard*, las señales representarían a los cables que van conectados a las entradas de los componentes o que provienen de sus salidas. Para hacer referencia a las señales en la arquitectura del circuito, es necesario etiquetarlas con un nombre; se recomienda que el nombre de la señal sea fácil de identificar, es decir, si existe una señal que se encargue de reiniciar algún proceso, entonces el nombre más indicado para la señal sería ***reset***.

Para crear un nombre correcto, es necesario conocer las reglas de construcción:

- VHDL no distingue entre mayúsculas. Es decir, las señales '**A**' y '**a**' se consideran iguales.
- Solo se permiten letras, números y guion bajo, debe comenzar con una letra y no puede contener dos guiones seguidos.
- No es posible utilizar palabras reservadas.
- No deben repetirse nombres para señales distintas.

Las mismas reglas de construcción aplican al nombre general del circuito que se indica después de **entity**.

Actividad de repaso del tema 1.6

En el siguiente fragmento de código se encuentran algunos errores respecto a los nombres de las señales. Identifíquelos y explique dónde se encuentra el error.

```
entity Nombres is
      port( a, A, ab12c, 1Abc, t_4: in std_ logic;
            y_xr, 1bt,_ce3, y_t_b: out std_logic);
  end Nombres;
```

©Editorial Digital Tecnológico de Monterrey

1.7 Tipos de datos

Los tipos de datos son utilizados para asignar un "formato lógico" a los niveles eléctricos de una señal. Generalmente, el diseñador usa el tipo de dato que más se ajuste al circuito que desea construir. Los tipos de datos son:

- **Std_logic**, que puede tomar los siguientes valores:
  - **0** o **1** lógico, es decir, un *bit*.
  - **Z** indica alta impedancia, es decir, desconexión.
  - **U** indica *undetermined* (indeterminado), esto ocurre cuando a una señal no se le ha asociado un valor, solo aplica a la simulación del circuito, así que indistintamente podría ser 0 o 1.
  - **X** indica un corto, y sucede cuando una señal recibe valores de distintas fuentes. Por ejemplo: **A<=B; A<=C**.
- *Bit*. Solo toma valores de **0** y **1**. A diferencia de **std_logic**, toma **0** como valor en la simulación cuando una señal no tiene valor.
- ***Integer*.** Permite que en la descripción de un circuito se utilicen valores con representaciones decimales enteras. Por ejemplo: **A <= 5**, en vez de A <= "101". El compilador que se utilice se encarga de convertir a binario los datos escritos en decimal. Si la señal está acotada, entonces el número se ajusta a un vector de *bits* de un tamaño acotado; si no, un entero se convierte a un bus de datos de 32 *bits*.
- **Std_logic_vector**. Bus de *bits* tipo std_logic. La definición de una señal múltiple, es decir, un bus de datos, el cual permite que se utilice solo un nombre para hacer referencia a un conjunto de señales digitales (o *bits*). Cada señal individual puede tomar los valores de un dato std_logic. La sintaxis que se utiliza y su uso se estudiarán en el capítulo 2.

©Editorial Digital Tecnológico de Monterrey

- *Bit*_**vector**. Es un “bus” de *bits* de tipo *bit*. En el capítulo 2 se estudiará su uso y definición.

Actividad de repaso del tema 1.7

Para las siguientes definiciones, indique cuál es su valor inicial en caso de simularlos. A, C: std_logic;

B: *bit*;

Indique si es correcta las siguientes asignaciones:

a) A<=’0’;

b) B<=’0’;

c) A<=C;

d) A<=B;

©Editorial Digital Tecnológico de Monterrey

1.8 Arquitectura: niveles de descripción

Como ya se mencionó, la arquitectura es la sección en la que se describe el circuito. Existen tres niveles de descripción de una arquitectura en VHDL: nivel estructural, nivel ecuaciones y nivel comportamiento o funcional. En seguida se explicarán estos niveles.

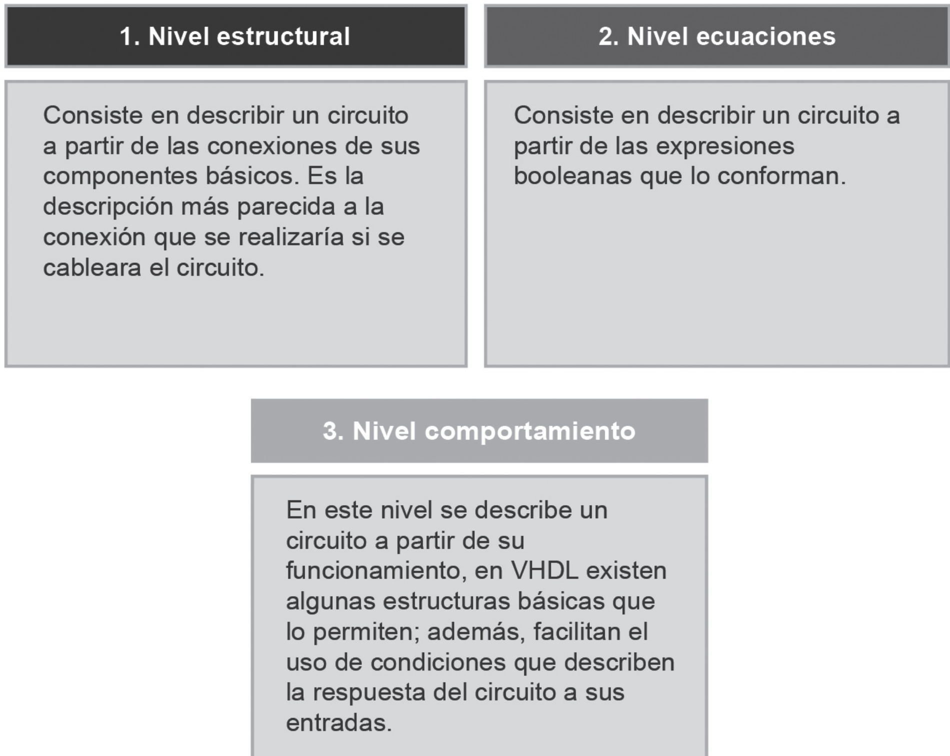


Figura 1.21

©Editorial Digital Tecnológico de Monterrey

1.9 Sintaxis de VHDL

A continuación, se muestra nuevamente la codificación de la arquitectura del circuito de la figura 1.1 para analizar su estructura sintáctica.

```
architecture ecuacion of comb1 is
begin
d <= (a and b) or (a and c) or (b and c);
end ecuacion;
```

Esta y cualquier otra descripción cumple con el siguiente patrón sintáctico. Sintaxis en VHDL de la definición de un componente:

```
entity nombre-componente is
       Port  (señal: tipo;
              señal: tipo;     ...
              señal: tipo;);
end nombre-componente ;

architecture tipo-de-descripción of nombre-
componente is
begin
instrucción_concurrente;
     instrucción_concurrente;
      ...
          instrucción_concurrente;
          end tipo-de-descripción ;
```

©Editorial Digital Tecnológico de Monterrey

La palabra concurrente tiene dos propósitos:

Propósito 1	Propósito 2
Todos los circuitos o segmentos de circuito que se describan en la arquitectura funcionan al mismo tiempo, es decir, son concurrentes.	Diferenciarse del tipo de instrucciones secuenciales

Figura 1.22

Las palabras que se encuentran en negritas (*bold*) son palabras reservadas del lenguaje. No son intercambiables por otras palabras y son requeridas en la posición que se indica. No es posible emplear palabras reservadas como nombre, componente ni señales del circuito.

Observe que al final de cada instrucción concurrente –y en general de cada sección de la descripción– se incluye un punto y coma. Este signo de puntuación no puede omitirse.

Las reglas para la construcción válida de los nombres de las señales se presentan en el capítulo 2.

```
architecture ecuacion of comb1 is
begin
d <= (a and b) or (a and c) or (b and c);
end ecuacion;
```

Volviendo al ejemplo dado:

La palabra **ecuacion** indica el nivel en que está codificada la arquitectura, y puede ser una palabra cualquiera elegida por el diseñador con el propósito de documentar el circuito. Por otra parte,

©Editorial Digital Tecnológico de Monterrey

comb1 es el nombre de la entidad, es decir, es el nombre del circuito, así, es posible definir señales internas (si las hay). A continuación, se incluye la descripción de la arquitectura que inicia con la palabra reservada **begin**. La descripción de la arquitectura finaliza por la palabra **end** seguida de la palabra que se haya utilizado para indicar el nivel de la arquitectura.

El siguiente ejemplo muestra un circuito con una señal interna para ilustrar el lugar sintáctico de la declaración. En este ejemplo, d y f representan nombres distintos para la misma señal.

```
architecture Behavioral of circuito_prueba is
  signal f: std_logic;
begin
  f <= (a and b) or c;
  d <= f;
end Behavioral;
```

Actividad de repaso del tema 1.9

Indique qué errores sintácticos tienen los siguientes códigos:

```
architecture Behavioral of circuito_prueba is

     signal f: std_logic;

begin

     f <= (a and b) or c

     d <= f

end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

```
architecture Behavioral of circuito_prueba is
     signal f: std_logic;
     f <= (a and b) or c;
     d <= f;
```

©Editorial Digital Tecnológico de Monterrey

1.10 Descripción de circuitos combinacionales con mayor número de elementos

Ya comprendidos los conceptos de entidad y arquitectura con los sencillos ejemplos que se presentaron y resolvieron, usted está listo para empezar a codificar en VHDL diseños más extensos. Enseguida le mostramos más ejemplos de circuitos combinacionales con mayor número de elementos:

Ejemplo 1. Circuito combinacional de múltiples niveles y compuertas diversas.

Empecemos codificando el circuito combinacional que se presenta en la figura 1.23, a nivel ecuaciones.

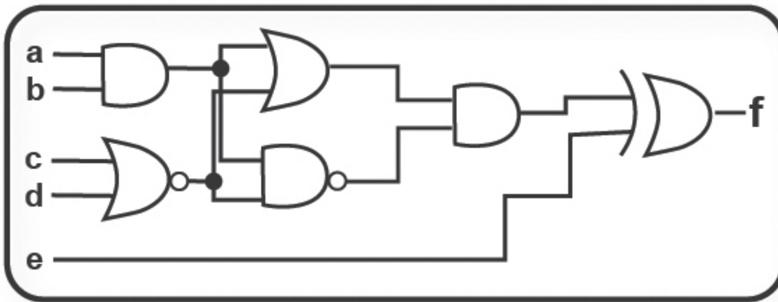


Figura 1.23 Circuito combinacional de múltiples niveles

La codificación en VHDL de este circuito a nivel ecuaciones puede realizarse mediante el uso de señales internas o en solo una ecuación. Para algunos resulta más cómodo y menos propenso a errores codificarlo con señales internas.

©Editorial Digital Tecnológico de Monterrey

La manera de codificarlo con el uso de señales es de la siguiente manera:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Ejemplo is
port(a,b,c,d,e: in std_logic;
           f : out std_logic);
end Ejemplo;

architecture Behavioral of tercerafigura is
signal g,h,i,j,k: std_logic;
begin
--f <= e xor ( ( (a and b) or (c nor d) ) and (
(a and b) nand (c nor d) ) );
g <= a and b;
h <= c nor d;
i <= g or h;
j <= g nand h;
k <= i and j;
f <= e xor k;
end Behavioral;
```

Nota: La manera en que se codificaría sin el uso de señales se encuentra en el código a manera de comentario. Los comentarios se incluyen después de dos guiones.

Ejemplo 2. Juego de TV

En un programa de concursos de televisión hay un juego en el que participan tres personas. Cada participante cuenta con un interruptor. Los interruptores cerrados (posición on) generan un 1 lógico. Los interruptores abiertos (posición off) generan un 0 lógico. Por otra parte, hay cuatro indicadores (LEDs) en un tablero: A, B, C y D. Un indicador enciende si recibe un 1 lógico.

©Editorial Digital Tecnológico de Monterrey

- El indicador A debe encender solo cuando todos los jugadores posicionan en on sus interruptores.
- El indicador B debe encender si y solo si A está apagado.
- El indicador C debe encender si y solo si dos o más jugadores posicionan en on sus interruptores.
- El indicador D debe encender si y solo si uno o ninguno de los jugadores posicionan en on sus interruptores.

Dadas las especificaciones anteriores, codifique en VHDL los siguientes circuitos:

1. Circuito para encender el indicador A.

2. Circuito para encender el indicador B.

3. Circuito para encender el indicador C.

4. Circuito para encender el indicador D.

Para solucionar este problema –y en el caso de cualquier circuito que sea desee codificar– es altamente recomendable realizar un esquemático antes de comenzar a programarlo, con el fin de tener más clara la idea para su elaboración.

Después de realizar las funciones booleanas necesarias para que este circuito funcione, se procede al esquemático mostrado en la figura 1.24.

©Editorial Digital Tecnológico de Monterrey

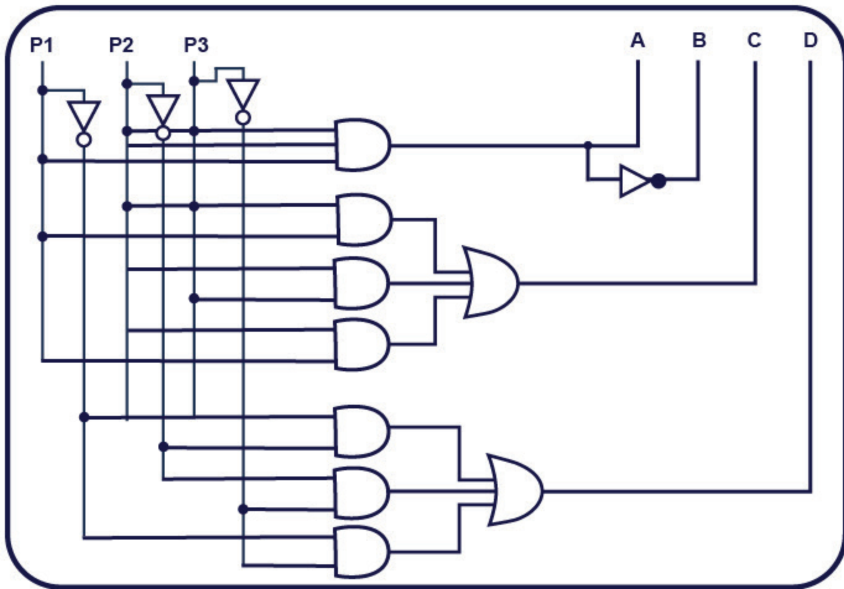


Figura 1.24 Diseño esquemático del juego de TV

A continuación, se muestra el juego de TV codificado basado en el esquemático del mismo.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TV is
  Port ( p1,p2,p3 : in  STD_LOGIC;
      a,b,c,d : out  STD_LOGIC);
end TV;

architecture Behavioral of TV is
 signal snp1, snp2, snp3: std_logic;
begin
 snp1 <= not p1;
 snp2 <= not p2;
 snp3 <= not p3;
 a <= p1 and p2 and p3;
 b <= not(p1 and p2 and p3);
 c <= (p1 and p2) or (p1 and p3) or (p2 and p3);
 d <= (snp1 and snp2) or (snp2 and snp3) or (snp1 and snp3);
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Ejemplo 3. Alarmas residenciales

Se está implementando el sistema de alarmas de un pequeño departamento. A usted se le ha contratado para diseñar y construir los circuitos que hagan funcionar el sistema.

El sistema cuenta con sensores, interruptores y diferentes tipos de alarmas.

Se instalaron cinco tipos de sensores:

- Magnéticos, para las ventanas.
- Infrarrojos, para detectar movimiento.
- De humo.
- De CO2 (detecta que pase cierto nivel).
- De humedad (detecta que pase cierto nivel).

El número de sensores que se instalaron fue:

- Dos magnéticos.
- Un infrarrojo.
- Uno de humo.
- Uno de CO2.
- Uno de humedad.

Los sensores deben activar cinco tipos de alarmas:

- Contra robo.
- Contra incendios.

©Editorial Digital Tecnológico de Monterrey

- Para avisar sobre la humedad.
- Para avisar sobre el CO2.
- Para avisar que hay alguna alarma encendida.

Se colocaron dos interruptores, que en “ON (1)” sirven para:

- Activar la alarma de robos.
- Indicar que hay gente en casa (que probablemente encienda estufa, cerillos, etc.).

Para implementar el diseño (piense en un prototipo):

- Utilice dip *switches* para representar a los sensores e interruptores.
- Utilice LEDs para representar a las alarmas.

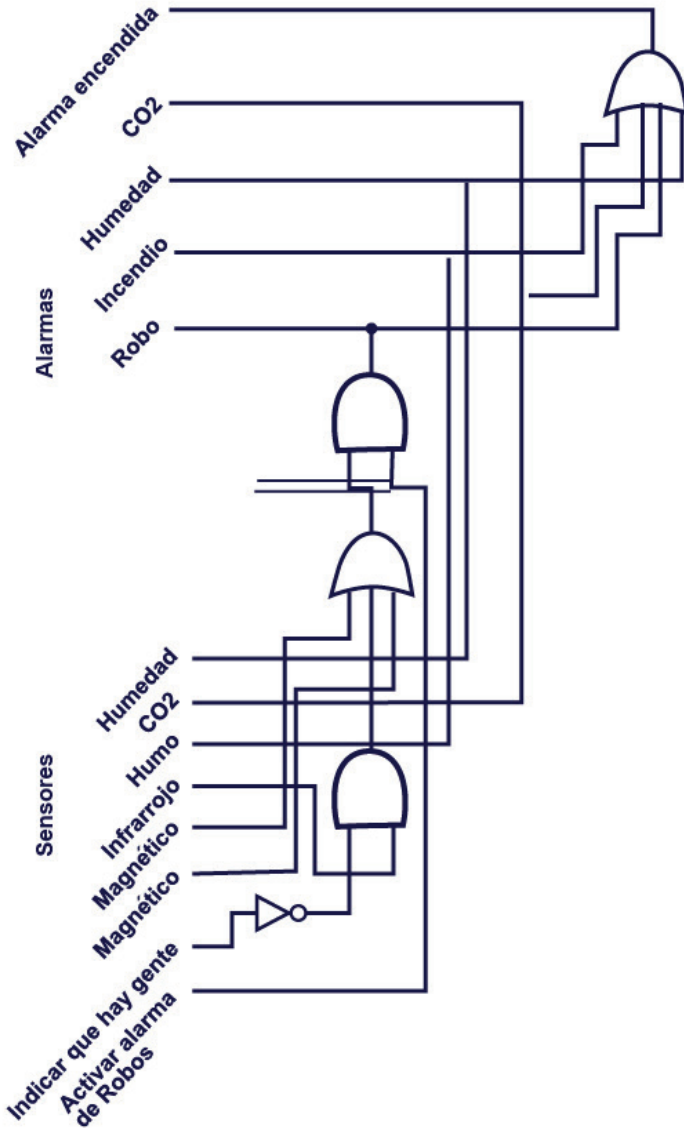
Un diseño puede ser el siguiente:

- La alarma de robos se enciende cuando el interruptor de alarma de robo se encuentre en ON y el interruptor que indica presencia de residentes se encuentre en OFF.
- Teniendo en cuenta los interruptores, la alarma se encenderá cuando el sensor infrarrojo esté en 1 o alguno de los sensores magnéticos esté en 1.
- La alarma de incendio se encenderá cuando el sensor de humo detecte un alto nivel de humo.
- La alarma de humedad dependerá únicamente del sensor de humedad.
- La alarma de CO2 se encenderá solo si la alarma de CO2 lo indica.
- La alarma “alguna alarma encendida” se prenderá si cual-

©Editorial Digital Tecnológico de Monterrey

quiera de las alarmas está encendida.

De esta manera, el esquemático se representaría tal como se observa en la figura 1.25.



©Editorial Digital Tecnológico de Monterrey

Figura 1.25 Diseño esquemático de alarmas residenciales

La codificación de este diseño esquemático es:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Alarmas is
  Port ( iar : in  STD_LOGIC; --interruptor alarma de robo
     iig : in  STD_LOGIC; --interruptor indica hay gente
     sm1 : in  STD_LOGIC; --sensor magnetico 1
     sm2 : in  STD_LOGIC; --sensor magnetico 2
     sifr : in  STD_LOGIC; --sensor infrarojo
     shumo : in  STD_LOGIC; --sensor humo
     sco : in  STD_LOGIC; --sensor co2
     shume : in  STD_LOGIC; --sensor humedad d
     arob : inout  STD_LOGIC; --alarma de robo
     ainc : inout  STD_LOGIC; --alarma incendio
     ahume : inout  STD_LOGIC; --alarma humedad d
     aco : inout  STD_LOGIC; -- alarma co2
     aenc : inout  STD_LOGIC); -- alarma encendida
end Alarmas;

architecture Behavioral of Alarmas is

begin
arob <= ((not iig and sifr) or sm1 or sm2) and iar;
ainc <= shumo;
ahume <= shume;
aco <= sco;
aenc <= arob or ainc or ahume or aco;
end Behavioral;
```

1.10.1 Modos

El modo define la dirección o sentido de conexión de una señal. Se definen en el puerto del circuito. Existen cuatro modos: **in** (entrada), **out** (salida), **inout** (entrada/salida) y **buffer**.

©Editorial Digital Tecnológico de Monterrey

- **Modo in**. Se utiliza para definir a las señales de entrada al circuito.

- **Modo out**. Se refiere a las señales de salida del circuito.

- **Modo inout**. Indica que una señal del puerto puede ser tanto de entrada como de salida (bidireccional).

- **Modo buffer**. Se usa para señales de salida que internamente son utilizadas también como entrada.

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 1

1. Codifique en VHDL un circuito *encoder* de 8 a 3. Utilice solamente funciones booleanas. El *encoder* debe operar de la siguiente manera:

Se definirá un componente que cuenta con 8 entradas (D_IN) y 3 salidas (D_OUT). Suponga que solo una de las entradas que se den a D_IN es 1. La salida reflejará cuál de las entradas, de la 7 a la 0, es 1.

Resumiendo, la operación de este dispositivo se tiene el siguiente comportamiento de la salida de acuerdo a la entrada.

D_IN	D_OUT
00000001	000
00000010	001
00000100	010
00001000	011
00010000	100
00100000	101
01000000	110
10000000	111

Tabla 1.5

©Editorial Digital Tecnológico de Monterrey

Suponga que a la entrada del dispositivo siempre presenta una combinación válida. Al diseñar no tome en cuenta las combinaciones inválidas, puede emitir cualquier combinación que usted elija ante una entrada inválida ("*garbage in*, *garbage out*").

Utilice la siguiente definición para el puerto del *encoder*:

```
entity encoder is
Port (D_IN: in std_logic_VECTOR(7 downto 0);
D_OUT: out std_logic_VECTOR(2 downto 0));
end encoder;
```

2. Se tiene un tanque de agua con cinco sensores de nivel. Los sensores representan el nivel del tanque: O (dentro del rango deseado), H (alto), VH (muy alto), L (bajo) y VL (muy bajo), Cuando los sensores están en 1 es que el borde del agua se encuentra en ese nivel. A su vez, hay dos bombas, In y Out, que controlan la entrada y la salida de agua, y cada bomba tiene un control de encendido y un control de velocidad (L/H, bajo/alto), ambos digitales. Un '1' en la entrada de velocidad hará que la bomba funcione a su velocidad alta, y un '0' quiere decir que funcionará a su velocidad baja, esto siempre y cuando la bomba se encuentre encendida. El líquido en el tanque se gasta debido a un proceso industrial externo al tanque.

Se desea que haga un control sencillo que accione la bomba de salida cuando el nivel de agua esté por encima del nivel deseado, a velocidad baja si el nivel es H y a velocidad alta si es VH. Asimismo, debe activar la bomba de entrada a velocidad baja si el nivel es L y a velocidad alta si es VL. Si el nivel está en el rango deseado, ambas bombas deben estar apagadas. Adicionalmente, debe generar una señal de error si la salida del sensor es inválida. Esta señal se usa para proteger a las bombas y evitar que se dañen. Considere que el error ocurre si ambos encendidos están en 1. Aunque la velocidad

©Editorial Digital Tecnológico de Monterrey

en cualquiera de las bombas esté en 1 no ejerce acción si la bomba está apagada.

El puerto se muestra en la figura 1.26.

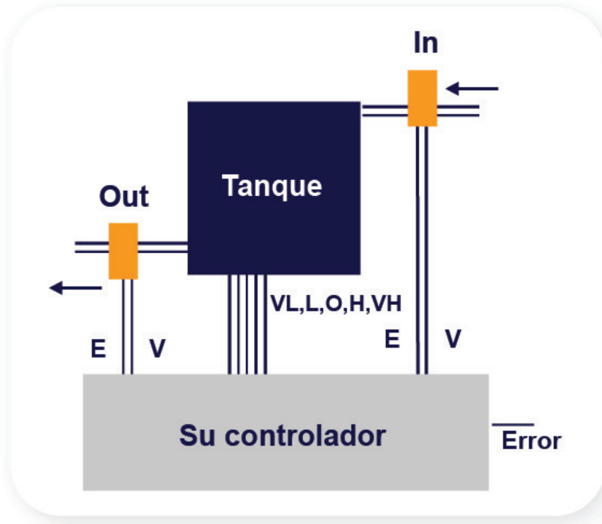


Figura 1.26

Defina en VHDL la arquitectura en el espacio que se dejó para ello. No es necesario presentar el diseño esquemático de este circuito combinacional.

3. Hay un juego al que llamaremos el "juego de la adivinanza", que consiste en detectar si dos personas piensan y ejecutan la misma jugada. Cada jugador cuenta con dos botones, el #1 y el #2. Suponga que los botones están ocultos debajo de una mesa, o bien que hay alguna separación que no deja ver los botones del contrincante. Para cada jugador una jugada consiste en oprimir cero, uno o dos botones. Diseñe un circuito que encienda una salida cuando los dos jugadores

©Editorial Digital Tecnológico de Monterrey

hayan efectuado la misma jugada. Lo que detectará es que los botones #1 y #2 de ambos jugadores hayan quedado en la misma posición en un momento dado.

La figura 1.27 muestra las entradas y salidas de este problema.

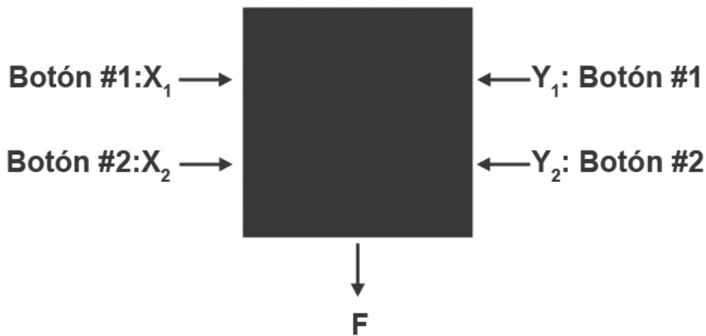


Figura 1.27 Puerto del juego de la adivinanza

El puerto se muestra a continuación. Defina en VHDL la arquitectura en el espacio que se dejó para ello. No es necesario presentar el diseño esquemático de este circuito combinacional.

4. En el siguiente conjunto de circuitos: A, B, C, representan un número binario (figura 1.28). Indiqué qué operación aritmética realizan al generar f3, f2, f1, f0, que a su vez es otro número binario.

©Editorial Digital Tecnológico de Monterrey

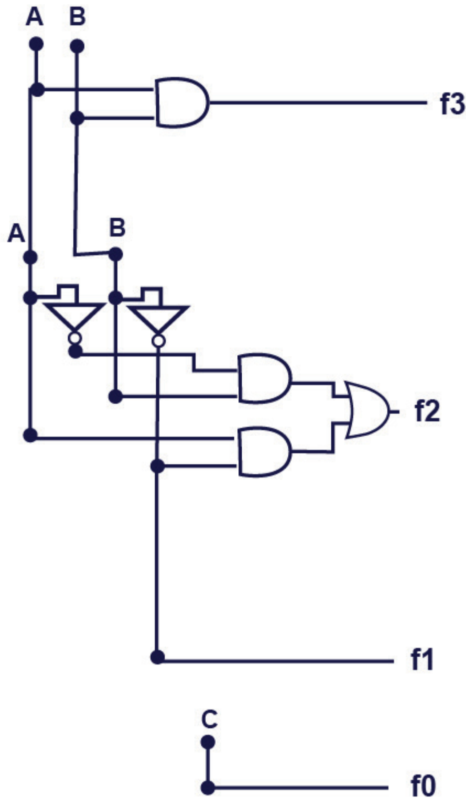


Figura 1.28 Circuito aritmético

5. Un automóvil tiene un sistema de seguridad basado en varios sensores, los cuales se enlistan enseguida:

a) Sensor de llave (K): genera un 1 cuando está puesta la llave, 0 de otro modo.

b) Dos sensores (D) colocados uno en cada una de las dos puertas que indican con 1 si la puerta está cerrada, 0 si está abierta.

©Editorial Digital Tecnológico de Monterrey

c) Cuatro sensores (T) colocados uno en cada una de las cuatro llantas que indican con 1 si la llanta tiene una presión diferente a 28 lbs, 0 si es igual a 28 lbs.

d) Un sensor colocado en el pedal de freno (B), que indica con 1 si está presionado, 0 si no lo está.

Además, hay un interruptor (S) que puede utilizar el conductor para que no se tomen en cuenta los sensores descritos en 3. Si el interruptor se cierra, genera un 1 y esto indica que no se deben tomar en cuenta los sensores de llantas; si se deja abierto, genera un 0 e indica que se tome en cuenta la presión de las llantas.

Se le solicita diseñar, en VHDL, las funciones necesarias para generar las salidas que irán conectadas a las siguientes alarmas:

- Alarma 1 de llantas: toma el valor de 1 cuando alguna de las llantas tiene una presión diferente de 28 lbs, 0 de otro modo.
- Alarma 2 de puertas: toma el valor de 1 cuando alguna de las puertas esté abierta, 0 de otro modo.
- Alarma general: se activa cuando hay alguna alarma encendida.
- Diseñe una salida (Out1) que se conectará al circuito de encendido del carro, se debe generar 1 para que este encienda, esto debe ocurrir cuando:

  1. Está la llave puesta.

  2. Están las dos puertas cerradas.

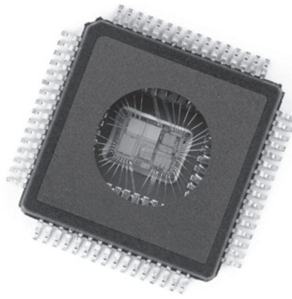
  3. Están las cuatro llantas a 28 lbs o el interruptor está cerrado.

  4. Está presionado el pedal de freno.

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 1

En este capítulo se ha presentado una introducción al lenguaje de modelación de circuitos VHDL. Se inició el aprendizaje de VHDL describiendo circuitos combinacionales utilizando álgebra booleana. Después de estudiar este capítulo, el lector debe ser capaz de reconocer las entradas, salidas y señales intermedias de un circuito combinacional. También debe poder plantear las ecuaciones booleanas necesarias para describir el circuito.



©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 2. Niveles de descripción de un circuito

Niveles de descripción de un circuito

- Definiciones del mux 4:1 en diferentes niveles
- Verificación del if
- Circuitos combinacionales diseñados con procesos
- Uso del case

En el capítulo 1 se mencionó que hay tres tipos (o niveles) de descripción para un circuito: estructural, ecuaciones booleanas y comportamiento. Un buen ejemplo con el que se observarán claramente los tres tipos de descripción lo constituye la definición de un *multiplexer* (selector). Se recordará que un selector de dos entradas de control cuenta con cuatro entradas de las cuales se elige la salida. Su dibujo (como bloque esquemático general) se muestra en la figura 2.1.

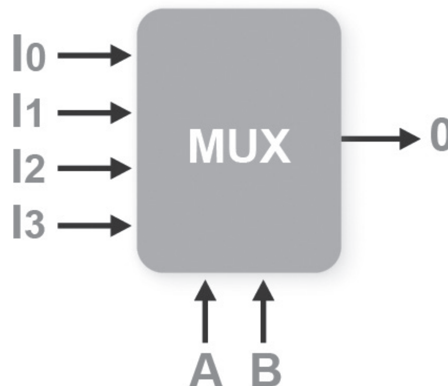


Figura 2.1. Dibujo esquemático de un *multiplexer*

©Editorial Digital Tecnológico de Monterrey

El circuito interno es el que se aprecia en la figura 2.2.

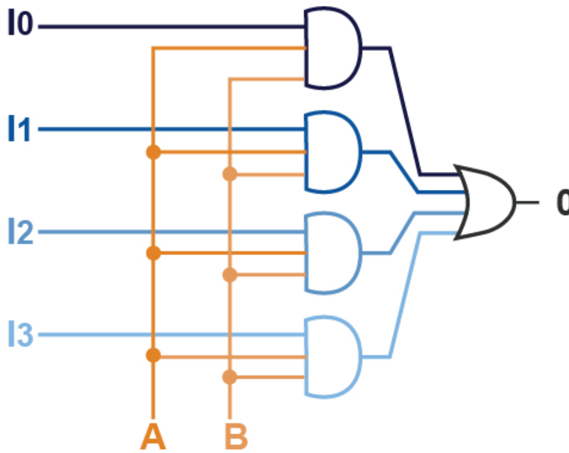


Figura 2.2 Circuito interno del *multiplexer* con dos entradas de control

La función booleana es:

```
O = I0A'B' + I1A'B + I2AB' + I3AB
```

A continuación, se presentan los diferentes niveles de definición para este ejemplo, selector de dos entradas de control (que se abreviará como mux 4:1, o bien por su nombre en inglés: *multiplexer*).

©Editorial Digital Tecnológico de Monterrey

2.1 Definiciones del mux 4:1 en diferentes niveles

2.1.1 Nivel ecuaciones

Para describir el selector en este nivel, se utiliza la función booleana:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mux_ecuaciones is
 Port ( I0 : in std_logic;
      I1 : in std_logic;
      I2 : in std_logic;
      I3 : in std_logic;
      A : in std_logic;
      B : in std_logic;
      O : out std_logic);
end mux_ecuaciones;

architecture ecuaciones of mux_ecuaciones is

begin
O <= (I0 and (not A) and (not B)) or
        (I1 and (not A) and B) or
        (I2 and A and (not B)) or
        (I3 and A and B);
end ecuaciones;
```

©Editorial Digital Tecnológico de Monterrey

2.1.2 Nivel estructural

Para definir un circuito en nivel estructural es preciso conectar los componentes que lo conforman. Los elementos pueden tener diferentes niveles de integración (número de compuertas lógicas).

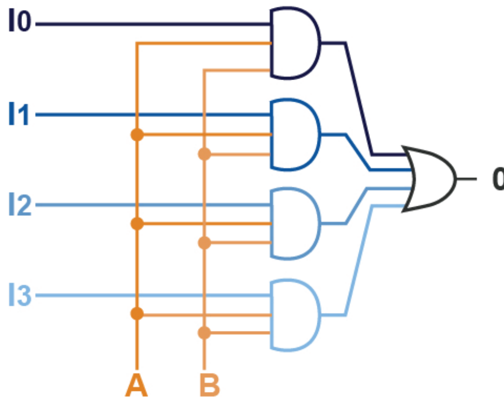


Figura 2.3 Circuito de un selector de dos entradas de control

Como un selector es un circuito muy simple, los elementos que lo conforman son compuertas lógicas, como se aprecia en la figura 2.4, donde el último es el resumen esquemático que representa al mux.

©Editorial Digital Tecnológico de Monterrey

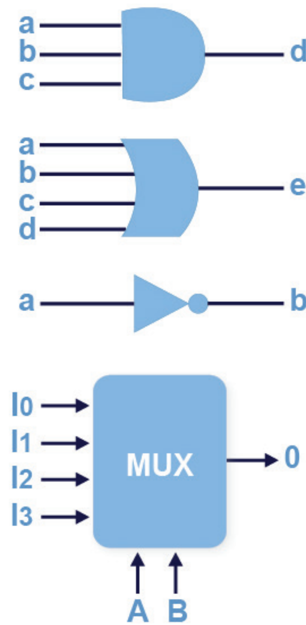


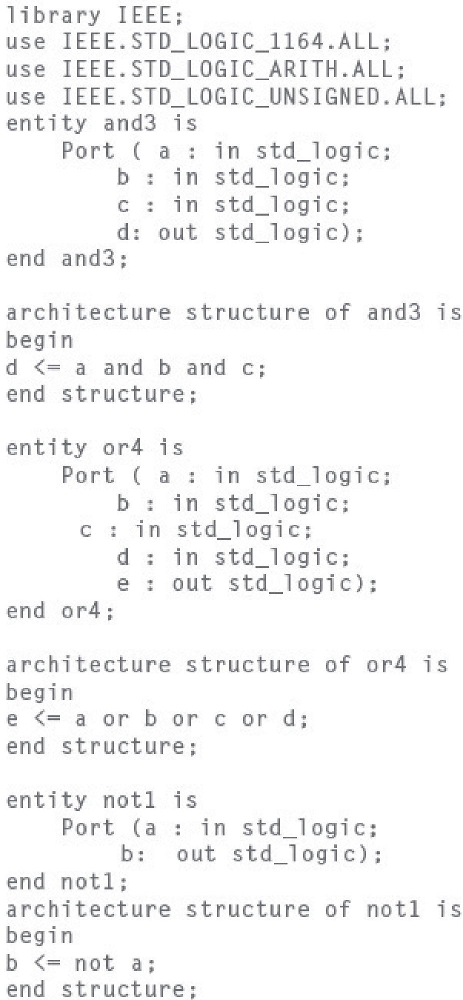
Figura 2.4 Elementos que conforman el circuito

A continuación, se expone la interconexión de los elementos anteriores para ejemplificar el nivel estructural, aunque regularmente no se interconectan elementos tan básicos como las compuertas lógicas. Primero se define la entidad y arquitectura de cada elemento y luego se conectan instanciándolos.

Observe que se define una compuerta **and** para tres entradas, una **or** para cuatro entradas –esto porque es justo lo que requiere este circuito– y una **not**. Luego se define la entidad del *multiplexer*. Recuerde que no es posible utilizar nombres como **and** u **or** porque son palabras reservadas del lenguaje. Con el nombre del componente las palabras reservadas **port map** se instancia (es decir, se conecta creando una instancia del componente) y enseguida se indican las conexiones que se realizan a las entradas y salidas del componente. Es preciso seguir el orden de las conexiones indicadas en el puerto. En este ejemplo se repitieron nombres de señales en los puertos de los diferentes componentes y esto no tiene importancia, pueden ser los mismos o pueden diferir. Finalmente son nombres locales a cada componente.

©Editorial Digital Tecnológico de Monterrey

Las arquitecturas de las compuertas se definen con sus ecuaciones, no existe otra forma.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity and3 is
    Port ( a : in std_logic;
        b : in std_logic;
        c : in std_logic;
        d: out std_logic);
end and3;

architecture structure of and3 is
begin
d <= a and b and c;
end structure;

entity or4 is
    Port ( a : in std_logic;
        b : in std_logic;
      c : in std_logic;
        d : in std_logic;
        e : out std_logic);
end or4;

architecture structure of or4 is
begin
e <= a or b or c or d;
end structure;

entity not1 is
    Port (a : in std_logic;
        b:  out std_logic);
end not1;
architecture structure of not1 is
begin
b <= not a;
end structure;
```

Cada componente se define como un circuito, con su entidad y arquitectura.

©Editorial Digital Tecnológico de Monterrey

Finalmente es posible definir la arquitectura del *multiplexer*. Para realizar las interconexiones habrá que definir algunas señales internas que se muestran en la figura 2.5.

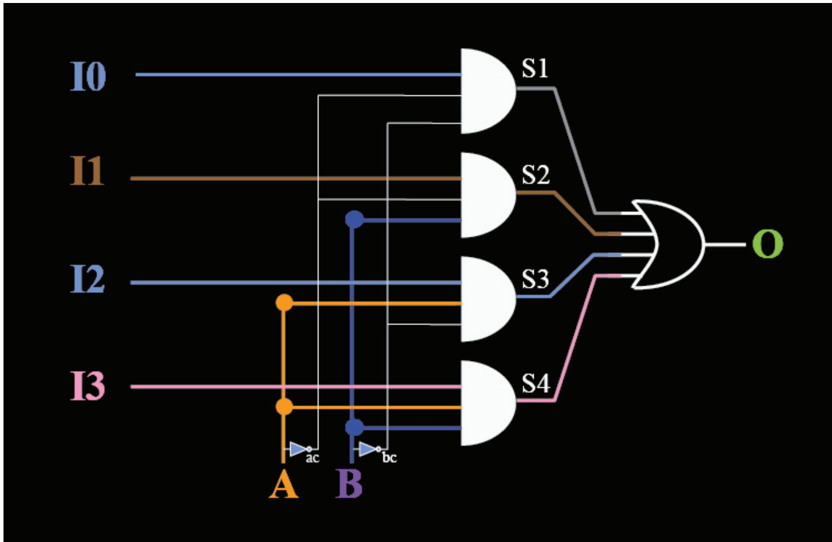


Figura 2.5 Circuito de referencia

Cada circuito, en este caso cada compuerta, es un componente del selector. Para hacer referencia a un componente, que se define afuera de la arquitectura de un circuito, es necesario indicarlo a través de la estructura sintáctica **component** y **a** continuación incluir una copia de su puerto.

©Editorial Digital Tecnológico de Monterrey

Una definición del selector completo a nivel estructural es la siguiente:

```
entity mux_estructural is
  Port (I0, I1, I2, I3, A, B: in std_logic;
       O: out std_logic);
end mux_estructural;

architecture estructural of  mux_estructural is
signal ac, bc, S0, S1, S2, S3:std_logic;
component and3 is
  Port ( a : in std_logic;
      b : in std_logic;
c : in std_logic;
        d : out std_logic);
end component;
component or4 is
    Port ( a : in std_logic;
        b : in std_logic;
    c : in std_logic;
    d : in std_logic;
         e : out std_logic);
end component;
component not1 is
    Port ( a: in std_logic;
         b: out std_logic);
end component;
begin
et1: not1 port map (a, ac);
et2: not1 port map (b, bc);
et3: and3 port map (ac, bc, I0, S0);
et4: and3 port map (ac, b, I1, S1);
et6: and3 port map (a, bc, I2, S2);
et7: and3 port map (a, b, I3, S3);
et8: or4  port map (S0, S1, S2, S3, O);
end estructural;
```

La sintaxis que corresponde a la referencia de un componente requiere iniciar con una etiqueta, como por ejemplo **et1**, **et2**, etc. A continuación, se indica el nombre del componente –el nombre utilizado en la entidad del componente– y posteriormente, entre paréntesis, se indican las conexiones que se realizarán a los elementos del puerto, las conexiones a las señales de entrada y las señales que se conectarán a las salidas. Por ejemplo, si describimos en álgebra booleana la señal intermedia **S1** sería: **S1 = a'b**.

©Editorial Digital Tecnológico de Monterrey

Cuando se hace una referencia a un componente se dice que se crea una instancia de este, es decir, un ejemplo, copia o clon del componente. Para efectos prácticos, el componente se utiliza las veces que sea necesario y solo se define una vez. De esta manera, en el ejemplo anterior, el circuito consta de dos compuertas **not**, cuatro compuertas **and** y una compuerta **or**.

Esta es una instancia del componente and3:

```
et7: and3 port map (a, b, I3, S3);
```

Para fines de simulación y de síntesis del circuito (para su implementación en un FPGA), esta instancia es exactamente equivalente a lo siguiente. Se aclara que la síntesis es la etapa en la que se reconocen los componentes.

```
S3 <=a and b and I3;
```

Hay dos tipos de referencias estructurales: las referencias cortas presentadas en el ejemplo anterior, en las que se indica qué se conectará a cada señal del puerto del componente que se instancia, y las referencias extendidas, como se muestra a continuación.

```
et1: not1 port map (a=>a, b=>ac);

et2: not1 port map(a=>a, b=>bc);

et6: and3 port map( a=>a, b=>bc, c=>I2, d=>S2);
```

En estas referencias se explicita a qué señal del puerto se realizará cada conexión. Esto se señala por medio de los símbolos **=>** (que representan una flecha →, como ya se había aclarado en el capítulo 1). Por otra parte, es posible realizar una definición híbrida,

©Editorial Digital Tecnológico de Monterrey

estructural y con ecuaciones, por ejemplo, escogiendo aleatoriamente dos referencias estructurales y cambiándolas por ecuaciones.

et1: not1 port map (a=>a, b=>ac);

et2: not1 port map(a=>a, b=>bc);

S0<=ac and bc and I0;

Es importante aclarar que no es válido conectar a una entrada una expresión lógica, solo es posible conectar una señal simple sin compuertas.

Ejemplo: et4: and3 port map (not a, b, I1, S1).

Lo que está en **bold** es inválido. Esta referencia estructural se debe modificar como sigue:

```
ac <= not a;
et4: and3 port map (ac, b, I1, S1).
```

Sin embargo, en versiones de 2019 en adelante, esta opción ya se ha modificado y se permiten expresiones booleanas en las conexiones.

Por otro lado, es posible definir componentes (para luego referenciarlos) en un mismo documento, colocando primero todas las entidades que se van a instanciar (en cualquier orden) y la entidad principal al último. A continuación, se definen todas las arquitecturas de los componentes externos, en cualquier orden y la arquitectura principal al final, de tal forma que quede. Ejemplo:

```
entity and3 is
  Port ( a : in std_logic;
       b : in std_logic;
       c : in std_logic;
       d: out std_logic);
end and3;
```

©Editorial Digital Tecnológico de Monterrey

```
entity or4 is
    Port ( a : in std_logic;
        b : in std_logic;
     c : in std_logic;
          d : in std_logic;
          e : out std_logic);
end or4;

entity not1 is
    Port (a : in std_logic;
         b:  out std_logic);
end not1;

entity mux_estructural is
  Port (I0, I1, I2, I3, A, B: in std_logic;
       O: out std_logic);
end mux_estructural;

architecture structure of and3 is
begin
d <= a and b and c;
end structure;

architecture structure of or4 is
begin
e <= a or b or c or d;
end structure;

architecture structure of not1 is
begin
b <= not a;
end structure;

architecture estructural of  mux_estructural is
signal ac, bc, S0, S1, S2, S3:std_logic;
```

©Editorial Digital Tecnológico de Monterrey

```
component and3 is
    Port ( a : in std_logic;
       b : in std_logic;
c : in std_logic;
       d : out std_logic);
end component;

component or4 is
    Port ( a : in std_logic;
       b : in std_logic;
      c : in std_logic;
      d : in std_logic;
       e : out std_logic);
end component;

component not1 is
    Port ( a: in std_logic;
        b: out std_logic);
end component;

begin
et1: not1 port map (a, ac);
et2: not1 port map (b, bc);
et3: and3 port map (ac, bc, I0, S0);
et4: and3 port map (ac, b, I1, S1);
et6: and3 port map (a, bc, I2, S2);
et7: and3 port map (a, b, I3, S3);
et8: or4 port map (S0, S1, S2, S3, O);
end estructural;
```

Otra opción es dejar por separado, en diferentes documentos, pero en el mismo proyecto, los diversos componentes. Esta es una mejor opción porque las diferentes versiones de compiladores permiten opciones diversas para el acomodo de los componentes en un mismo documento, así que hay la posibilidad de que el orden anterior cause errores. Por esto, la opción segura es la siguiente:

Tal como se había mencionado anteriormente, es posible utilizar esta versión corta de la instancia *port map*, o bien la extendida.

©Editorial Digital Tecnológico de Monterrey

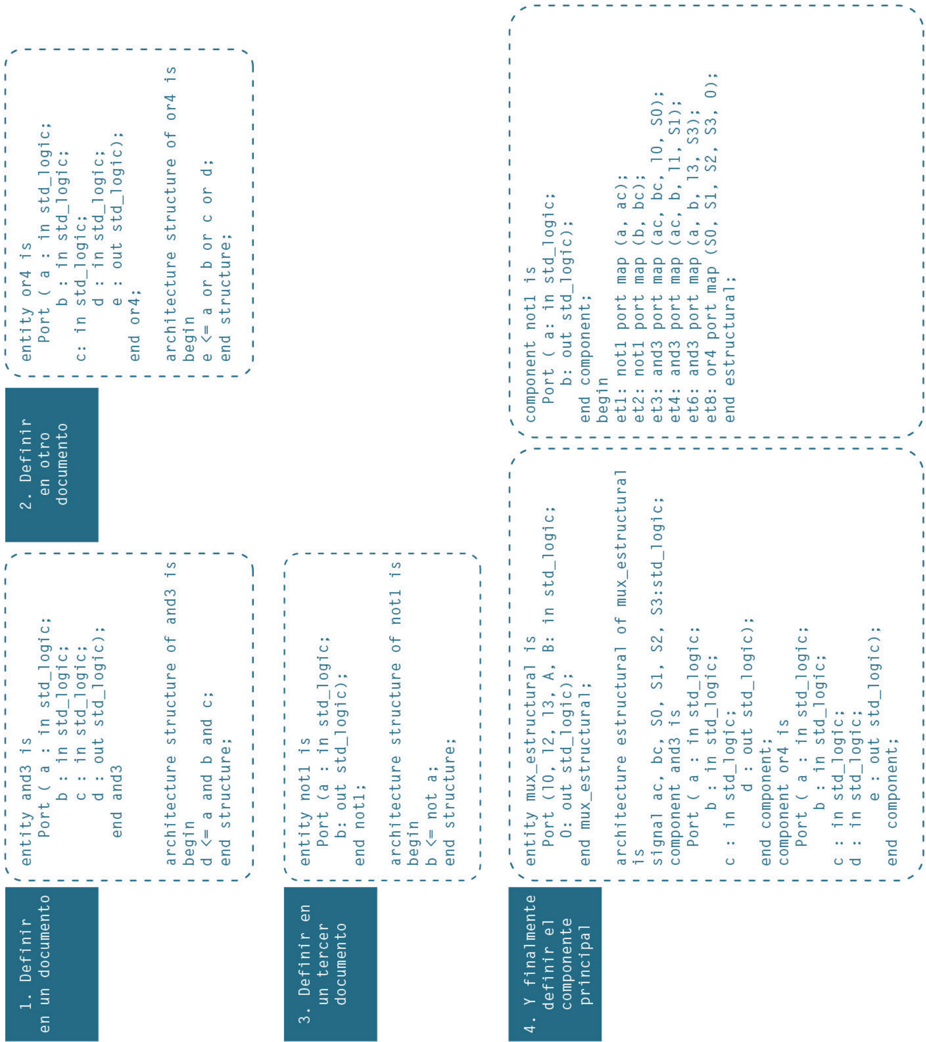


Figura 2.6

©Editorial Digital Tecnológico de Monterrey

2.1.3 Nivel comportamiento

El nivel comportamiento se usa para describir un circuito a partir de su funcionamiento. VHDL cuenta con diversos formatos de instrucciones para este propósito. A continuación, se presentan tres opciones para el *multiplexer.*

```
entity muxs is
  Port ( I0 : in std_logic;
       I1 : in std_logic;
       I2 : in std_logic;
       I3 : in std_logic;
       A : in std_logic;
       B : in std_logic;
       O : out std_logic);
end muxs;

architecture Behavioral of muxs is

begin
O <= I0 when (A ='0') and (B = '0')
else I1 when (A ='0') and (B = '1')
else I2 when (A ='1') and (B = '0')
else I3;
end Behavioral;
```

Esta asignación selectiva no es una expresión booleana. La asignación selectiva permite definir una secuencia de opciones, en la que prevalece un orden, es decir, que **I3** tenga el valor de **1** no significa que **O** resulte con **1**, sino que tiene que ocurrir que no se cumplan las condiciones previas a **I3** (las que se proponen para **I0**, **I1**, **I2**) para que **O** refleje a **I3**. Nótese que no se usan expresiones booleanas, sino relacionales con operadores de comparación.

©Editorial Digital Tecnológico de Monterrey

Otra definición equivalente de **O** bajo este mismo estilo es:

```
O <= ‘1’ when (A = '0') and (B = '0') and (I0 = ’1’)
else ‘1’ when (A = '0') and (B = '1') and (I1=’1’)
else ‘1’ when (A = '1') and (B = '0') and (I2 = ’1’)
```

Cabe aclarar que este tipo de asignación tiene una sintaxis especial, que es la siguiente:

```
Señal <=   valor1 when expresión_relacional1 else
           valor2 when expresión_ relacional2 else
           valor3 when expresión_ relacional3 else
           .....
           else valorn
```

Una expresión relacional es diferente a una expresión booleana. En una expresión relacional debe estar presente una comparación. Para construir una comparación se utilizan los siguientes operadores relacionales: =, >, <, >=, <=.

Recuerde que una expresión en VHDL se construye como si fuera evaluada de izquierda a derecha, por lo que la siguiente asignación está correctamente construida:

```
O <= I0 when (not A and not B=’1’)
else I1 when (not A and B=’1’)
```

Al igual que:

```
O <= I0 when (not A and not B) = ’1’ else I1 when (not A and
B) = ’1’
else I2 when (A and not B) = ’1’
```

©Editorial Digital Tecnológico de Monterrey

O bien,

```
O <= '1' when (not A and not B = '1') and (I0 = '1')
else '1' when (not A and B = '1') and (I1 = '1')
else '1' when(A and not B = '1') and (I2 = '1')
```

Después de haber observado estas opciones, sería posible concluir que la primera de esta clase que es la que está planteada de la manera más simple:

```
O <= I0 when (A = '0') and (B = '0')
else I1 when (A = '0') and (B = '1')
else I2 when (A = '1') and (B = '0')
```

A continuación, se muestran otras dos opciones con las cuales se describe el mismo *multiplexer* por su comportamiento:

```
architecture Behavioral of mux_comportamiento is
begin
process (A,B,I0,I1,I2,I3)
begin
if (A = '0') and (B = '0')  then O <= I0;
    elsif (A = '0') and (B = '1') then O <= I1;
    elsif (A = '1') and (B = '0') then O <= I2;
     elsif (A = '1') and (B = '1') then O <= I3;
     else null;
  end if;
  end process;
  end Behavioral;
```

Como habrá visto, esta forma de descripción utiliza **if**. En VHDL la descripción por *if* requiere de otra estructura llamada ***process***, entre paréntesis se encuentran las entradas al circuito descrito en esa estructura process. La descripción por *if* y *process* se explicará a detalle en el próximo capítulo.

©Editorial Digital Tecnológico de Monterrey

La siguiente descripción utiliza el estatuto ***case***, el cual reúne múltiples expresiones relacionales. Esta instrucción requiere establecer expresiones relacionales en las que intervenga solo una señal, es por eso que se está haciendo referencia a **Sel**, que no es una señal simple, sino un bus de dos *bits* que conjunta las dos señales de control **A**, **B**:

```
architecture Behavioral of caso is
begin
process (Sel,I0,I1,I2,I3)
begin
case Sel is
   when "00" => O <= I0;
   when "01" => O <= I1;
   when "10" => O <= I2;
   when "11" => O <= I3;
   when others => null;
end case;
end process;
end Behavioral;
```

Al final de este capítulo se presentará una explicación más detallada sobre la descripción de circuitos a través de procesos. En el capítulo 3 se explicará la definición y el uso en VHDL de buses de datos.

Actividad de repaso del tema 2.1

1. Diseñe en VHDL un decodificador de una entrada de control llamada A y dos salidas llamadas S0 y S1:

a) A nivel estructural.

b) A nivel ecuaciones.

©Editorial Digital Tecnológico de Monterrey

Antes de describir el circuito en VHDL conviene dibujar el diseño esquemático que para un decodificador con estas especificaciones es el que se muestra en la figura 2.6:

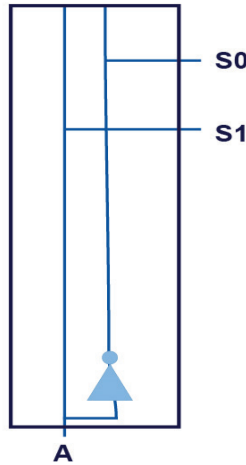


Figura 2.7 Diseño esquemático de un decodificador de una entrada de control

A continuación, se muestra una plantilla con espacios en blanco para completar la solución.

Para la definición estructural de este componente se requiere definir una compuerta not. A continuación, se presenta la definición de una compuerta notx con un espacio en blanco en la arquitectura para que el lector lo complete.

```
entity notx is
Port ( x : in std_logic;
       w : out std_logic);
end notx;
architecture ecuacion of notx is
begin
```

```
entity notx is
    Port ( x : in std_logic;
           w : out std_logic);
```

©Editorial Digital Tecnológico de Monterrey

```
end notx;
architecture ecuacion of notx is
begin
end ecuacion;
```

Enseguida se muestran las plantillas de la descripción estructural y de ecuaciones con espacios en blanco para ser completados con la descripción de sus arquitecturas:

a) A continuación, se presenta la plantilla para el diseño estructural de la descripción estructural de un decodificador con una entrada de control.

```
entity dec_struc is
Port ( a : in std_logic;
          s0 : out std_logic;
          s1 : out std_logic);
end dec_struc;
architecture estructural of dec_struc is
component notx is
port (x: in std_logic; w: out std_logic);
end component;
begin

end estructural;
```

b) La plantilla para el diseño por ecuaciones es la siguiente.

```
entity dec1_2ec is
Port ( a : in std_logic;
          s0 : out std_logic;
          s1 : out std_logic);
end dec1_2ec;
architecture ecuaciones of dec1_2ec is
begin

end ecuaciones;
```

©Editorial Digital Tecnológico de Monterrey

2. El siguiente ejercicio le propone diseñar en VHDL un decodificador de dos entradas de control A, B y cuatro salidas S0, S1, S2 y S3:

a) A nivel estructural.

b) A nivel ecuaciones.

c) A nivel comportamiento

El diseño esquemático con estas especificaciones es el que se observa en la figura 2.8:

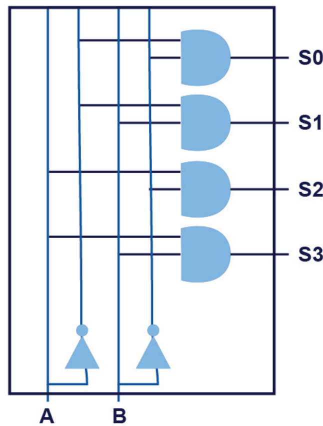


Figura 2.8 Diseño esquemático del decodificador con dos entradas de control

Enseguida se presentan las entidades, a manera de plantilla, para completar con la descripción de la arquitectura:

```
entity notx is
Port ( x : in std_logic;
       w : out std_logic);
end notx;
architecture ecuacion of notx is
begin

end ecuacion;
```

©Editorial Digital Tecnológico de Monterrey

Las plantillas de las compuertas que se van a requerir en la solución estructural de este circuito son las siguientes. Recuerde que hace falta especificar su arquitectura.

```
entity and2 is
Port ( x : in std_logic;
        y : in std_logic;
    z : out std_logic);
end and2;
architecture ecuacion of and2 is

begin

end ecuacion;
```

a) La plantilla para la definición estructural del decodificador de dos entradas selectoras es:

```
entity dec2_3estruc is
Port ( a : in std_logic;
    b : in std_logic;
    s0 : out std_logic;
    s1 : out std_logic;
    s2 : out std_logic;
    s3 : out std_logic);
end dec2_3estruc;
architecture struc of dec2_3estruc is
signal ac, bc: std_logic;
component and2 is Port ( x : in std_logic;
                    y : in std_logic;
                z : out std_logic);
end component;
component notx is port (x: in std_logic;
                    w: out std_logic);
end component;
begin

end struc;
```

©Editorial Digital Tecnológico de Monterrey

b) La plantilla para la definición por ecuaciones es:

```
 entity de2_3ecuac is
Port ( a : in std_logic;
          b : in std_logic;
          s0 : out std_logic;
          s1 : out std_logic;
          s2 : out std_logic;
          s3 : out std_logic);
 end de2_4ecuac;

 architecture ecuaciones of de2_3ecuac is

 begin

 end ecuaciones;
```

c) Para la solución por comportamiento utilice la asignación selectiva (con when), ya que todavía no se analizan las otras opciones.

```
 entity dec2_3comport is
Port ( a : in std_logic;
          b : in std_logic;
          s0 : out std_logic;
          s1 : out std_logic;
          s2 : out std_logic;
          s3 : out std_logic);
 end dec2_4comport;

 architecture Behavioral of dec2_3comport is begin

 end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

2.2 Circuitos combinacionales diseñados con procesos

Un proceso es una instrucción especial en VHDL, no encontrará estructuras semejantes en lenguajes de programación. Generalmente se utiliza para modelar un circuito por su comportamiento, ya que permite el uso de instrucciones condicionales.

Aunque la definición de circuitos combinacionales es más directa a partir de sus ecuaciones booleanas, también es posible utilizar ***ifs***. La restricción que tiene el uso del ***if*** es que este debe encontrarse dentro de un proceso. La condición que debe cumplir para el diseño de un circuito combinacional con un proceso es incluir a todas las señales de entrada en la lista llamada "lista sensible".

```
Process (lista sensible)
Begin
  Instrucción_secuencial;
  Instrucción_secuencial;
…
  Instrucción secuencial;
End process;
```

Las instrucciones secuenciales son: la asignación simple (no selectiva), el ***if*** y el ***case***. Las entradas al circuito son las que, al modificarse, pueden causar un cambio en las salidas del circuito. En contraste con las instrucciones secuenciales están las instrucciones concurrentes, que son: los procesos, las asignaciones simples, las asignaciones selectivas (*when*) y las referencias estructurales (*port map*).

©Editorial Digital Tecnológico de Monterrey

Ejemplos:

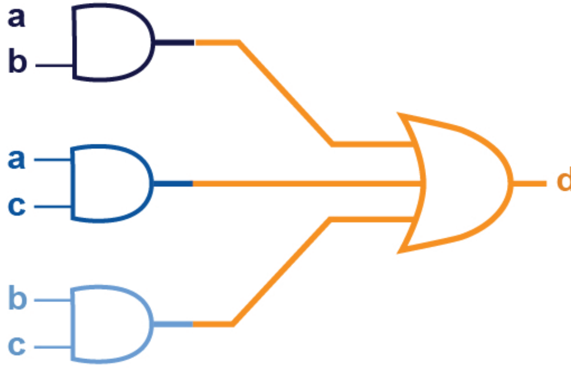


Figura 2.9 Ejemplo de circuito combinacional

Para describir este circuito hay varias alternativas, la que ya se presentó es:

```
d <= (a and b) or (a and c) or (b and c);
```

Otra alternativa de descripción utilizando un proceso es:

```
Process (a,b,c)
Begin
d <= (a and b) or (a and c) or (b and c);
end process;
```

Como cualquier alteración en **a**, **b** o **c** provocaría un posible cambio en la salida **d**, las tres señales deben incluirse en la lista sensible del process.

Las dos alternativas mostradas son equivalentes, generan el mismo circuito. Si usted se pregunta por qué utilizar un process para describir con una asignación simple, tiene razón en cuestionárselo:

©Editorial Digital Tecnológico de Monterrey

no tiene caso alguno, ya que el sistema de desarrollo reconoce las entradas en la ecuación d <= (a and b) or (a and c) or (b and c), así que no hay necesidad del proceso y su lista sensible.

Si desea describir el circuito a partir de su tabla de verdad, sería adecuado utilizar un process para describir las combinaciones por medio de condiciones.

```
Process (a,b,c)
Begin
  If (a='1') and (b='1') then d<='1';
          elsif (a='1') and (c='1') then d<='1';
          elsif (b='1') and (c='1') then d<='1';
          else d<='0';
   end if;
end process;
```

La construcción del *if* permite anidar condiciones (expresiones condicionales), utilizando la opción ***elsif condición***. El *if* debe terminar con las palabras reservadas **end if**. Si en vez de *if* … *elsif* se utilizan ifs independientes, cada *if* requerirá su propio end *if* como se muestra a continuación.

```
Process (a,b,c)
Begin
  If (a='1') and (b='1') then d<='1';
  else if (a='1') and (c='1') then d<='1';
       else if (b='1') and (c='1') then d<='1';
              else d<='0';
              end if;
        end if;
  end if;
end process;
```

Esta construcción es más laboriosa, no tiene caso usarla, es mejor utilizar ***elsif*** en vez de ifs independientes ya que producen el mismo circuito.

©Editorial Digital Tecnológico de Monterrey

Por otra parte, la lista sensitiva del proceso puede entenderse a partir de su simulación. El circuito descrito en el proceso es evaluado cada vez que una señal de la lista sensitiva tenga un cambio. Si alguna de las entradas al circuito no se especifica en la lista sensible, el circuito quedaría especificado erróneamente.

Se concluye la explicación del *if* con el siguiente diagrama de flujo (figura 2.9) en el que C1, C2 y C3 representan condiciones y S1, S2, hasta S8 simbolizan estatutos secuenciales:

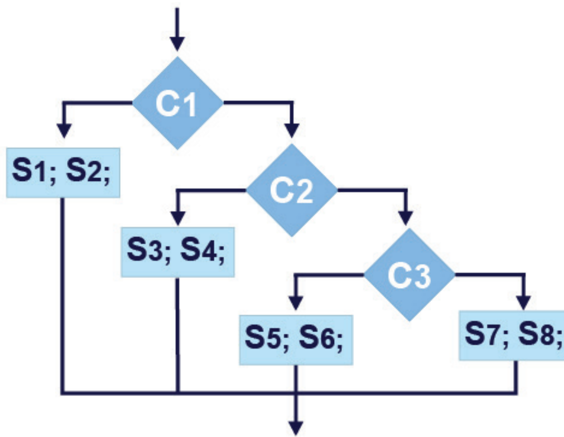


Figura 2.10 Estructura de Ifs anidados

Las dos formas principales de codificación son:

```
if C1 then S1; S2;                        if C1 then S1; S2;
else if C2 then S3; S4;                           elsif C2 then S3; S4;
             else if C3 then S5; S6;        elsif C3 then S5; S6;
                        else S7; S8;                 else S7; S8;
                    end if;                    end if;
     end if;
 end if;
```

©Editorial Digital Tecnológico de Monterrey

Actividad de repaso del tema 2.2

Codifique una arquitectura equivalente a la que se presenta a continuación, pero utilizando ecuaciones solamente (sin proceso).

```
architecture Behavioral of combi is
signal s1:std_logic;
begin
process (a,b,c, s1)
begin
if (a='1') and (b='1') then s1<='1';
                else s1<='0';
end if;
end process;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

2.3 Verificación del *if*

Convertir un código en un circuito usando VHDL no es nada fácil; para ello, se debe comprender el funcionamiento del sintetizador. Siempre recuerde que el eje de un *if* no lo constituyen las condiciones, sino la señal o señales de salida a las que se van a asociar expresiones de acuerdo a una condición.

Con los *ifs* anidados se construye la lógica similar a la de un *multiplexer*, las condiciones representan las señales de control. Para estar seguros de que un *if* está bien estructurado, conviértalo a asignaciones *when*, una por cada señal de salida, si no le es posible, es que algo anda mal en la estructuración del código en VHDL.

Actividad de repaso del tema 2.3

1. Indique qué resultados se reportarían al simular los siguientes circuitos con los datos proporcionados.

```
entity combi is
  Port ( a : in std_logic;
       b : in std_logic;
       c : in std_logic;
       d : out std_logic;
       e : out std_logic;
  end combi;

architecture Behavioral of combi is
```

©Editorial Digital Tecnológico de Monterrey

a)

```
force a 1
force b 1
force c 0
```

b)

```
force c 1
force a 1
force b 0
```

2. Corrija la arquitectura, conservando el proceso, para que represente un circuito combinacional correcto.

3. Codifique una arquitectura equivalente a la que hizo en el problema 2 pero utilizando ecuaciones solamente (sin proceso).

©Editorial Digital Tecnológico de Monterrey

2.4 Uso del case

El *case* tiene una implementación especial (en algunos casos no se realiza una buena síntesis del circuito). La opción **when others** no es un else, ya que solo contempla algunos casos. Si la señal que va a modelar puede caer en un caso no especificado, mejor modélela con if. Si tiene duda en su uso, no la use, use *if* o asignaciones selectivas con *when*.

En la siguiente opción no hay problema, porque el when others solo cubre un caso.

```
case A is
when "00" =>  …
when "01" => …
when "10" => …
when others => …
end case;
```

En el siguiente ejemplo sí puede haber una especificación incorrecta del circuito, es mejor convertirlo a if.

```
case B is
when "0000010" =>  …
when "0100001" => …
when "1000000" => …
when others => …
end case;
```

El uso intensivo del *if*, del *case* y de procesos se hará en descripciones en circuitos secuenciales, así que se volverán a revisar estas estructuras más adelante con mayor profundidad. Por lo pronto, para la descripción por comportamiento se sugiere el uso de la asignación selectiva. De igual forma, el manejo de buses de datos se deja para el capítulo 3.

©Editorial Digital Tecnológico de Monterrey

Hasta este punto se han analizado componentes comunes aislados, es decir, sin aplicaciones en dispositivos. En el capítulo 3 se estudiará la manera de describir circuitos aritméticos en diferentes niveles y se utilizarán circuitos aritméticos en dispositivos que consten de varios componentes, iniciando así el diseño de aplicaciones.

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 2

1. Diseñe los circuitos que encuentren el complemento a 2 de un número de tres *bits*. Recuerde que el complemento a 2 puede ser calculado sumando 1 al complemento a 1 del número, es decir: N * = N' + 1. Por ejemplo: 001 * = 111.

Encuentre las funciones lógicas y modélelas en VHDL en nivel de ecuaciones. Utilice el siguiente puerto:

```
entity compa2 is
Port ( N2,N1,N0 : in std_logic;
Nc2, Nc1, Nc0: out std_logic);
end compa2;
architecture behavioral of compa2 is
begin
end behavioral;
```

2. Indique qué resultados se reportarían al simular los siguientes circuitos.

```
entity combi is
Port ( a : in std_logic;
b : in std_logic;
c : in std_logic;
d : out std_logic;
e : out std_logic);
end combi;

architecture Behavioral of combi is
signal s1:std_logic;
begin
process(a,b)
begin
```

©Editorial Digital Tecnológico de Monterrey

```
s1 <=a and b;
e <= s1 or c;
d <= s1;
end process;
end Behavioral;
```

a) Los valores que se asocian a las señales al inicio de la simulación del circuito, utilizando comandos del simulador para especificarlos, son:

```
force a 1
force b 1
force c 0
```

b) Los valores para este inciso son:

```
force c 1
force a 1
force b 0
```

3. Corrija la siguiente arquitectura en VHDL, conservando el proceso, para que represente un circuito combinacional correcto.

```
architecture Behavioral of combi is
signal s1:std_logic;
begin
process(a,b,c)
begin
s1 <=a and b;
e <= s1 or c;
d <= s1;
end process;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

4. Codifique una arquitectura equivalente a la que se presenta a continuación, pero utilizando ecuaciones solamente (sin proceso).

```
architecture Behavioral of combi is
signal s1:std_logic;
begin
process (a,b,c, s1)
begin
s1 <=a and b;
e <= s1 or c;
d <= s1;
end process;
end Behavioral;
```

5. Diseñe en VHDL una sola expresión booleana que genere la salida F de un *multiplexer* 4 a 1, es decir, de dos entradas selectoras, A, B.

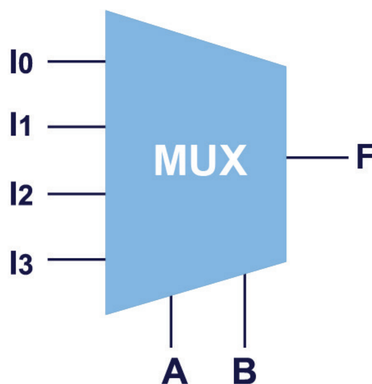


Figura 2.11 Multiplexer (o selector) de 4 a 1

©Editorial Digital Tecnológico de Monterrey

6. Modele el circuito *Demultiplexer* en VHDL (figura 2.12), a nivel de ecuaciones.

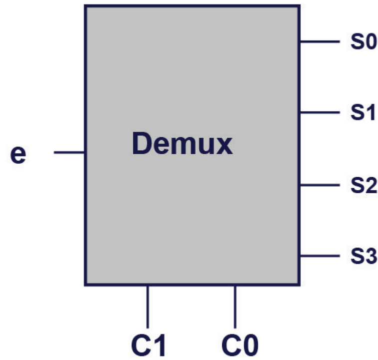


Figura 2.12 Bloque esquemático de un de*multiplexer*

Recuerde que, en un Demux, la salida que se genera es cuando se selecciona una de las salidas a través de C1 y C0. Utilice la siguiente plantilla para resolver este problema:

7. Se desea diseñar circuitos combinacionales que conviertan un número de 3 *bits* –que se encuentra en formato de signo magnitud– a uno que se encuentre en formato de complementos a 2. Considere que las entradas son: A2, A1,A0. Las salidas son: C2,C1,C0.

Para este problema simplifique el *bit* menos significativo de la salida. Nota: recuerde que los números positivos son iguales en todos los formatos.

8. A continuación, se especifica una unidad lógica (tabla 2.1), que cuenta con dos *bits* S1 y S0, que seleccionan la operación a realizar.

©Editorial Digital Tecnológico de Monterrey

Selectores S1 S0	Operación
0 0	R= A or B
0 1	R= A and B
1 0	R= not A
1 1	Sin uso

Tabla 2.1 Unidad lógica

Diseñe las expresiones lógicas de las salidas R. Tome en cuenta que:

- A y B son números de 3 *bits*.
- Las operaciones lógicas or and y not se aplican *bit* a *bit*.
- Las entradas al circuito son A2, A1, A0, B2, B1, B0, S1 y S0.

El circuito combinacional es el que se observa en la figura 2.13.

©Editorial Digital Tecnológico de Monterrey

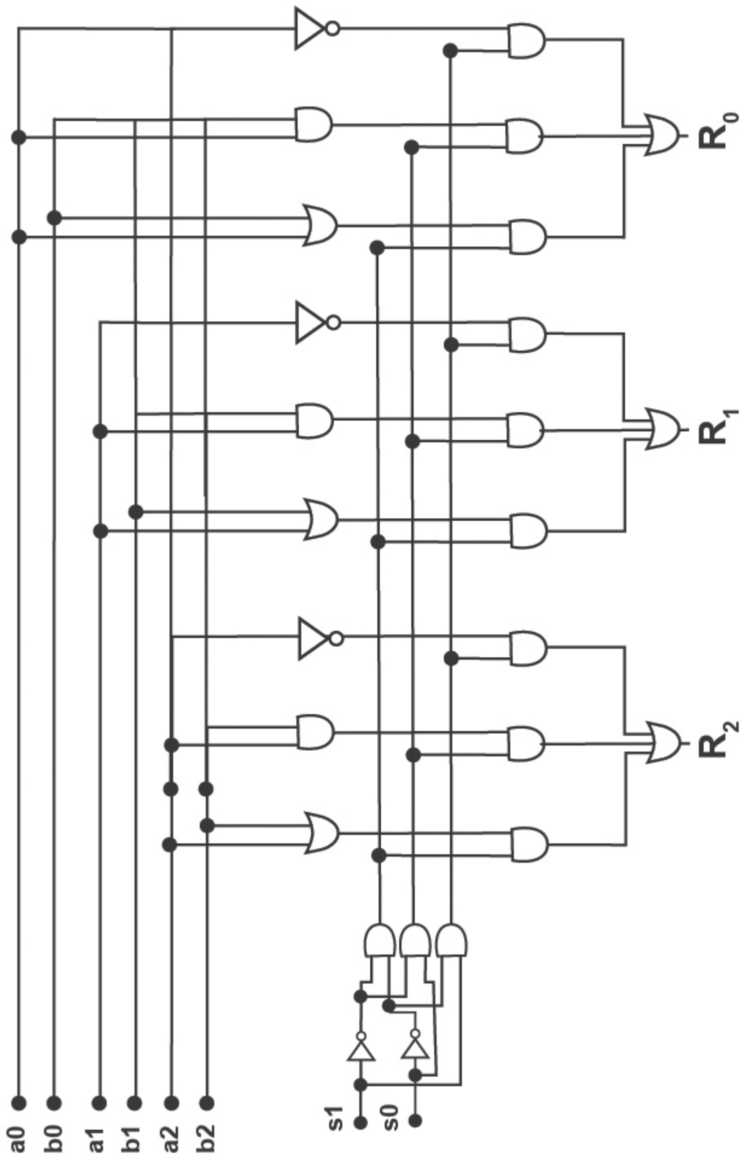


Figura 2.13 Circuito combinacional

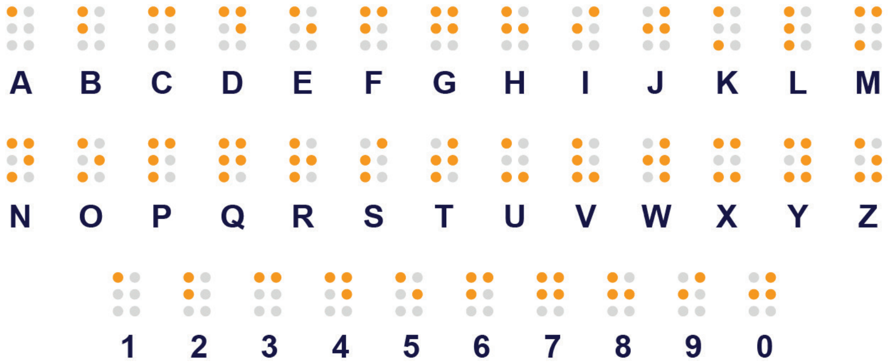
©Editorial Digital Tecnológico de Monterrey

9. Este problema consiste en diseñar los circuitos que se instalarán en una impresora Braille (alfabeto que usan para leer las personas que tienen problemas de visibilidad) que funciona para un subconjunto del alfabeto (para que el diseño no resulte tan extenso). Dicha impresora en vez de imprimir con tinta produce perforaciones para que las letras puedan ser reconocidas por el tacto. Una letra en Braille consta de seis puntos, en los que puede o no haber una perforación. Para realizar las perforaciones, la impresora cuenta con seis martillos, que operan todos al mismo tiempo. La impresora imprime de la A a la P mayúsculas (omitiendo la ñ y la ch).

El código estándar para las letras es el ASCII (American Standard Code for Information Interchange), pero requiere al menos ocho *bits*. Como la impresora Braille en cuestión solo imprime dieciséis letras distintas, para resolver este problema, se utilizará un código en binario para de cuatro *bits*. El código de la A es el 0000 y el de la P es 1111. La salida del circuito será el conjunto de unos y ceros que representan las señales que operarán los martillos. Con un 1 se realiza una perforación y con un 0 no se realiza. La entrada del circuito que se requiere diseñar son los cuatro *bits* del código de una letra y las salidas son las que se conectarán a los seis martillos de la impresora que perfora la letra en Braille.

Llame w, x, y z a los cuatro *bits* de la codificación de las letras, por ejemplo con w = 0, x = 0, y = 0,z = 1 se representa a la letra B. El alfabeto Braille es:

©Editorial Digital Tecnológico de Monterrey



A Punto 1

B Puntos 1 y 2

C Punto 1 y 4

D Puntos 1, 4 y 5

E Puntos 1 y 5

F Puntos 1, 2 y 4

G Puntos 1, 2, 4 y 5

H Puntos 1,2 y 5

I Puntos 2 y 4

J Puntos 2, 4 y 5

K Puntos 1 y 3

L Puntos 1, 2 y 3

M Puntos 1, 3 y 4

N Puntos 1, 3, 4 y 5

O Puntos 1, 3 y 5

P Puntos 1, 2, 3 y 4

©Editorial Digital Tecnológico de Monterrey

Para este problema diseñe únicamente la expresión booleana necesaria para accionar el martillo que perfore el punto 1, llámele P1.

10. Un equipo de científicos está diseñando un dispositivo para imprimir un código de barras en una credencial de plástico. Existen diferentes estándares para codificar un número en barras, el que están utilizando los científicos consiste en siete barras para cada número. Una barra puede ser negra, en cuyo caso habrá que imprimirla, o quedarse como un espacio. Dos barras impresas seguidas se observarán como una barra gruesa. Tres barras impresas seguidas se observarán como una barra más gruesa.

El grosor de las barras se aprecia en la figura 2.14:

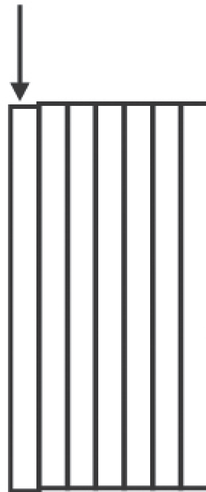


Figura 2.14. Esquema del código de barras y barra a modelar

El estándar que se usa para describir el código de un número indica un número de barras negras continuas, un guion para un espacio y en conjunto, barras negras y espacios, y debe sumar siete.

©Editorial Digital Tecnológico de Monterrey

Por ejemplo: el código del número 1 es: 1-2-1- y en barras se observaría como en la figura 2.15.

Figura 2.15 Representación del código para el número 1

El dispositivo que se está diseñando imprime un número en paralelo, es decir, hay siete micro “pinceles” que pintan cada barra de un número, o bien lo dejan en blanco dependiendo de su código. Los códigos de cada número son:

0: 1-2-2

1: 1-2-1-

2: 3-1-1

3: 2-2-1

4: -2-2-

5: -2-1-1

Para no alargar más el problema, consideraremos que los números solo existen del cero al cinco. El resto de las combinaciones no importan. Considere el resto de las combinaciones como que no importan. Modele los números con las variables a, b, c.

©Editorial Digital Tecnológico de Monterrey

Se le solicita diseñar la función más simple que accione al "pincel" que pinte la primera barra (señalada por una flecha), llámele B1. Hay un pincel por barra.

11. Se tienen cuatro interruptores I1 a I4 para seleccionar una de cuatro letras: E, I, O, U como se muestra en la figura 2.16. Se desea iluminar en un *display* de siete sementos como el que se muestra en la figura 2.17 la letra E cuando se cierre (se ponga en 1) el interruptor I1, se ilumine la letra I cuando se cierre el interruptor I2 y así sucesivamente. Solo son cuatro vocales las que hay que iluminar en mayúsculas (la I hacia la derecha, segmentos b, c).

Los interruptores se observan en la figura 2.16.

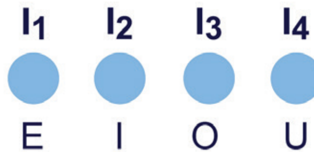


Figura 2.16 Interruptores para iluminar las letras E, I, O, U

©Editorial Digital Tecnológico de Monterrey

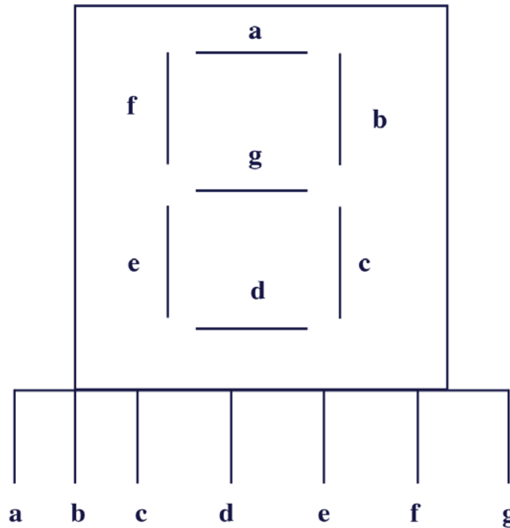


Figura 2.17 Display de siete segmentos

Si no se pone en 1 ningún interruptor, entonces el *display* permanecerá apagado, si se ponen en 1 dos o más interruptores, no importa la configuración que se ilumine, es decir, tome esas combinaciones como "no importan" (don't care). Diseñe las funciones necesarias para iluminar los segmentos.

Diseñe en VHDL solo la salida que se conectará al segmento b.

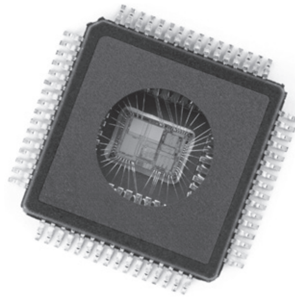
©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 2

Al término de este capítulo el lector habrá recordado componentes comunes, como los *multiplexer*s y los decodificadores, y conocerá las diferentes maneras que existen en VHDL para describir un circuito. Una manera, la de ecuaciones, está cercana al álgebra booleana, otra, la estructural, a la conexión de componentes y la de más alto nivel es la descripción funcional o de comportamiento.

No importa en qué nivel se describa un componente mientras se describa correctamente. No hay una manera mejor que otra ni que genere un mejor circuito. La cantidad de líneas de código no importan con respecto a la configuración del circuito.

La prueba de que un circuito está bien descrito es realizar su síntesis. Si el sistema infiere los componentes que se describieron, se ha hecho una codificación correcta.



©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 3. Circuitos para operaciones aritméticas y lógicas

Circuitos para operaciones aritméticas y lógicas

Diseño de un sumador de dos números de 4 bits	La resta
Diseño del sumador en VHDL en nivel de ecuaciones	Comparador
Buses (vectores) de datos	Ejemplos
Diseño del sumador en VHDL en nivel comportamiento	

Este capítulo 3 está dedicado a construir circuitos que realizan operaciones aritméticas y comparaciones. El capítulo finaliza con el diseño de dispositivos prácticos en los que se integran los circuitos aritméticos. Se exponen los diferentes tipos de descripciones en VHDL para cada uno de los circuitos de operaciones aritméticas.

©Editorial Digital Tecnológico de Monterrey

3.1 Diseño de un sumador de dos números de 4 *bits*

En esta sección se aborda el diseño de un sumador de dos números de cuatro *bits* cada uno. Para resolver este problema existen diversas opciones, la diferencia radica en el nivel de integración de los elementos utilizados. En VHDL esto se distingue por el nivel de descripción que se utilice.

Partiremos del diseño que utiliza componentes más básicos: compuertas lógicas. Para sumar dos números de cuatro *bits* sería posible construir una tabla de verdad para ocho variables booleanas independientes y cinco funciones booleanas, los cinco *bits* de respuesta. De la tabla resultarían 256 combinaciones y cinco funciones. De ahí se extraen los cinco circuitos que podrían requerir compuertas and de ocho entradas y compuertas or de cientos de entradas. Sería posible pensar que diseñar la suma para que ocurra en paralelo, es decir, a un mismo tiempo, llevaría a circuitos muy rápidos. Sin embargo, en la realidad, en los circuitos configurables no se cuenta con compuertas lógicas con un número tan extenso de entradas, así que para configurar los circuitos habría que segmentar las funciones y aumentar significativamente su número de niveles.

Recordará que el retraso de un circuito depende de su número de niveles; por esta razón se recurre al hecho de que el sistema numérico binario es posicional y esto permite sumar *bits* por posiciones, como se muestra en los siguientes ejemplos:

El primero de los ejemplos efectúa la suma por pasos, recordando que al sumar los dos *bits* de la posición 0 puede ocurrir: 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 10. Cuando la respuesta requiere dos *bits* se deja el *bit* menos significativo en la posición 0 y se agrega un sumando de 1 en la posición 1 (figura 3.1).

Figura 3.1 Suma binaria por posiciones hasta la posición 0

©Editorial Digital Tecnológico de Monterrey

Al sumar tres *bits*, los dos números de A y B en su posición i más el acarreo generado en la posición i-1, pueden ocurrir los siguientes casos: 0 + 0 + 0 = 0,0 + 0 + 1 = 1, 0 + 1 + 0 = 1, 1 + 0 + 0 = 1, todos estos casos requieren solo un *bit*, así que el acarreo que generan es de 0. En los siguientes casos se genera un acarreo de 1: 0 + 1 + 1 = 10, 1 + 0 + 1 = 10, 1 + 1 + 0 = 10, 1 + 1 + 1 = 11.

A continuación, se presenta la suma de las posiciones restantes (figura 3.2).

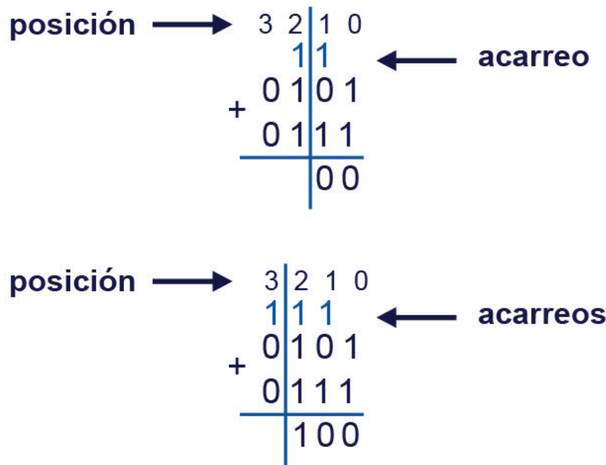


Figura 3.2 Suma binaria por posiciones hasta la posición 2

El resultado final, en el que el acarreo final forma parte de la respuesta (figura 3.3), es:

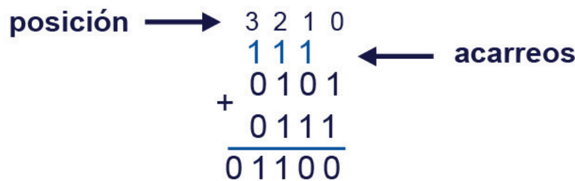


Figura 3.3 Suma binaria completa hecha por posiciones

©Editorial Digital Tecnológico de Monterrey

Generalizando, al sumar dos números binarios de cuatro *bits* se requiere sumar dos números que pueden ser modelados por cuatro variables booleanas cada uno.

$$\begin{array}{rcccc} & A_3 & A_2 & A_1 & A_0 \\ + & B_3 & B_2 & B_1 & B_0 \\ \hline \end{array}$$

La suma de ambos números ocurre de la siguiente manera:

$$\begin{array}{ccccc} & C_3 & C_2 & C_1 & \\ & A_3 & A_2 & A_1 & A_0 \\ + & B_3 & B_2 & B_1 & B_0 \\ \hline S_4 & S_3 & S_2 & S_1 & S_0 \end{array}$$

Se utiliza el enfoque de diseño en cascada que se refiere a utilizar módulos que producen salidas, que a su vez son entradas de un siguiente módulo.

Colocando las variables booleanas que intervienen en esta suma como entradas y salidas de los diversos módulos, el circuito se presenta de la siguiente manera (figura 3.4):

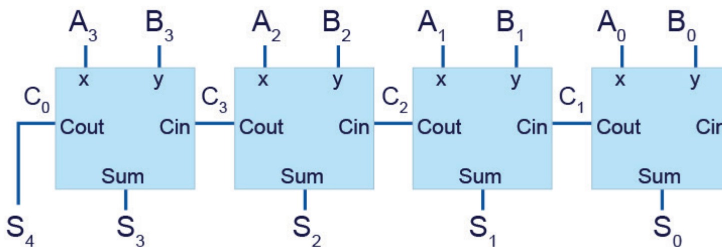


Figura 3.4 Diseño del sumador en cascada en módulos

©Editorial Digital Tecnológico de Monterrey

El circuito que corresponde al módulo de la posición 0 es un circuito comercial conocido como *Half Adder* (HA) o medio sumador. De la posición 1 a la más significativa (la 3) se requieren módulos que sumen tres *bits*, que también son circuitos que existen como circuitos integrados conocidos como *Full Adders* (FA) o sumadores completos (Figura 3.5).

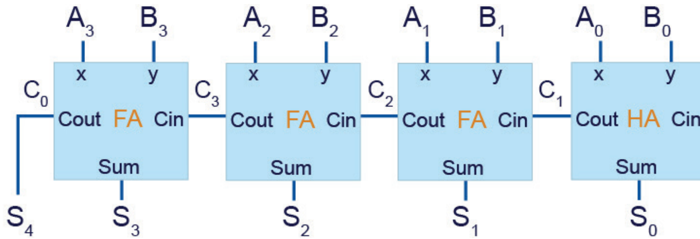


Figura 3.5 Diseño del sumador en cascada con *Full Adders*

Aunque este diseño tiene el retraso que ocasionan los acarreos, el diseño por tabla de verdad para realizar la suma en paralelo llevaría a circuitos con requerimientos de compuertas de múltiples entradas que conduce a un retraso equiparable. A este tipo de diseño se le llama circuito en cascada, porque la salida de una etapa del circuito se conecta a la siguiente.

Otra opción en este diseño es utilizar únicamente sumadores completos. En este caso la conexión es la que se muestra en la figura 3.6:

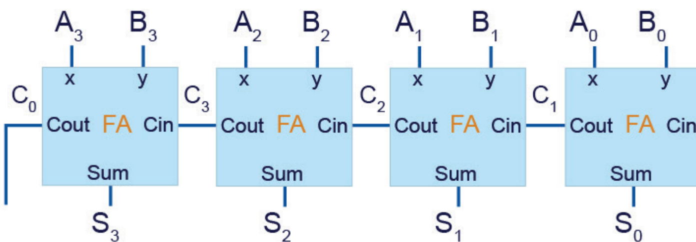


Figura 3.6 Diseño del sumador en cascada con *Full Adders*

Este diseño tiene la ventaja de poder representar a su vez un módulo que podría ser conectado a otros para sumar números de mayor magnitud.

©Editorial Digital Tecnológico de Monterrey

3.1.1 Diseño estructural del sumador de números de 4 bits

Para describir el circuito sumador compuesto por sumadores completos (*Full Adders*) es necesario describir, en primera instancia, el circuito del sumador completo con las ecuaciones derivadas de la tabla de verdad. Una vez hecho esto es posible interconectar cuatro sumadores completos para lograr el diseño en cascada.

Diseño del sumador completo

El diseño de uno de los módulos sumador completo se aprecia enseguida (figuras 3.7 y 3.8):

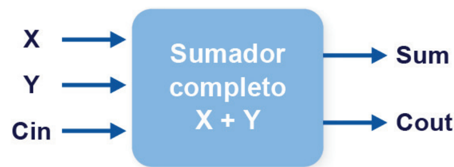


Figura 3.7 Diseño del *Full Adders*

X	Y	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figura 3.8 Tabla de verdad del *Full Adders*

©Editorial Digital Tecnológico de Monterrey

```
Cout = X'YCin + XY'Cin + XYCin' + XYCin = XY +
XCin + YCin
Sum = X'Y'Cin + X'YCin' + XY'Cin' + XYCin =
      = X'(Y'Cin + YCin') + X(Y'Cin' + YCin)
      = X ⊕ (Y ⊕ Cin)
```

Habrá que recordar que (Y'Cin + YCin')'= Y'Cin' + YCin (recuerde las propiedades del or exclusivo). Este diseño presentado en VHDL a nivel estructural es el siguiente:

```
entity FullAdder is
port (X, Y, Cin: in std_logic; -- Inputs
Cout, Sum: out std_logic); -- Outputs
end FullAdder;

architecture Equations of FullAdder is
begin
Sum <= X xor Y xor Cin;
Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end Equations;
```

Una vez definido el **component fulladder** es posible determinar el circuito del sumador de números de cuatro *bits* que se presenta a continuación. Note que los acarreos deben definirse como señales internas. Para que este circuito pueda conectarse con otros sumadores se le agregará la suma de un acarreo de entrada y en vez de dejar que la respuesta sea de 5 *bits*, se dejará de 4 más la producción de un acarreo de salida, de tal modo que el sumador quede con el siguiente puerto que se muestra en la figura 3.9.

©Editorial Digital Tecnológico de Monterrey

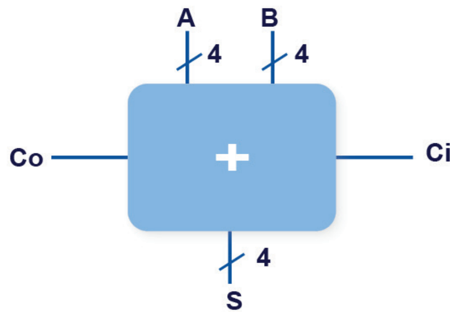


Figura 3.9 Bloque esquemático de sumador de dos números de cuatro *bits*

En VHDL resulta:

```
entity sumador4 is
port (A3, A2, A1, A0, B3, B2, B1, B0: in std_logic; Ci: in
std_logic;
S3,S2,S1,S0: out std_logic; Co: out std_logic);
end sumador4;

architecture Structure of sumador4 is
component FullAdder
port (X, Y, Cin: in std_logic;
Cout, Sum: out std_logic);
end component;
signal C1,C2,C3: std_logic;
begin
FA0: FullAdder port map (A0, B0, Ci, C1, S0);
FA1: FullAdder port map (A1, B1, C1, C2, S1);
FA2: FullAdder port map (A2, B2, C2, C3, S2);
FA3: FullAdder port map (A3, B3, C3, Co, S3);
end Structure;
```

Otra opción alterna para la descripción de este circuito utiliza el concepto de bus de datos, que consiste en dar un mismo nombre a un conjunto de *bits*. En VHDL, un bus se define como un vector de *bits*. Un *bit* específico dentro de un vector se referencia con paréntesis redondos y su posición dentro del vector. Esto se observa en la siguiente definición:

©Editorial Digital Tecnológico de Monterrey

```
entity sumador4 is
port (A, B: in std_logic_vector (3 downto 0); Ci: in std_log-
ic;
S: out std_logic_vector (3 downto 0); Co: out std_logic);
end sumador4;

architecture Structure of sumador4 is
component FullAdder
port (X, Y, Cin: in std_logic;
Cout, Sum: out std_logic);
end component;
signal C: std_logic _vector (3 downto 1);
begin --instantiate four copies of the FullAdder
FA0: FullAdder port map (A (0), B (0), Ci, C (1), S (0));
FA1: FullAdder port map (A (1), B (1), C (1), C (2), S (1));
FA2: FullAdder port map (A (2), B (2), C (2), C (3), S (2));
FA3: FullAdder port map (A (3), B (3), C (3), Co, S (3));
end Structure;
```

©Editorial Digital Tecnológico de Monterrey

3.2 Diseño del sumador en VHDL en el nivel de ecuaciones

El mismo diseño en cascada puede ser convertido en VHDL utilizando las ecuaciones del *Full Adder* en vez de hacer referencia a este componente. Esto se muestra en el siguiente código:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sumador4_ec is
port (A, B: in std_logic_vector(3 downto 0); Ci: in std_logic;
S: out std_logic_vector(3 downto 0); Co: out std_logic);
end sumador4_ec;

architecture Behavioral of sumador4_ec is
signal C: std_logic_vector (3 downto 1);
begin
S (0) <= A (0) xor B (0) xor Ci;
C (1) <= (A (0) and B (0)) or (A (0) and Ci) or (B (0) and
Ci);
S (1) <= A (1) xor B (1) xor C (1);
C (2) <= (A (1) and B (1)) or (A (1) and C (1)) or (B (1) and
C (1));
S (2) <= A (2) xor B (2) xor C (2);
C (3) <= (A (2) and B (2)) or (A (2) and C (2)) or (B (2) and
C (2));
S (3) <= A (3) xor B (3) xor C (3);
Co <= (A (3) and B (3)) or (A (3) and C (3)) or (B (3) and C
(3));
end Behavioral;
```

Antes de continuar con la definición del sumador a partir de su comportamiento, que es la descripción más sencilla y de mayor nivel en VHDL, abriremos un paréntesis para profundizar sobre la definición de vectores en VHDL.

©Editorial Digital Tecnológico de Monterrey

3.3 Buses (vectores) de datos

Un bus de *bits* se define indicando el subíndice que se utilizará para su *bit* más significativo, enseguida el subíndice que se utilizará para su *bit* menos significativo. Por ejemplo, A: in std_logic_vector (3 downto 0) define un bus que podría ser visualizado como se observa en la figura 3.10.

Figura 3.10 Vector A de cuatro *bits*

Un vector puede ser referenciado completo por su nombre, por ejemplo A, que equivale a indicar A (3 downto 0), o por partes, en cuyo caso se puede indicar un *bit*, como A (1), o varios, como A (2 downto 1) que hace referencia a los *bits* 2 y 1 del vector (figura 3.11).

Figura 3.11 Fragmento del vector A

Cuando un vector representa un número binario es apropiado que los índices representen las posiciones de sus *bits* (siempre finalizando con 0 para la posición menos significativa). Pero si el vector solo representa la asociación de un conjunto de *bits*, entonces cualquier grupo de subíndices es apropiado. Es posible definir algo como C: std_logic_vector (1 to 3) como se hizo para los tres acarreos del sumador. En este caso no hay una posición más o menos significativa, sino que solo se están agrupando tres *bits*: C (1), C (2) y C (3) bajo un mismo nombre.

©Editorial Digital Tecnológico de Monterrey

3.3.1 Asociación de valores o circuitos a vectores de datos

Es posible conectar (asociar o asignar) a un bus (vector) de *bits* de dos maneras:

a) Un bus (vector) de *bits* definido con el mismo tipo, siempre y cuando sea exactamente del mismo tamaño, es decir, mismo número de *bits*. En este caso solo se está renombrando al vector.

b) La salida de un circuito que efectúe operaciones, siempre y cuando la operación produzca exactamente el mismo número de *bits* correspondiente al bus al que se conectará (asignará).

Por ejemplo, dada la siguiente definición de puerto:

```
Port (  A : in std_logic_vector (3 downto 0);
          b : out std_logic_vector (3 downto 0);
           c : in std_logic_vector (4 downto 0);
           D, e : out std_logic_vector (4 downto 0);
           f : out std_logic);
```

Es posible observar que algunas de las siguientes conexiones son correctas y otras incorrectas:

b <=c es incorrecta, porque b y c son vectores de diferentes tamaños.

D <=A es incorrecta, porque d y a son vectores de diferentes tamaños.

©Editorial Digital Tecnológico de Monterrey

b <= A es correcta.

b <=c (4 downto 1) es correcta.

Concatenación

La concatenación de vectores o de porciones de vectores se logra con el operador **&**. Por ejemplo, **D <= A & A(0)**, causa que el *bit* 4 de D sea el *bit* 3 de A, como se ilustra en la figura 3.12.

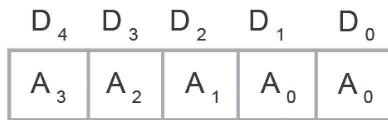


Figura 3.12 Concatenación y asociación

Otras concatenaciones válidas son:

e <=a(2 downto 0)&"10"; b <=a(3)&c(2 downto 0).

3.3.2 Suma descrita en forma funcional

Es válido definir un circuito sumador, un restador o un multiplicador a nivel funcional, que hemos manejado como nivel comportamiento. Para este propósito se utilizan los operadores aritméticos +, − y *. En este subtema, solo se abordará la suma.

Los operandos válidos para la suma son:

a) Vectores o porciones de vectores que correspondan a la longitud del bus de salida (al que serán asignados), como **b <= a + a**, o bien, **b <= a + a + a** (o cualquier número de operandos, no tienen que ser dos o tres) aunque la respuesta se genera por el tamaño de los operandos, perdiendo todos los acarreos.

©Editorial Digital Tecnológico de Monterrey

b) Vectores o porciones de vectores que correspondan a la longitud del bus de salida a los que se le sume un *carry* entero, cuya conversión a binario ocupe a lo más, el número de *bits* que tenga el otro operando, como **b <= a + 1**. El acarreo que se suma puede ser una constante entera, un *bit* o bien una señal del puerto o interna, como **b <= a + Cin** en el que **Cin** puede ser un *carry* de entrada.

c) Vectores o porciones de vectores que correspondan a la longitud del bus de salida a los que se le sumen números binarios, expresados en cadenas de *bits* acotados por los símbolos **" "**, de tal manera que la cadena no exceda la longitud del bus de salida, como **b <= a + "11".**

Cabe mencionar que la señal de salida correspondiente al acarreo final en la suma no es producida por el circuito.

©Editorial Digital Tecnológico de Monterrey

3.4 Diseño del sumador en VHDL en el nivel de comportamiento

En el nivel comportamiento, el sumador de cuatro *bits* queda descrito como se muestra enseguida –aunque en esta primera versión resulta faltante el *carry* de salida.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity sumador4_ec is
port (A, B: in std_logic _vector (3 downto 0); Ci: in std_logic; -- Entradas
S: out std_logic _vector (3 downto 0); Co:out std_logic;); -- Salidas
end sumador4_ec;

architecture Behavioral of sumador4_ec is
begin
S <= A + B + Ci;
--Co <=    no es posible calcular Co, es decir, el carry de salida
end Behavioral;
```

Si se requiere el **acarreo de salida**, se pueden extender a 5 *bits* los vectores de entrada, para que el acarreo quede integrado a la respuesta y así realizar la suma en un vector auxiliar, el acarreo final quedará en la posición 5 del resultado. El cálculo se describe como sigue (ver figura 3.13).

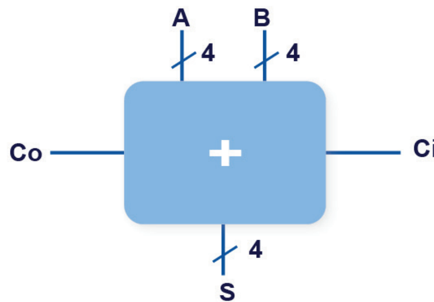


Figura 3.13

©Editorial Digital Tecnológico de Monterrey

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sumador4_com is
port (A, B: in std_logic _vector(3 downto 0); Ci: in std_logic;
-- Inputs
S: out std_logic _vector(3 downto 0); Co:out std_logic;); --
Outputs
end sumador4_com;  --el puerto utilizado es el mismo que en las
descripciones anteriores

architecture Behavioral of sumador4_com is
signal Aux1, Aux2: std_logic;
signal S_aux: std_logic _vector(4 downto 0);
begin
Aux1 <= '0'&A;
Aux2 <= '0'&B;
S_aux <= Aux1+Aux2+Ci;
S <= S_aux(3 downto 0);
Co <=  S_aux(4);
end Behavioral;

o bien,
architecture Behavioral of sumador4_com is
signal S_aux: std_logic _vector(4 downto 0);
begin
S_aux <= ('0'&A) + ('0'&B) +Ci;
Nota: Los paréntesis son necesarios. Recuerde que no hay
prioridades en las operaciones.
S <= S_aux(3 downto 0);
Co <= S_aux(4);
end Behavioral;
```

Esta última descripción produce un circuito funcionalmente equivalente al descrito en forma estructural o de ecuaciones. Cuando el sistema de desarrollo reconoce el operador funcional de suma (+), el circuito lo implementa con *Full Adders*, los cuales generalmente ya se encuentran en el FPGA. De otro modo, en la descripción estructural o de ecuaciones, los *Full Adders* los construye con los circuitos lógicos descritos para estos. Obtendremos un mejor circuito, con menos retraso si se conectan directos los FA, que si se construyen con la lógica combinacional de estos.

Se recomienda entonces describir la suma con el operador funcional + y dejar que el sistema produzca el circuito óptimo.

©Editorial Digital Tecnológico de Monterrey

3.5 La resta

Como la suma, también la resta puede ser descrita en diferentes niveles. En este tema se presenta un restador construido por módulos que efectúan una resta directa de números binarios sin signo. Luego se detalla la descripción funcional.

3.5.1 Restador construido con restadores conectados en cascada

Esta sección muestra el diseño de un restador que efectúa una resta directa –como si se hubiera hecho a mano– de dos números a y b con un enfoque de resta en cascada. La resta solo es correcta cuando **a >= b**, en caso contrario, el resultado será incorrecto (intente restar directamente 5 – 7).

Veamos el diseño esquemático en cascada para números de **n + 1** *bits* (figura 3.14).

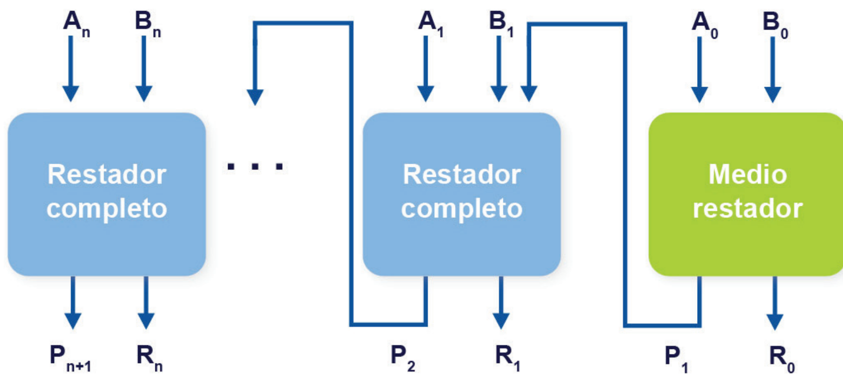


Figura 3.14 Diseño esquemático en cascada

El diseño detallado de cada elemento es:

- **Medio restador.** En estas combinaciones se presenta el caso, por ejemplo, de 0 −1 = 1, con 1 que se acarrea para la siguiente posición. El *bit* que se acarrea se resta a A_i o se suma a B_i (figura 3.15).

©Editorial Digital Tecnológico de Monterrey

A_0	B_0	P_1	R_0
0	0	0	0
0	1	1	1
0	0	0	1
1	1	0	0

Figura 3.15 Medio restador

$P_1 = A_0' B_0$

$R_0 = A_0$ Xor B^0

- **Restador completo.** Este módulo considera la entrada de un acarreo y la producción de un acarreo. Las combinaciones son las que se aprecian en la figura 3.16.

A_i	B_i	P_i	P_{i+1}	R_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Figura 3.16 Restador completo

La función del *bit* de resta resulta similar a la de la suma.

$R_i = A_i$ Xor B_i Xor P_i

$P_{i+1} = B_i P_i + A_i' P_i + A_i' B_i$

©Editorial Digital Tecnológico de Monterrey

3.5.2 Diseño de un restador a partir de restadores completos para números de 4 bits sin signo

A continuación, se presenta la codificación en VHDL de este restador.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity restadorcompleto is
    Port ( x : in std_logic;
           y : in std_logic;
           pin : in std_logic;
      R : out std_logic;
           pout : out std_logic);
end restadorcompleto;

architecture ecuaciones of restadorcompleto is
begin
R<= x xor y xor pin ;
pout<= (not x and pin) or (not x and y) or (y and pbin);
end ecuaciones;

entity subtracter4 is
    Port ( a : in std_logic_vector(3 downto 0);
           b : in std_logic_vector(3 downto 0);
                 r: out std_logic_vector(3 downto 0));
end subtracter4;

architecture estructural of subtracter4 is
component restadorcompleto
      Port ( x : in std_logic;
                    y : in std_logic;
                 pin : in std_logic;
             R : out std_logic;
                pout : out std_logic);
end component;
```

©Editorial Digital Tecnológico de Monterrey

```
signal P :std_logic_vector(4 downto 1);
begin
ro: restadorcompleto port map (a(0), b(0), '0',
r(0), P(1));
r1: restadorcompleto port map (a(1), b(1), P(1),
r(1), P(2));
r2: restadorcompleto port map (a(2), b(2), P(2),
r(2), P(3));
r3: restadorcompleto port map (a(3), b(3), P(3),
r(3), P(4));
end estructural;
```

Note que se omitió el componente medio restador y en su lugar se utilizó un restador completo con un **borrow-in** de **0**. Si **P(4)** queda con un valor de 1 indica que la respuesta es incorrecta por ser **a<b**.

3.5.3 Diseño del restador a nivel comportamiento

La resta a nivel funcional (o de comportamiento) se define con el operador aritmético de la resta (–). Las reglas para la conformación de los operandos de la suma también aplican para la resta (constantes y señales). La resta que se genera cuando se describe a nivel funcional es un circuito sumador que suma la conversión a complementos a 2 del operando que restará. Si se desea calcular **A – B**, el circuito que se generará es **A + B' + 1**, ya que el complemento a 2 de **B** se calcula con la suma del complemento a **1** (la negación de cada ***bit*** del operando) de **B + 1**.

Por lo pronto, se proporcionan ejemplos de los resultados que se generan, tanto en la simulación como en la síntesis de un circuito restador.

A continuación, se presentan dos ejemplos en los que se restan números de cuatro *bits* sin signo:

©Editorial Digital Tecnológico de Monterrey

1) R<= “1001”-“0011”.

Internamente la operación que se realiza es: **1001+1100+1**.

El resultado que se obtiene de esta resta es **“0110”** que es un resultado positivo correcto.

2) R<= “1001”-“1010”.

Internamente la operación que se realiza es: **1001+0101+1**.

El resultado que se obtiene de esta resta es **“1111”** que es un resultado que representa el complemento a 2 de 1, ya que el resultado es -1.

A nivel comportamiento, el restador queda definido como sigue (ver figura 3.17).

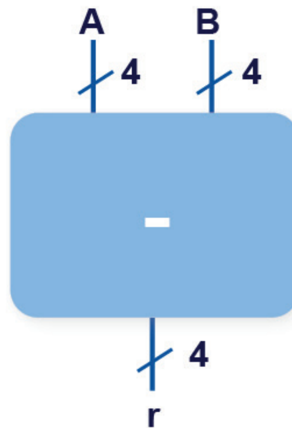


Figura 3.17 Restador de número de cuatro *bits*

©Editorial Digital Tecnológico de Monterrey

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity restador4_com is
    Port ( a : in std_logic_vector(3 downto 0);
           b : in std_logic_vector(3 downto 0);
         res : out std_logic_vector(3 downto 0));
end restador4_com;

architecture Behavioral of restador4_com is
begin
res <= a - b;
end Behavioral;
```

En esta definición, la respuesta de la resta queda en un formato de complementos a **2**. Así que si **a<b**, el ***bit* 3** de **r** es **1**, reflejando que la respuesta es negativa. Sin embargo, si los números que se encuentran en **a** y en **b** no tienen signo, entonces no será posible distinguir si la respuesta es o no correcta, porque podría haber resultados positivos que utilizan el ***bit* 3** de **r**, como, por ejemplo: **1111 − 0111 = 1000 15 − 7 = 8**.

El manejo del signo requiere de algún truco aritmético, por ejemplo, efectuar una comparación. La comparación es una operación necesaria para un sinnúmero de aplicaciones, su circuito se analiza en la siguiente sección.

©Editorial Digital Tecnológico de Monterrey

3.6 Comparador

A continuación, un comparador de dos números **A, B** sin signo de cuatro *bits* cada uno. El comparador debe generar dos respuestas **M** (mayor) e **I** (igual). **M=1** si y solo si **A > B, I = 1** si y solo si **A = B**. Este comparador está construido a partir de módulos conectados en cascada, utiliza funciones para comparar dos *bits* y generar dos salidas, una que detecte la condición "mayor que" y la otra "igual que". Estas salidas se utilizan como dos acarreos, uno para indicar que el número ya es mayor y otro para indicar que va igual; si ya se determinó que es menor, ambos acarreos son **0**.

La comparación debe realizarse a partir de los *bits* más significativos, como si **A** y **B** se fueran a ordenar alfabéticamente. La lógica se explica a continuación:

Digamos que, si **A** está compuesto por $\mathbf{a_3}$, $\mathbf{a_2}$, $\mathbf{a_1}$ y $\mathbf{a_0}$ y **B** por $\mathbf{b_3}$, $\mathbf{b_2}$, $\mathbf{b_1}$ y $\mathbf{b_0}$, el procedimiento es el siguiente:

©Editorial Digital Tecnológico de Monterrey

1. Si $\mathbf{a_3}$ es mayor a $\mathbf{b_3}$ (1, 0) entonces {el número **A** es mayor que el número **B**, se genera $\mathbf{M_3=1}$ e $\mathbf{I_3=0}$. Ir al paso 4.}

2. Si $\mathbf{a_3}$ es igual a $\mathbf{b_3}$ entonces {se debe seguir comparando ya que aún no se puede determinar si **A** es mayor, menor o igual a **B**; se genera $\mathbf{M_3=0}$ e $\mathbf{I_3=1}$. Ir al paso 4.}

3. La condición que queda es $\mathbf{a_3}$ menor $\mathbf{b_3}$, esto ya es seguro en este paso, entonces se genera $\mathbf{M_3 = 0}$ e $\mathbf{I_3 = 0}$ que indica que **A** es menor que **B**.

4. Si $\mathbf{M_3 = 1}$ entonces {$\mathbf{M_2 = 1}$ e $\mathbf{I_3 = 0}$. Ir al paso 9.}

5. Si $\mathbf{M_3 = 0}$ e $\mathbf{I_3 = 0}$ entonces {$\mathbf{M_2 = 0}$ e $\mathbf{I_2 = 0}$ ir al paso 9.}

6. Si $\mathbf{I_3 = 1}$ entonces {si $\mathbf{a_2}$ es mayor a $\mathbf{b_2}$ entonces {el número **A** es mayor que el número **B**, se genera $\mathbf{M_2 = 1}$ e $\mathbf{I_2 = 0}$. Ir al paso 9}}

7. Si $\mathbf{a_2}$ es igual a $\mathbf{b_2}$ entonces {se debe seguir comparando ya que aún no puede determinarse si **A** es mayor, menor o igual a **B**; se genera $\mathbf{M_2 = 0}$ e $\mathbf{I_2 = 1}$. Ir al paso 9.}

8. La condición que queda es $\mathbf{a_2}$ menor $\mathbf{b_2}$, esto ya es seguro en este paso, entonces se genera $\mathbf{M_2 = 0}$ e $\mathbf{I_2 = 0}$ que indica que la condición es **A** es menor que **B**.

9. Repetir los pasos 4 al 8, pero ahora para $\mathbf{a_1}$ y $\mathbf{b_1}$.

10. Finalmente para $\mathbf{a_0}$ y $\mathbf{b_0}$. Los resultados de esta última comparación, $\mathbf{M_0}$ e $\mathbf{I_0}$ constituyen los resultados finales, **M** (mayor) e **I** (igual).

Generalizando:

$\mathbf{I_i}$ indica si hasta el momento **a = b**, es decir $\mathbf{I_i = 1}$ si $\mathbf{a_n}$, $\mathbf{a_{n-1}}$, … $\mathbf{a_i = b_n}$, $\mathbf{b_{n-1}}$, … $\mathbf{b_i}$.

$\mathbf{M_i}$ indica si hasta el momento **a>b**, es decir $\mathbf{M_i = 1}$ si $\mathbf{a_n}$, $\mathbf{a_{n-1}}$, … $\mathbf{a_i > b_n}$, $\mathbf{b_{n-1}}$, … $\mathbf{b_i}$ una vez que **a>b** esta condición se conserva.

Si $\mathbf{I_i=M_i=0}$, quiere decir que **a<b** y esta condición se conserva.

©Editorial Digital Tecnológico de Monterrey

El diseño esquemático es el que se observa en la figura 3.18, ejemplificándolo para números de 16 *bits*.

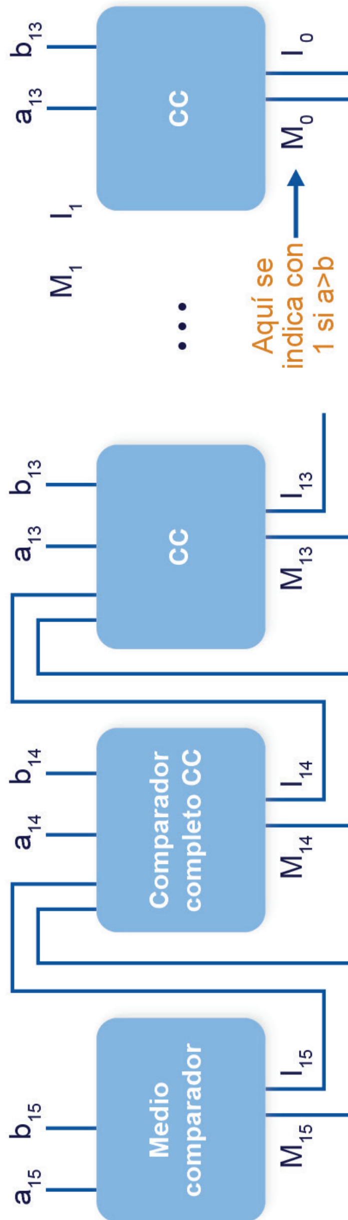


Figura 3.18 Diseño esquemático en cascada

©Editorial Digital Tecnológico de Monterrey

Para comparar los *bits* $\mathbf{a_n}$ $\mathbf{b_n}$ se requiere un medio comparador, ya que es la posición más significativa y no requiere un acarreo de entrada (ver tabla 3.1).

a_n	b_n	M_n	I_n
0	0	0	1
0	1	0	0
1	0	1	0
1	1	1	1

Tabla 3.1 Tabla de verdad del medio comparador

Para el resto de las posiciones se requiere un comparador completo, con la finalidad de que ingresen los acarreos de entrada y se generen los de salida. Observe la tabla 3.2.

A_{n-i}	B_{n-i}	M_{n-i+1}	I_{n-i+1}	M_{n-i}	I_{n-i}
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	X	X
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	1	0
0	1	1	1	X	X
1	0	0	0	0	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	X	X
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	X	X

Tabla 3.2

©Editorial Digital Tecnológico de Monterrey

Simplificando estas funciones resulta:

$$M_{n-i} = M_{n-i+1} + I_{n-i}A_{n-i}(B_{n-i})'$$
$$I_{n-i} = I_{n-i+1}\,(A_{n-i}\text{ Xnor }B_{n-i})$$

A continuación, se muestra el código en VHDL en nivel estructural de un comparador de números de 4 *bits*. En este código solo se han utilizado comparadores completos. Se sustituye el medio comparador con un comparador completo al que se ha conectado un **1** al acarreo **I** de entrada, como si lo que en posiciones más significativas tuviera la condición de "igual", y un 0 a la entrada **M**.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity comparadorcompleto is
    Port ( Ii : in bit;
           Mi : in bit;
           x : in bit;
           y : in bit;
           Io : out bit;
           Mo : out bit);
end comparadorcompleto;

architecture ecuaciones of comparadorcompleto is
begin
Io <= Ii and not (x xor y);
Mo <= Mi or (Ii and x and not y);
end ecuaciones;

entity comparador4 is
    Port ( a : in bit_vector(3 downto 0);
           b : in bit_vector(3 downto 0);
           igual : out bit;
           mayor : out bit);
end comparador4;
```

©Editorial Digital Tecnológico de Monterrey

```
architecture behavioral of comparador4 is
component comparadorcompleto is
   Port ( Ii : in bit;
           Mi  : in bit;
           x : in bit;
           y : in bit;
           Io : out bit;
           Mo : out bit);
     end component;
signal i,m:bit_vector (3 downto 1);
begin
fc3: comparadorcompleto port map
('1','0',a(3),b(3),i(3),m(3));
fc2: comparadorcompleto port map
(i(3),m(3),a(2),b(2),i(2),m(2));
fc1: comparadorcompleto port map
(i(2),m(2),a(1),b(1),i(1),m(1));
fc0: comparadorcompleto port map
(i(1),m(1),a(0),b(0),igual,mayor);
end behavioral;
```

El mismo diseño, pero a nivel de ecuaciones, sustituyendo la referencia a comparador completo por sus ecuaciones booleanas, queda descrito en VHDL a continuación:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity comparador4_ec is
    Port ( a : in bit_vector(3 downto 0);
           b : in bit_vector(3 downto 0);
           igual  : out bit;
           mayor : out bit);
end comparador4_ec;
```

©Editorial Digital Tecnológico de Monterrey

```
architecture ecuaciones of comparador4_ec is
signal I,M:bit_vector (3 downto 1);
begin
I(3) <= a(3) xnor b(3);
M(3) <= a(3) and not b(3);
I(2) <= I(3) and not (a(2) xor b(2));
M(2) <= M(3) or (I(2) and a(2) and not b(2));
I(1) <= I(2) and not (a(1) xor b(1));
M(1) <= M(2) or (I(2) and a(1) and not b(1));
igual <= I(1) and not (a(0) xor b(0));
mayor <= M(1) or (I(1) and a(0) and not b(0));
end ecuaciones;
```

Ya que se ha explicado el circuito interno del comparador, se muestra la definición a nivel comportamiento de este circuito. Como se observa, la definición es muy simple, lo que no debe perder de vista es el circuito en el que un comparador se convierte.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity comparador4_comp is
    Port ( a : in bit_vector(3 downto 0);
           b : in bit_vector(3 downto 0);
           igual : out bit;
           mayor : out bit);
end comparador4_comp;

architecture comportamiento of comparador4_comp
is
begin
igual <= '1' when (a = b) else '0';
mayor <= '1' when (a > b) else'0';
end comportamiento;
```

©Editorial Digital Tecnológico de Monterrey

Así como se define la comparación por "mayor que", es posible definir otras comparaciones:

Operador en VHDL	Operación
=	igual que
>	mayor que
<	menor que
>=	mayor o igual que
<=	menor o igual que
/=	diferente que

Tabla 3.3

©Editorial Digital Tecnológico de Monterrey

3.7 Ejemplos

A continuación, se presentan tres ejemplos relacionados con los circuitos estudiados en este capítulo.

3.7.1 Restador con la técnica de complementos a 1

Una manera de realizar una resta **A – B** es utilizar la técnica de complementos a 1 que consiste en sumar el complemento a **1** de **B** y sumarle el ***carry out*** que se produzca (las entradas no se alteran, por eso el circuito se estabiliza). Si el resultado es positivo, será correcto, si es negativo, quedará en complemento a **1**.

El circuito basado en *Full Adder* es el que se observa en la figura 3.19.

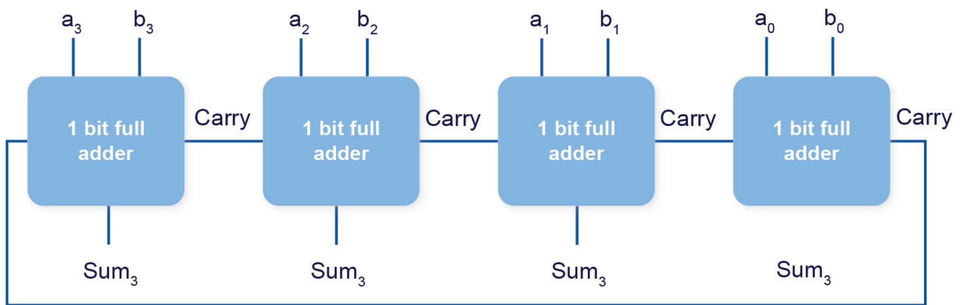


Figura 3.19 Restados que utiliza técnica de complementos a 1

Considerando que ya se cuenta con la definición de un *Full Adder*, diseñe este restador en VHDL a nivel estructural.

©Editorial Digital Tecnológico de Monterrey

```
entity FullAdder is
port (X, Y, Cin: in std_logic; -- Inputs
Cout, Sum: out std_logic); -- Outputs
end FullAdder;

architecture Equations of FullAdder is
begin -- Concurrent Assignments
Sum <= X xor Y xor Cin;
Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end Equations;
```

Siguiendo el diagrama esquemático, la arquitectura de este restador es:

```
entity restador4 is
port (A, B: in std_logic_vector(3 downto 0); Ci: in
std_logic; -- Inputs
S: out std_logic_vector(3 downto 0)); -- Output
end restador4;

architecture Structure of restador4 is

component FullAdder
port (X, Y, Cin: in std_logic; -- Inputs
Cout, Sum: out std_logic); -- Outputs
end component;

signal comp1 : std_logic_vector(3 downto 0);
     signal Co0 : std_logic;
     signal Co1 : std_logic;
     signal Co2 : std_logic;
     signal Co3 : std_logic;

begin
     comp1 <= not B;
     s0: FullAdder port map(A(0), comp1(0), Co3, Co0, S(0));
     s1: FullAdder port map(A(1), comp1(1), Co0, Co1, S(1));
     s2: FullAdder port map(A(2), comp1(2), Co1, Co2, S(2));
     s3: FullAdder port map(A(3), comp1(3), Co2, Co3, S(3));
end Structure;
```

©Editorial Digital Tecnológico de Monterrey

3.7.2 Circuito oculto

En el siguiente código se encuentra un circuito. Para descubrir la operación que realiza conviene identificar el circuito en cascada que representa.

```
entity operador is
 Port ( x : in std_logic;
           bin : in std_logic;
R : out std_logic;
           bout : out std_logic);
end operador;

architecture ecuaciones of operador is
begin
R <= x xor bin ;
bout <= x and bin;
end ecuaciones;

library IEEE;…
entity circuito is
    Port ( a : in std_logic_vector(3 downto 0);
           s : out std_logic_vector(4 downto 0));
end circuito;

architecture estructural of circuito is
component operador is
    Port ( x : in std_logic;
           bin : in std_logic;
R : out std_logic;
           bout : out std_logic);
end component;
signal c : std_logic_vector (3 downto 1);
begin
r3: operador port map (a(3), c(3), s(3), s(4));
r2: operador port map (a(2), c(2), s(2), c(3));
r1: operador port map (a(1), c(1), s(1), c(2));
ro: operador port map (a(0), '1',  s(0), c(1));
end estructural; *
```

©Editorial Digital Tecnológico de Monterrey

Siguiendo las conexiones, el circuito es el que se observa en la figura 3.20.

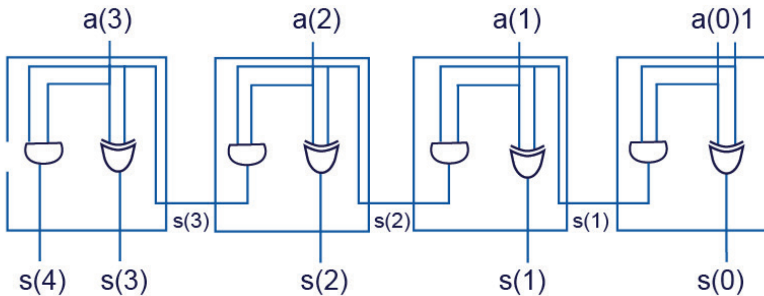


Figura 3.20 Expansión del circuito

En este caso es más sencillo determinar el funcionamiento del circuito observando el diseño esquemático que analizando únicamente el código, dado que el orden de la descripción es confuso.

Al simular este circuito, si se asocia la combinación **0111** a la señal **a** en un tiempo **= 0**, los valores que toma **s (4 downto 0)** en los tiempos: **0**, **delta**, **2 deltas**, **3 deltas**, **4 deltas**, **5 deltas** son:

tiempo=0 s=uuuuu,

tiempo=delta s=0uuu0, (como a (3) es 0 s (4) se determina en el primer delta de tiempo) como se muestra en el esquemático (figuras 3.21, 3.22, 3.23, 3.24).

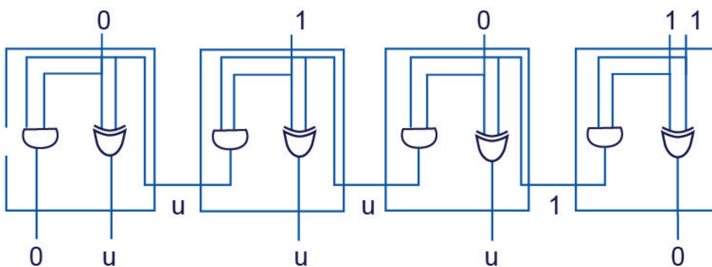


Figura 3.21

©Editorial Digital Tecnológico de Monterrey

tiempo=2deltas s=0uu00,

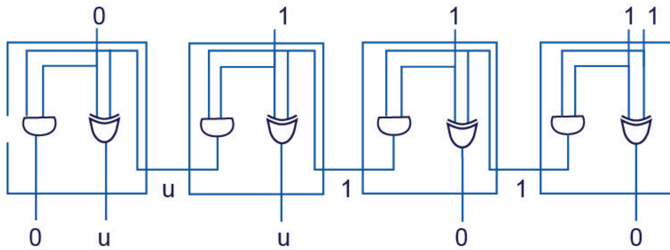


Figura 3.22

tiempo=3 deltas s=0u000,

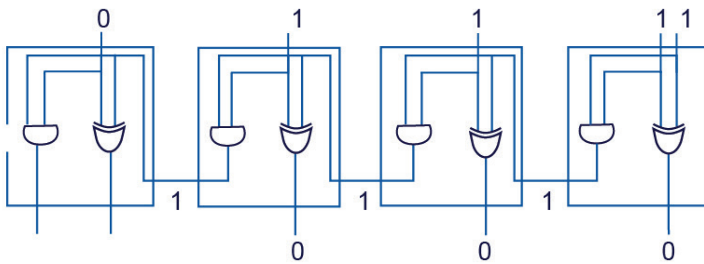


Figura 3.23

tiempo = 4 deltas en adelante s=01000.

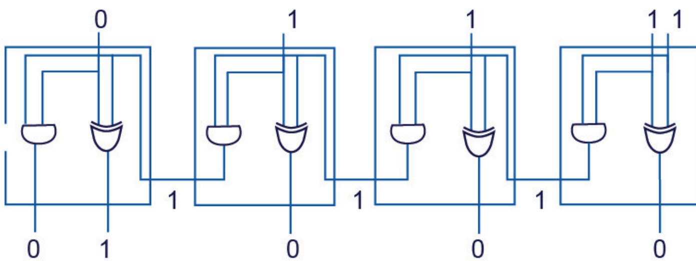


Figura 3.24

©Editorial Digital Tecnológico de Monterrey

Con la combinación **0101** a la señal **a** en un tiempo **= 0** valor final de s = 00110.

Con estos resultados es más fácil reconocer que el circuito se conforma por medios sumadores interconectados que incrementan **a** en 1.

La arquitectura de este circuito puede ser descrita en VHDL como: s <= ‘0’&a + 1, conservando el mismo *hardware* y por supuesto, la misma funcionalidad.

3.7.3 Unidad lógica

A continuación, se especifica una Unidad lógica que cuenta con 2 *bits* de control S1 y S0 que seleccionan una operación a realizar entre dos números A y B de 3 *bits* cada uno, así que las entradas al circuito son: A2, A1, A0, B2, B1, B0, S1 y S0 (tabla 3.4).

Selectores		Operación
S1	S0	
0	0	R=A or B
0	1	R=A or B
1.	0	R=not A
1	1	Sin uso

Tabla 3.4

En álgebra booleana, las funciones necesarias son:

$$\mathbf{R_2 = S_1'S_0'(A_2 + B_2) + S_1'S_0(A_2B_2) + S_1S_0'(A_2')}$$
$$\mathbf{R_1 = S_1'S_0'(A_1 + B_1) + S_1'S_0(A_1B_1) + S_1S_0'(A_1')}$$
$$\mathbf{R_0 = S_1'S_0'(A_0 + B_0) + S_1'S_0(A_0B_0) + S_1S_0'(A_0')}$$

©Editorial Digital Tecnológico de Monterrey

El circuito diseñado con compuertas lógicas (sin recurrir a tres *multiplexers*) es el que se muestra en la figura 3.25.

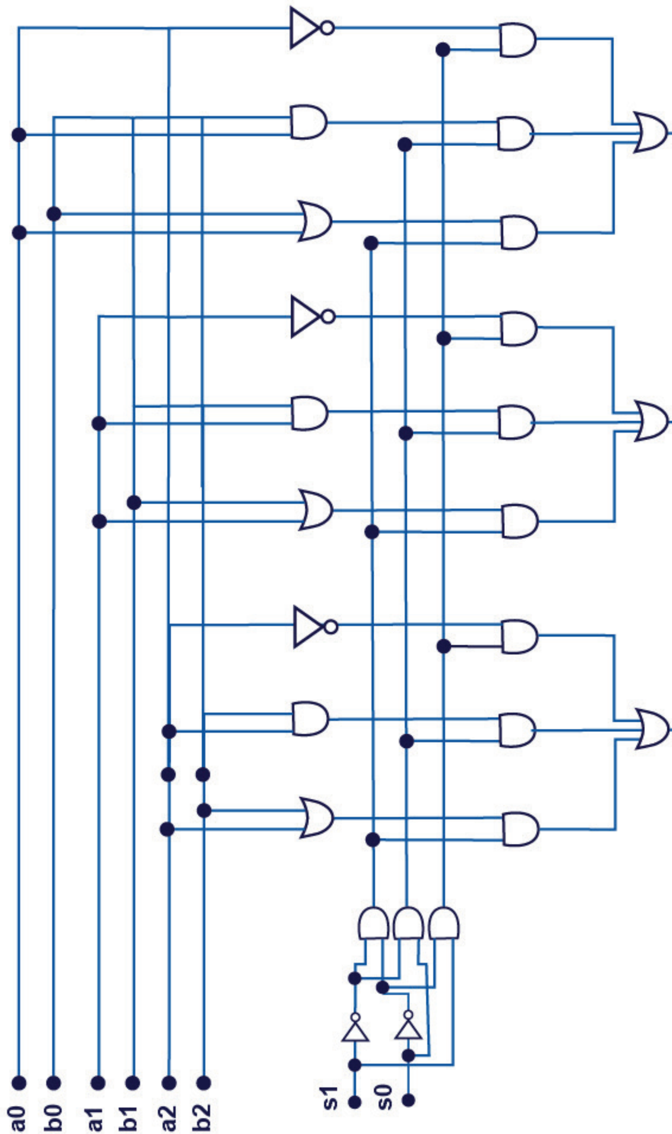


Figura 3.25 Circuito de un ALU a nivel de compuertas lógicas

©Editorial Digital Tecnológico de Monterrey

Un diseño esquemático más breve consiste en utilizar un *multiplexer* de varias salidas, que resume el dibujo de un *multiplexer* por cada *bit* de salida, como el que se observa en la figura 3.26:

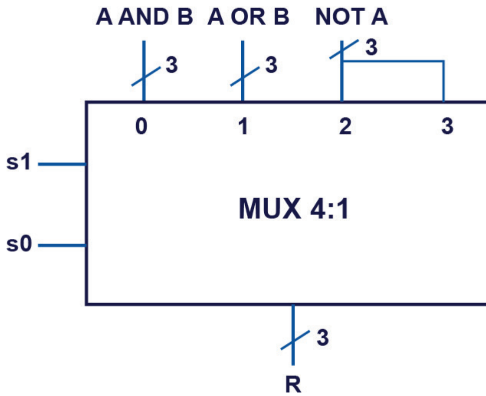


Figura 3.26 Circuito de un ALU a nivel de bloque (o de circuito integrado)

En este caso, A AND B simboliza el circuito que se aprecia en la figura 3.27.

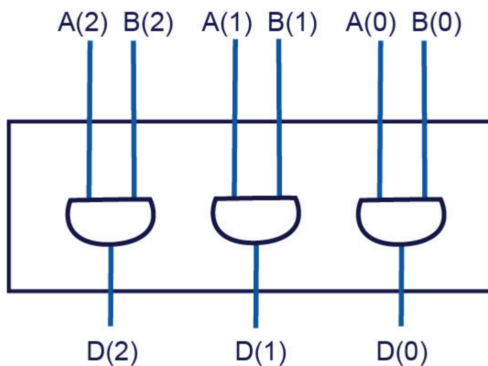


Figura 3.27 Circuito interno de A and B

En VHDL se tiene la facilidad de definir este circuito con: D <= A AND B

©Editorial Digital Tecnológico de Monterrey

Algo similar se representa con A OR B y con NOT A.

Así que en el nivel comportamiento, esta unidad lógica queda definida de la siguiente manera:

```
Entity LU is
Port (A,B: in std_logic_vector (2 downto 0);
           S1, S0: in std_logic;
          R: out std_logic_vector (2 downto 0));
End LU;
Architecture cualquiera of LU is
Begin
--El siguente es el multiplexer múltiple
R <= A OR B when (S1 = '0') AND (S0 = '0') else A AND B when (S1
= '0') AND (S0 = '1') else NOT A;
 -- si se presenta la combinación 1 1 en la entrada de control,
la salida será la operación not
End cualquiera;
```

3.7.4 Calculadora

Este ejemplo reúne muchos de los conceptos vistos hasta este punto. Se desea diseñar un sistema digital al cual ingresen dos números sin signo de 8 *bits* cada uno, A y B. A este sistema se conectarán cuatro *switches* para realizar estas cuatro posibles operaciones:

IncA: Incrementa A en 1

DecA: Decrementa A en 1

IncB: Incrementa B en 1

Suma: Suma A + B

©Editorial Digital Tecnológico de Monterrey

La salida de este sistema son 9 *bits*. Suponga que en un mismo tiempo el usuario solo utiliza **uno** de los cuatro *switches*. No realice acciones de diseño para otro caso (dos o más *switches* a la vez).

Para la solución se debe utilizar un diagrama de bloques con circuitos generales como sumadores y restadores de números binarios, comparadores, *multiplexer*s y *encoder*s. De esta forma, el circuito resulta como el que se presenta en la figura 3.28.

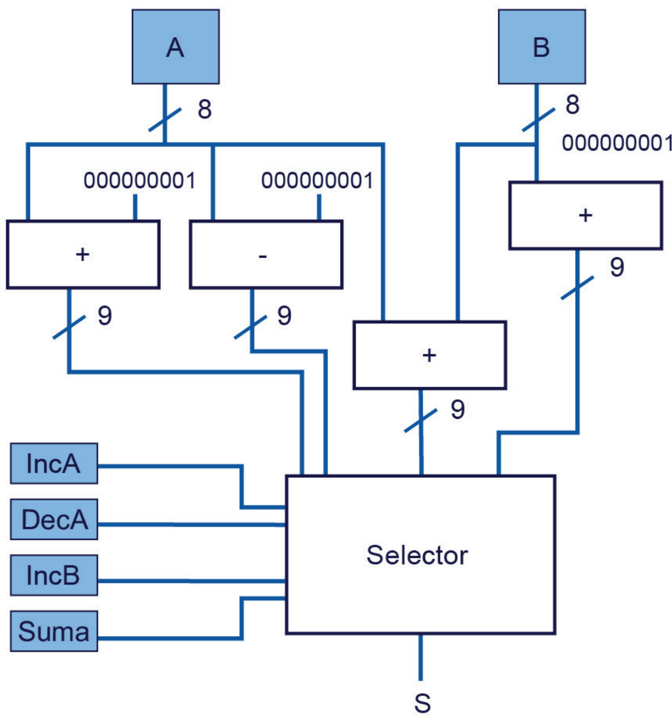


Figura 3.28 Diseño esquemático de la calculadora

El circuito selector resume la lógica combinacional requerida para seleccionar la salida de acuerdo a la entrada de control que se elija.

©Editorial Digital Tecnológico de Monterrey

La definición en VHDL de esta arquitectura es la siguiente:

```
entity problema2 is
    Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
           B : in  STD_LOGIC_VECTOR (7 downto 0);
           Tecla : in  STD_LOGIC_VECTOR (3 downto
0);
           S : out  STD_LOGIC_VECTOR (8 downto 0));
end problema2;
architecture Behavioral of Problema2 is

     signal incA : STD_LOGIC_VECTOR (8 downto 0);
     signal decA : STD_LOGIC_VECTOR (8 downto 0);
     signal incB : STD_LOGIC_VECTOR (8 downto 0);
     signal suma : STD_LOGIC_VECTOR (8 downto 0);

begin
     incA  <= '0'&A + 1;
     decA  <= '0'&A - 1;
     incB  <= '0'&B + 1;
     suma  <= '0'&A + '0'&B;
     result: process(Tecla) begin
          if tecla = "1000" then
               s  <= incA;
          elsif tecla = "0100" then
               s  <= decA;
          elsif tecla = "0010" then
               s  <= incB;
          elsif tecla = "0001" then
               s  <= suma;
          else null;
          end if;
     end process;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 3

1. Diseñe en VHDL un comparador que detecte la condición mayor o igual (salida llamada MI) sin utilizar operadores relacionales (=, >, <, etc.), ni medios comparadores o comparadores completos; en vez de esto, utilice la operación de resta.

2. Dado el siguiente circuito descrito en VHDL, realice las actividades que se indican más adelante:

```
entity RC is
Port ( x : in std_logic;
        bin : in std_logic;
        s : out std_logic;
        bout : out std_logic);
end RC;

architecture ecuaciones of RC is
begin
s<= x xor bin ;
bout<= not x and bin;
end ecuaciones;

entity s4 is
Port ( a : in std_logic_vector(2 downto 0);
        r : out std_logic_vector(2 downto 0));
end s4;

architecture estructural of s4 is
component RC
```

©Editorial Digital Tecnológico de Monterrey

```
Port ( x : in std_logic;
        bin : in std_logic;
          s : out std_logic;
        bout : out std_logic);
end component;
signal c :std_logic_vector(3 downto 1);
begin
RC2: RC port map (a(2), c(2), r(2), c(3));
RC1: RC port map (a(1), c(1), r(1), c(2));
RC0: RC port map (a(0), '1', r(0), c(1));
end estructural;
```

a) Muestre las salidas a y r desde tiempo=0, hasta tiempo=5 deltas mostrando la actualización de a y r en cada delta.

b) Indique qué operación aritmética realiza este circuito.

3. ¿Cuál es la función del siguiente circuito descrito en VHDL?

```
entity acertijo is
  Port (a:in bit_vector(0 to 3);
       b: in bit_vector(0 to 3);
       salida : out bit);
end acertijo;

architecture ecuaciones of acertijo is
signal z: bit_vector (1 to 3);
begin
z(1)<= (not (a(0) xor b(0)));
z(2)<= (not (a(1) xor b(1))) and z(1);
z(3)<= (not (a(2) xor b(2))) and z(2);
salida<= (not (a(3) xor b(3))) and z(3);
end ecuaciones;
```

©Editorial Digital Tecnológico de Monterrey

4. Dado el siguiente circuito descrito en VHDL, realice las actividades que se indican más adelante.

```
 entity RC is
Port ( x : in bit;
          bin : in bit;
          s : out bit;
          bout : out bit);
 end RC;
 architecture ecuaciones of RC is
 begin
 s <= x xor bin ;
 bout <= (not x and bin);
 end ecuaciones;

 entity s4 is
Port ( a : in bit_vector(2 downto 0);
          r : out bit_vector(2 downto 0));
 end s4;

 architecture estructural of s4 is
 component RC
Port ( x : in bit;
          bin : in bit;
               s : out bit;
          bout : out bit);
 end component;
 signal c :bit_vector(3 downto 1);
 begin
 r2: RC port map (a(2), c(2), r(2), c(3));
 r1: RC port map (a(1), c(1), r(1), c(2));
 ro: RC port map (a(0), '1', r(0), c(1));
 end estructural;
```

a) Si en tiempo cero a = 110, indique los valores de r en tiempo cero, delta, dos deltas, tres deltas, cuatro deltas.

b) Indique qué operación aritmética se realiza en este circuito.

©Editorial Digital Tecnológico de Monterrey

5. Diseñe una unidad lógica (diseño esquemático y descripción en VHDL), que realice las operaciones indicadas en la tabla 3.5.

Selectores ***S0***	Operación
0	***R=A or B***
1	***R=A or B***

Tabla 3.5 Unidad lógica de dos operaciones

A y B son números de 3 *bits*. El puerto del circuito es el siguiente:

```
entity LU is
  Port (A:in bit_vector(2 downto 0);
        B: in bit_vector(2 downto 0);
        S0: in bit;
        R : out bit_vector(2 downto 0));
end LU;

architecture ecuaciones of LU is
begin

end ecuaciones;
```

©Editorial Digital Tecnológico de Monterrey

6. Se le pide diseñar un dispositivo muy básico que utilizarán bebés que solo conocen los dígitos del 0 al 4.

Este dispositivo cuenta con cinco teclas para seleccionar un dígito del 0 al 4 y otras dos teclas para elegir una de dos operaciones a realizar con el dígito: INC para incrementar en uno el dígito elegido y DUP para duplicarlo. La salida se desplegaría en un *display* de siete segmentos.

Si no se elige ninguna de las dos teclas de operaciones, entonces el *display* mostrará el dígito seleccionado (figura 3.29).

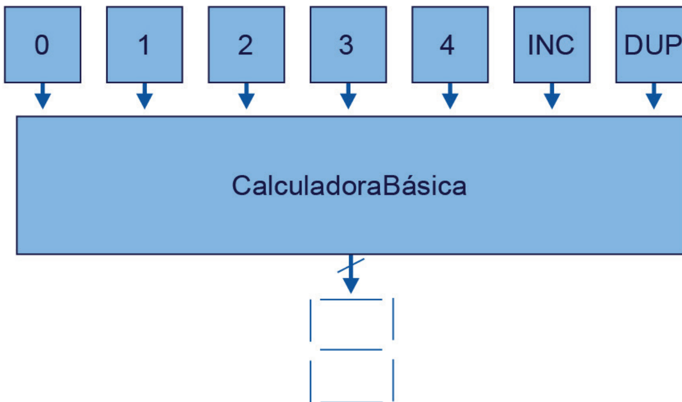


Figura 3.29 Diagrama de la calculadora

a) Muestre un diseño esquemático para resolver este circuito usando sumadores, *multiplexer*s y *encoder*s.

b) Muestre en VHDL la arquitectura de este circuito usando el siguiente puerto:

```
Entity calculadora is
Port (teclas: in std_logic-vector (4 downto 0); INC, DUP:in std_
logic;
Segmentos: out std_logic_vector(6 downto 0));
End calculadora;
```

©Editorial Digital Tecnológico de Monterrey

7. Dado el siguiente módulo en VHDL realice lo indicado en los incisos a y b.

```
entity sorpresa is
  Port (x:in bit_vector(2 downto 0);
       s: in bit_vector(2 downto 0));
end sorpresa;

architecture ecuaciones of sorpresa is
signal c:bit_vector (1 to 3);
begin
s(0)<= x(0) xor '1' after 1 ns;
c(1)<= x(0) after 1 ns;
s(1)<= x(1) xor c(1) after 1 ns;
c(2)<= x(1) and c(1) after 1 ns;
2(2)<= x(2) xor c(2) after 1 ns;
c(3)<= x(2) and c(2) after 1 ns;
end ecuaciones;
```

a) Si se asigna un valor de **101** a la señal **x** en un tiempo**=0**, ¿qué valor tiene s en tiempo = **2ns**?

b) Indique qué hace el módulo VHDL, es decir, ¿cuál es su función?

©Editorial Digital Tecnológico de Monterrey

8. Diseñe en VHDL una unidad lógica que cuente con las operaciones que se muestran en la tabla 2.5.

Selectores **S0**	Operación
0	**R=A or B**
1	**R= not A xor B**

Tabla 2.6 Unidad lógica de dos operaciones

A y B son números de 3 *bits*. Las operaciones lógicas or and y not se aplican *bit* a *bit*. El puerto es el siguiente.

```
entity LU is
  Port (A:in bit_vector(2 downto 0);
       B: in bit_vector(2 downto 0);
       S0: in bit;
       R : out bit_vector(2 downto 0));
end LU;

architecture ecuaciones of LU is
begin

end ecuaciones;
```

©Editorial Digital Tecnológico de Monterrey

9. Diseñe un módulo VHDL que describa de modo funcional el circuito requerido para calcular el complemento a 2 de un número de 3 *bits*. Recuerde que el complemento a 2 puede ser calculado sumando 1 al complemento a 1 del número, es decir, N * = N' + 1. Por ejemplo: 001 * = 111. El puerto se muestra a continuación:

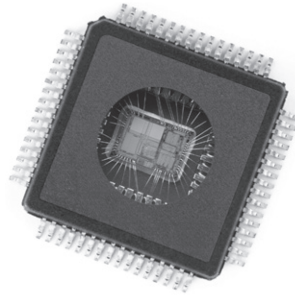
```
entity compa2 is
  Port (N:in bit_vector(2 downto 0);
       Nc2: out bit_vector(2 downto 0));
end compa2;

architecture behavioral of compa2 is

begin

end behavioral;
```

Nc2<=not N + 1;

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 3

Al término de este capítulo ya se tienen los conocimientos suficientes para construir dispositivos que utilicen circuitos combinacionales complejos, como son las unidades aritméticas y lógicas. El siguiente capítulo amplía las posibilidades de las aplicaciones aritméticas al agregar números BCD para la alimentación de datos.



©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 4. Aritmética BCD

Aritmética BCD

Diseño de un sumador BCD

Diseño de un restador BCD

Ejemplos

Este capítulo 4 está dedicado al diseño de los circuitos necesarios para capturar números BCD y realizar operaciones aritméticas con estos.

Un número **BCD** (Binary Coded Decimal) es un conjunto de dígitos, en los que cada dígito se encuentra en el rango de **0** a **9** y está expresado en binario. Un número BCD no es lo mismo que un número en base 2. Por ejemplo:

$$496_{10} = 0100\ 1001\ 0110_{BCD} = 111110000_2$$
$$514_{10} = 0101\ 0001\ 0100_{BCD} = 1000000010_2$$

En el número BCD se insertaron espacios cada cuatro *bits* solo para señalar el final de un dígito y el inicio del siguiente.

En BCD, el número decimal se encuentra expresado en binario dígito por dígito, no es un sistema posicional en binario en todo el conjunto, la posición es la misma cada cuatro *bits* y es una potencia de **10**, no de **2**.

©Editorial Digital Tecnológico de Monterrey

Cuando se interactúa con un sistema digital con un teclado, por ejemplo, las cifras que se proporcionen oprimiendo las teclas pueden quedar registradas fácilmente en BCD si lo que representa cada una se convierte a su equivalente en binario.

El componente que se usa en la detección de una señal y su conversión a binario se llama *encoder*. Por ejemplo, la representación esquemática de un *encoder* al cual se le ingresen ocho señales provenientes de ocho botones que representen dígitos del 0 al 7 y cuya salida es la conversión a binario del botón que se detecte oprimido es la que se aprecia en la figura 4.1:

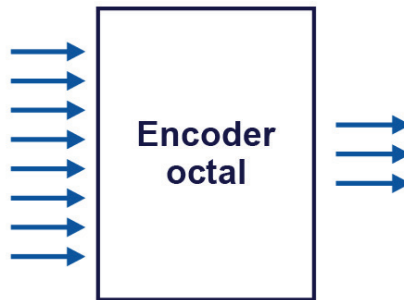


Figura 4.1 Encoder de ocho entradas a tres salidas

Se definirá un componente que cuenta con ocho entradas (**D_IN**) y tres salidas (**D_OUT**). Se supondrá que solo una de las entradas que se den a **D_IN** tendrá un valor de 1. La salida reflejará cuál de las entradas, de la **7** a la **0**, es **1**.

La codificación en VHDL de este *encoder* se presenta utilizando circuitos combinacionales modelados por ecuaciones concurrentes. Para diseñar estos circuitos es posible definir la operación a través de su operación, que es la siguiente:

©Editorial Digital Tecnológico de Monterrey

```
D_IN          D_OUT
Índice        índice
76543210      210
00000001      000
00000010      001
00000100      010
00001000      011
00010000      100
00100000      101
01000000      110
10000000      111
```

Se está suponiendo que D_IN tiene conectados botones del 7 al 0 respectivamente, y la salida indica el valor en binario de la tecla que se seleccione.

Si a la entrada del dispositivo ocurre una combinación inválida, la salida no está determinada, a menos que explícitamente se determine que sea cero.

La tabla de verdad resultaría demasiado extensa, ya que para ocho (con letra) entradas se tienen 28 = 256 combinaciones. Sin embargo, de la tabla de operación que se muestra, solamente se omiten las combinaciones que no importan, es decir, aquellas que tienen las salidas no determinadas; así que de esta tabla es posible extraer las expresiones booleanas necesarias, las cuales se muestran en el siguiente código:

©Editorial Digital Tecnológico de Monterrey

```
entity encoder is
 Port (D_IN: in STD_LOGIC_VECTOR(7 downto 0);
D_OUT: out STD_LOGIC _VECTOR(2 downto 0));
end encoder;
architecture ecuaciones of encoder is
begin
D_OUT(2) <= D_IN(7) or D_IN(6) or D_IN(5) or
D_IN(4);
D_OUT(1) <= D_IN(7) or D_IN(6) or D_IN(3) or
D_IN(2);
D_OUT(0) <= D_IN(7) or D_IN(5) or D_IN(3) or
D_IN(1);
end ecuaciones;
```

Ante esta definición, el lector puede precisar las salidas que ocurren en entradas que no se determinaron.

La equivalencia a binario de un número D en BCD formado por los dígitos $\mathbf{D_{BCD} = d_n d_{n-1}\ d_{n-2} d_{n-3} \ldots\ d_1 d_{0\ BCD}}$ se calcula con la siguiente operación:

$$D_2 = d_n * 10_{10}^{n} + d_{n-1} * 10_{10}^{n-1} + d_{n-2} * 10_{10}^{n-2} + d_{n-3} * 10_{10}^{n-3} + \ldots + d_1 * 10_{10}^{1} + d_0 * 10_{10}^{0}$$

La multiplicación es una operación que se realiza con un circuito combinacional, lo mismo que la suma, por lo que el circuito requerido para realizar esta operación tiene únicamente el retraso de un circuito combinacional. Sin embargo, si las operaciones a realizar con los números BCD son sumas y restas –y el resultado se requiere presentar en BCD– será más simple diseñar un circuito que sume directamente en BCD, que convertir a binario y después volver a convertir a BCD, porque este cálculo requiere divisiones. La división solo se resuelve con un circuito secuencial, por lo que el resultado requerirá cierto número de ciclos de reloj, dependiendo de la longitud del número, resultando un circuito más lento y complejo.

En el subtema que sigue se presenta un circuito para sumar números BCD.

©Editorial Digital Tecnológico de Monterrey

4.1 Diseño de un sumador BCD

Este problema utiliza circuitos para sumar, restar, comparar y seleccionar; los usa como circuitos integrados, indicándolos como bloques en los diseños esquemáticos. La suma de dos números BCD se resuelve en cascada, sumando cada posición BCD que consta de cuatro *bits*, esto genera un dígito BCD de respuesta y un acarreo (figura 4.2).

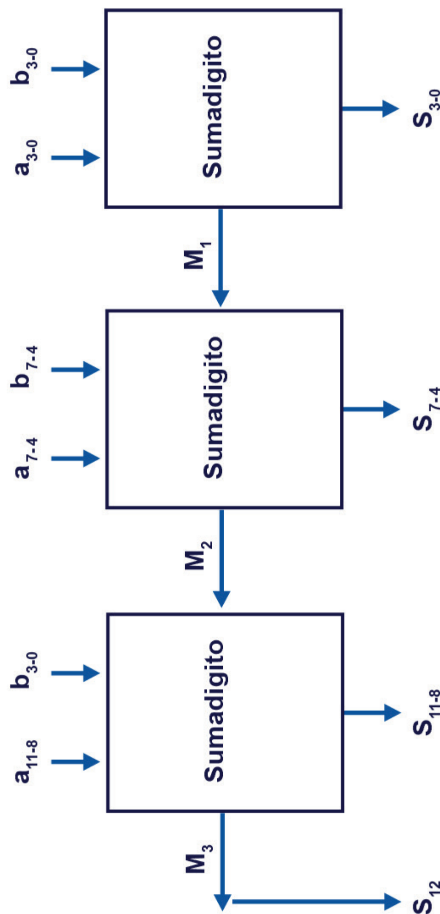


Figura 4.2 Sumador de números BCD con conexión en cascada de bloques de sumadores de dígitos BCD

©Editorial Digital Tecnológico de Monterrey

Para diseñar el circuito que sume dos números en BCD podemos iniciar revisando la suma en decimal.

Como un dígito es menor que **10**, si se excede a este, se toma el dígito que rebasa una posición y se suma en la siguiente (como *carry*).

```
   5
 + 4
 ---
   9

   5
 + 8
 ---
  13

   9
 + 9
 ---
  18

   1
  25
+ 38
----
  63

   0
  25
+ 34
----
  59

   1
  29
+ 38
----
  67

   1
  99
 +58
----
 157
```

Un dígito BCD se representa en cuatro *bits*, pero con cuatro *bits* podemos representar de 0 a 15 decimal. La suma máxima que puede resultar de sumar dos dígitos requiere cinco *bits*, ya que 8 + 8, 9+7 o cualquier suma que exceda a 15 requiere cinco *bits*. La suma mayor posible de dos dígitos más un acarreo es: 9+9+1 = 100112 que como se observa requiere cinco *bits* para su representación (con cinco *bits* es posible representar hasta 25-1=31).

A continuación se presentan las sumas en BCD de los ejemplos anteriores. Se hace notar que si al sumar dos dígitos el resultado

©Editorial Digital Tecnológico de Monterrey

excede a **9** es necesario restarle 1010 y propagar un acarreo. Además, el resultado necesita cinco *bits* para cubrir todos los posibles resultados:

```
      0101
    + 0100
    ------
     01001

      0101
    + 1000
    ------
     01101
    -01010
    ------
1    00011   en BCD: 0001 0011

    + 1001
    ------
     10010
    -01010
    ------
1    01000   en BCD: 0001 1000

         1
      0010    0101
     +0011    1000
     -------------
     00110   01101
            -01010
     -------------
     00110   01001

         0
      0010    0101
     +0011    0100
     -------------
     00110   01001 en BCD: 000 0110 1001

         1
      0010    1001
     +0011    1000
     -------------
     00110   10001
            -01010
     -------------
     00110   00111 en BCD: 000 0110 0111
```

©Editorial Digital Tecnológico de Monterrey

En resumen, el algoritmo para sumar dos dígitos BCD es: se suma en binario ambos dígitos, si el resultado es mayor a **9**, ahí se resta **10** y se pasa un acarreo de **1** a la siguiente posición BCD, si no, no se requiere resta y el acarreo es **0**.

En un circuito, una condición lógica se construye con un selector (*multiplexer*). Desde que se energiza, un circuito siempre está en operación, no es posible cancelar alguna de las operaciones, todas se efectúan, lo que permite el selector es elegir la que se requiere como respuesta, dada una condición alimentada por las entradas de control.

La suma de los dígitos BCD se compara con **9** y también se le resta **10**; si la suma es mayor a **9**, se selecciona el resultado de la resta, si no, se deja la suma. El valor máximo de la suma es **19**, como ya se analizó, al restarle **10** queda 9, así que solo se requieren 4 *bits* para el resultado, el quinto *bit*, el más significativo, debe ser cero y se desecha (figura 4.3).

©Editorial Digital Tecnológico de Monterrey

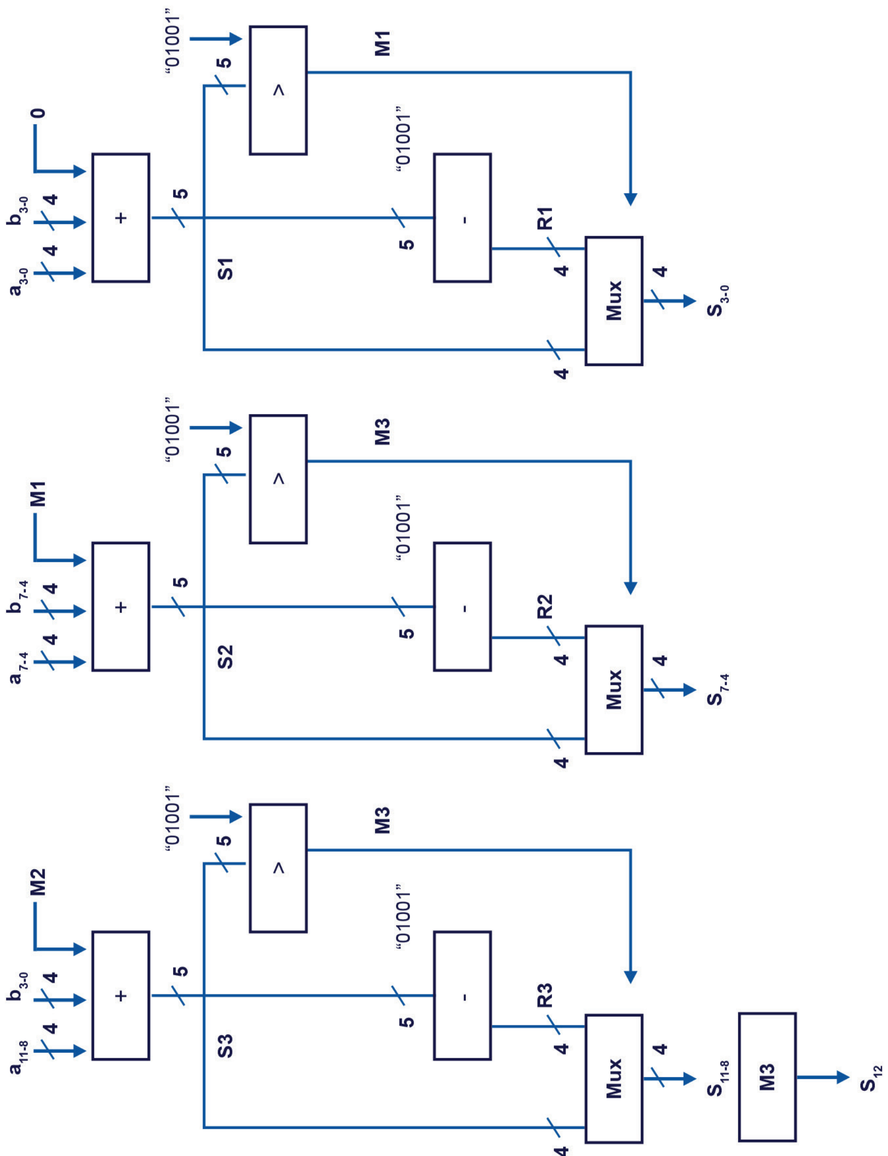


Figura 4.3 Sumador de números BCD con conexión en cascada de sumadores de dígitos BCD con su arquitectura interna

El *bit* 12 del resultado, que constituye el acarreo final, o que bien puede ser el *bit* menos significativo de un dígito **BCD** –que es parte de la respuesta–, es producido por **M3**.

©Editorial Digital Tecnológico de Monterrey

Una vez diseñado un módulo que sume dos dígitos BCD puede replicarse para sumar cualquier número de dígitos.

Para describir el circuito en VHDL tenemos diferentes posibilidades, una de ellas es definir el componente Sumadigito para luego instanciarlo tres veces (figura 4.4).

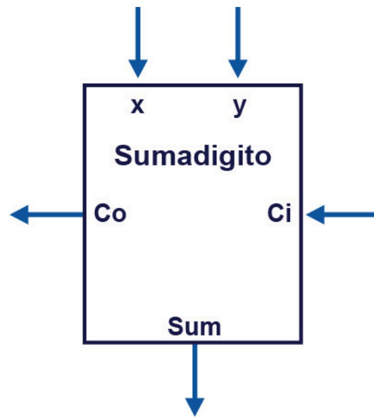


Figura 4.4 Descripción del circuito sumador de un dígito BCD en cascada de bloques de sumadores dígitos BCD

```
Entity Sumadigito is
Port (     x, y: in std_logic_vector(3 downto 0);
       Ci:in std_logic;
       Sum:out std_logic_vector(3 downto 0);
       Co: out std_logic);
End Sumadigito;

Architecture ecuaciones of Sumadigito is
signal s,r:std_logic_vector(4 downto 0);
signal m: std_logic;
begin
s <= ('0'&x)+ ('0'&y)+Ci;
m <= '1' when (s1>"01001") else '0';
r <= s1-"01010";
Sum <= s(3 downto 0) when (m = '0') else r(3 downto 0);
Co<)m;
End ecuaciones;
```

©Editorial Digital Tecnológico de Monterrey

Una vez descrito el componente es factible conectar tres de estos para conseguir la suma de los tres dígitos.

```
entity sumaBCD  is
    Port ( a : in std_logic_vector(11 downto 0);
           b : in std_logic_vector(11 downto 0);
             sum : out std_logic_vector(12 downto 0));
        end sumaBCD;

architecture estructural of sumaBCD is
signal m1,m2,m3: std_logic;
component sumadigito is
Port (    x, y: in std_logic_vector(3 downto 0);
   Ci:in std_logic;
     Sum:out std_logic_vector(3 downto 0);
       Co: out std_logic);
End component;
begin
SD0: sumadigito port map (a(3 downto 0), b(3 downto 0), ‘0’, s(3 downto 0), m1);
SD1: sumadigito port map (a(7 downto 4), b(7 downto 4), m1, s(7 downto 4), m2);
SD2: sumadigito port map (a(11 downto 8), b(11 downto 8), m2, s(11 downto 8), s(12));
End estructural;
```

©Editorial Digital Tecnológico de Monterrey

Otra opción para el código en VHDL es no instanciar los componentes, sino describir cada uno.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity sumaBCD is
     Port ( a : in std_logic_vector(11 downto 0);
            b : in std_logic_vector(11 downto 0);
            sum : out std_logic_vector(12 downto 0));
end sumaBCD;

architecture Behavioral of sumaBCD is
signal s1,aux1,aux2,r1,s2,r2,aux11,aux22, s3, r3, aux33, aux44
:std_logic_vector(4 downto 0);
signal m1,m2,m3: std_logic;
begin
aux1 <= '0'&a(3 downto 0);
aux2 <= '0'&b(3 downto 0);
s1 <= aux1 + aux2;
m1 <= '1' when (s1>"01001") else '0';
r1 <= s1-"01010";
sum(3 downto 0) <= s1(3 downto 0) when (m1 = '0') else r1(3 downto 0);

aux11 <= '0'&a(7 downto 4);
aux22 <= '0'&b(7 downto 4);
s2 <= aux11 + aux22 + m1;
m2 <= '1' when (s2>"01001") else '0';
r2 <= s2-"01010";
sum(7 downto 4) <= s2(3 downto 0) when (m2 = '0') else r2(3 downto 0);

aux33 <= '0'&a(11 downto 8);
aux44 <= '0'&b(11 downto 8);
s3 <= aux33 + aux44 + m2;
m3 <= '1' when (s3>"01001") else '0';
r3 <= s3-"01010";
sum(11 downto 8) <= s3(3 downto 0) when (m3 = '0') else r3(3 downto 0);

sum(12) <= m3;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

A continuación, se muestra lo que el sintetizador reconoce de este código, para así constatar que los componentes del diseño son descubiertos por esta pieza de software.

```
Synthesizing Unit <sumaBCD >.
Related source file is C:/Xilinx/bin/prueba1/sumaBCD .vhd.
WARNING:Xst:646 - Signal <r1<4>> is assigned but never used.
WARNING:Xst:646 - Signal <r2<4>> is assigned but never used.
WARNING:Xst:646 - Signal <r3<4>> is assigned but never used.
Found 5-bit comparator greater for signal <$n0000> created at line 19.
Found 5-bit comparator greater for signal <$n0002> created at line 26.
Found 5-bit comparator greater for signal <$n0004> created at line 33.
Found 5-bit subtractor for signal <r1>.
Found 5-bit subtractor for signal <r2>.
Found 5-bit subtractor for signal <r3>.
Found 5-bit adder for signal <s1>.
Found 5-bit adder carry in for signal <s2>.
Found 5-bit adder carry in for signal <s3>.
Found 12 1-bit 2-to-1 multiplexers.
Summary:
inferred 6 Adder/Subtracter(s).
inferred 3 Comparator(s).
inferred 12 Multiplexer(s).
Unit <sumaBCD > synthesized.
```

```
============================* HDL Synthesis Report *==============================
Macro Statistics
# Multiplexers : 12
2-to-1 multiplexer : 12
# Adders/Subtractors : 6
5-bit adder : 1
5-bit subtractor : 3
5-bit adder carry in : 2
# Comparators : 3
5-bit comparator greater : 3

===========================* Advanced HDL Synthesis *=============================

============================* Low Level Synthesis *===============================
Optimizing unit <sumaBCD > ...
Loading device for application Xst from file '2s15.nph' in environment C:/Xilinx.

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block sumaBCD , actual ratio is 11.
```

©Editorial Digital Tecnológico de Monterrey

```
================================* Final Report *=================================
Device utilization summary:
---------------------------

Selected Device : 2s15cs144-6

Number of Slices: 24 out of 192 12%
Number of 4 input LUTs: 39 out of 384 10%
Number of bonded IOBs: 37 out of 90 41%


==========================================================
TIMING REPORT


Clock Information:
------------------
No clock signals found in this design
```

©Editorial Digital Tecnológico de Monterrey

4.2 Diseño de un restador BCD

Para restar dos números BCD se puede aprovechar el sistema posicional y restar "por columnas", igual que lo hacemos a mano. Es posible diseñar un componente para restar dos dígitos y luego interconectar todos los que necesitemos para restar los números completos.

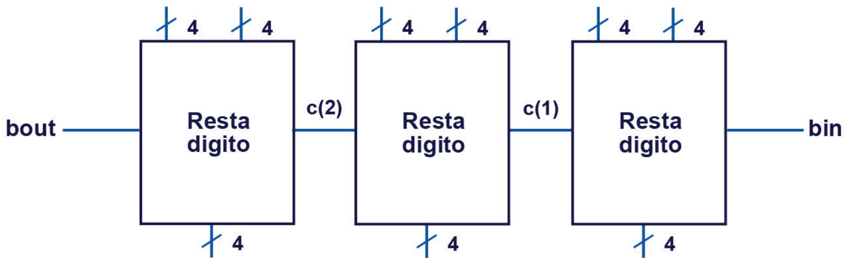


Figura 4.5 Restador de números BCD con conexión en cascada de bloques de restadores de dígitos BCD

Para restar dos dígitos BCD es posible utilizar el método que usamos cuando restamos manualmente, "pidiendo prestado" una unidad de la siguiente posición del minuendo y "devolviéndola" en la siguiente posición del sustraendo para no alterar los números que se están restando.

El procedimiento es el siguiente: si los dígitos a restar son X y Y, si X<Y entonces se restaría $X_i + 10 - Y_i$, es decir, se trae una unidad de la siguiente posición, que se regresará, para esto se generará un acarreo "borrow i+1" para saber que se debe una compensación que en la siguiente posición se corregiría $X_{i+1} - (Y_{i+1} + 10)$. El caso más crítico sería que al sumar (Yi+1 + 10) se produzca un acarreo a la siguiente posición.

©Editorial Digital Tecnológico de Monterrey

```
entity restadigito is
 Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
     y : in  STD_LOGIC_VECTOR (3 downto 0);
                  bin: in std_logic;
     z : out  STD_LOGIC_VECTOR (3 downto 0);
                  bout:out std_logic);
end restadigito;

architecture Behavioral of restadigito is
signal aux:std_logic_vector(4 downto 0);
signal M:std_logic;
begin
M <= '0' when x >= y+bin else '1';
z <= x-(y+bin) when M = '0'
 else aux(3 downto 0);
aux <= (('0'&x)+"01010") - (('0'&y)+bin);
bout <= M;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

La interconexión de estos componentes puede quedar de la siguiente manera:

```
entity restabcd is
 Port ( x : in  STD_LOGIC_VECTOR (11 downto 0);
     y : in  STD_LOGIC_VECTOR (11 downto 0);
     z : out  STD_LOGIC_VECTOR (11 downto 0);
     signo : out  STD_LOGIC);
end restabcd;

architecture Behavioral of restabcd is
component restadigito is
     Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
             y : in  STD_LOGIC_VECTOR (3 downto 0);
                 bin: in std_logic;
             z : out  STD_LOGIC_VECTOR (3 downto 0);
                 bout:out std_logic);
end component;
signal b: std_logic_vector(1 to 2);
begin
restadigito0:
               restadigito port map (x(3 downto 0), y(3
downto 0), '0', z(3 downto 0), b(1));
restadigito1:
           restadigito port map (x(7 downto 4), y(7
downto 4), b(1), z(7 downto 4), b(2));
restadigito2:
        restadigito port map (x(11 downto 8), y(11
downto 8), b(2), z(11 downto 8), signo);
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

4.3 Ejemplos

4.3.1 Calculadora base 3

A continuación, se presenta un problema que integra los conceptos revisados hasta este punto. El problema consiste en diseñar una calculadora que sume números en base 3. Para alimentar los datos se contará con 6 *switches*, tres para cada número en base 3, como se muestra en la figura 4.6 (se alimenta en base 3 y no en binario).

Figura 4.6 Descripción gráfica de la calculadora base 3

Para que esta calculadora funcione, el usuario deberá poner en on los *switches* correspondientes a los dos números que se deseen sumar.

La suma debe desplegarse en base 3 codificada en binario en un total de 3 *bits*, un *bit* de acarreo y 2 *bits* de un dígito base 3 codificado en binario. Cada dígito en base 3 se codifica en 2 *bits*.

La base 3 codificada en binario se genera de la siguiente manera: si el resultado de sumar los dos dígitos base 3 es mayor o igual a 3, entonces se resta 3, el dígito base 3 codificado en binario es el resultado de la resta y se genera un 1 como acarreo. Si el resultado no es mayor o igual a 3, entonces el acarreo es 0 y el dígito base 3 codificado en binario es igual a la suma obtenida (solo 2 *bits*).

Ejemplo 1.

$2 + 2 = 4$, $4 - 3 = 1$, resultado = 1 01, este es el número en base 3 codificado en binario.

©Editorial Digital Tecnológico de Monterrey

Ejemplo 2.

1 + 1 = 2 resultado = 0 10, este es el número en base 3 codificado en binario.

El diseño esquemático del circuito es el que se muestra en la figura 4.7.

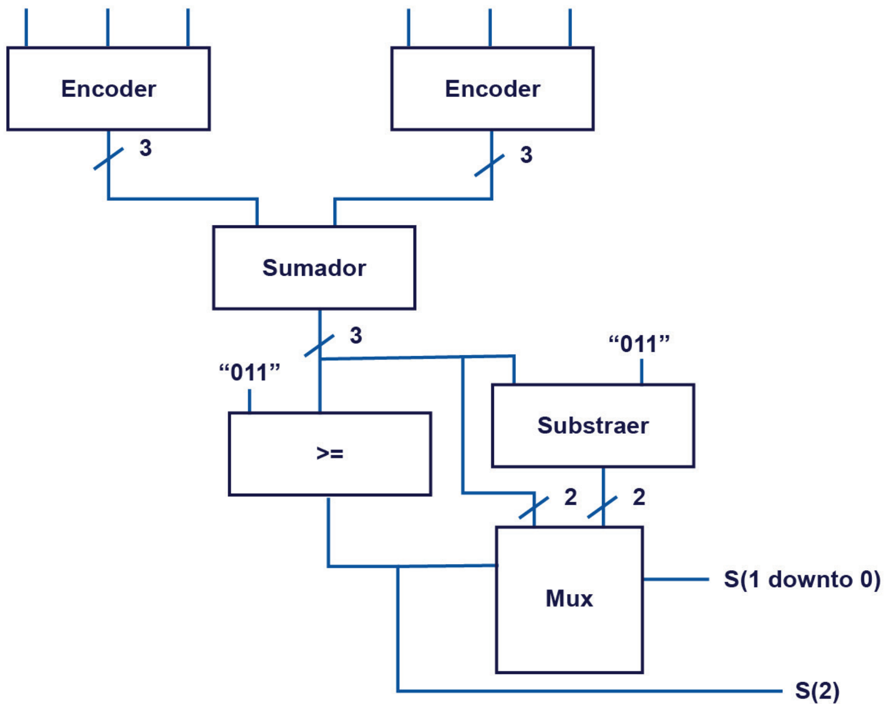


Figura 4.7 Diseño esquemático de la calculadora base 3

©Editorial Digital Tecnológico de Monterrey

```
entity problema1 is
     Port ( N1 : in  STD_LOGIC_VECTOR (2 downto 0);
                N2 : in  STD_LOGIC_VECTOR (2 downto
0);
                   S : out  STD_LOGIC_VECTOR (2 downto
0));
end problema1;
architecture Behavioral of problema1 is

signal num1 : STD_LOGIC_VECTOR (2 downto 0);
     signal num2 : STD_LOGIC_VECTOR (2 downto 0);
     signal suma : STD_LOGIC_VECTOR (2 downto 0);
     signal cod : STD_LOGIC_VECTOR (2 downto 0);

begin

num1 <= "000" when n1 = "100" else
               "001" when n1 = "010" else
               "010" when n1 = "001";

     num2 <= "000" when n2 = "100" else
               "001" when n2 = "010" else
               "010" when n2 = "001";

     suma <= num1 + num2;
     cod <= suma - "011";

     rslt: process(suma) begin
          if suma >= "011" then
               s(2) <= '1';
               s(1 downto 0) <= cod(1 downto 0);
          else
               s(2) <= '0';
               s(1 downto 0) <= SUMA(1 downto 0);
          end if;
     end process;

end Behavioral;
_________________________________________
```

©Editorial Digital Tecnológico de Monterrey

4.3.2 Suma de números en formato de signo magnitud

Este problema consiste en diseñar un circuito que sume dos números x y y de 8 *bits* que se encuentran en formato de signo magnitud (el signo 1 es el negativo). El *bit* de la posición 7 es el signo y el resto es la magnitud. El resultado suma mantiene el mismo formato.

- Al sumar dos números con signo magnitud pueden surgir los siguientes casos:
- Ambos positivos (solo se suman).
- Ambos negativos (solo se suman).
- De diferente signo y $|x| \geq |y|$ en cuyo caso hay que efectuar $|x| - |y|$ el signo resultante es el de x, ejemplo $-6 + 3 = -(6-3)$.
- De diferente signo y $|x| < |y|$ en cuyo caso hay que efectuar $|y| - |x|$ el signo resultante es el de y, ejemplo $3 + (-6) = -(6-3)$.

©Editorial Digital Tecnológico de Monterrey

El circuito esquemático para resolver esta suma se presenta en la figura 4.8.

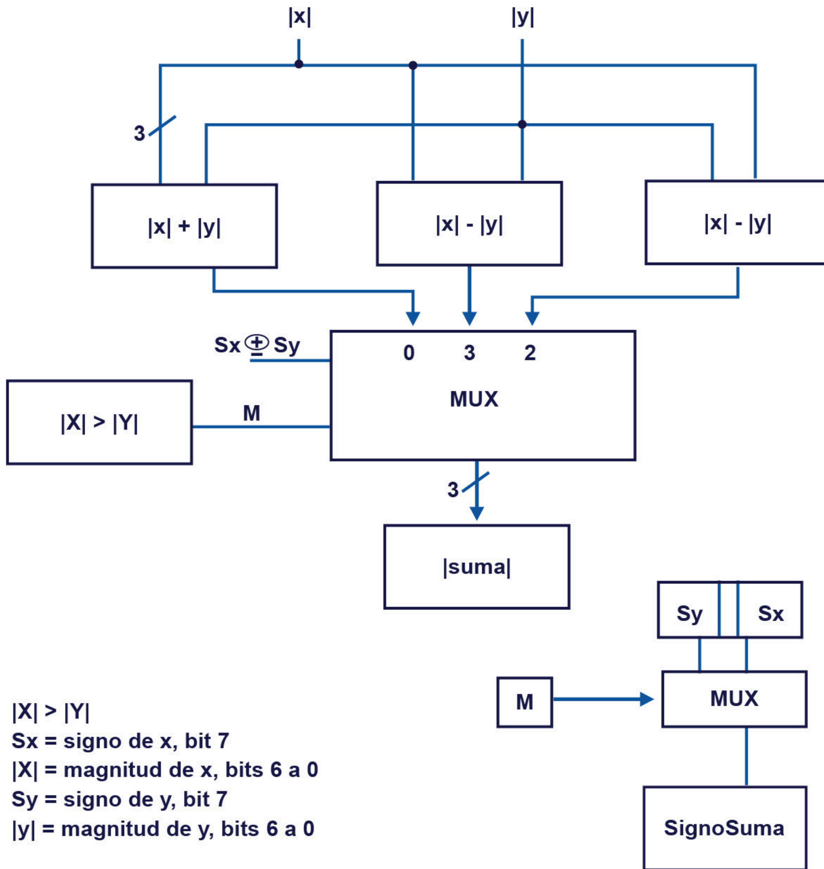


Figura 4.8 Diseño esquemático de sumador de números en formato de signo magnitud

©Editorial Digital Tecnológico de Monterrey

El circuito descrito en código para resolver este problema es el siguiente:

```
entity sumasm2 is
 Port ( x : in std_logic_vector(7 downto 0);
     y : in std_logic_vector(7 downto 0);
     suma : out std_logic_vector(7 downto 0));
end sumasm2;
architecture Behavioral of sumasm2 is
signal xm, ym, xc, yc, s, r1, r2:std_logic_vector(7
downto 0);
signal sx, sy, m, sig: std_logic;
begin
xm <= x(6 downto 0);
sx <=x(7);
ym<= y(6 downto 0);
sy <= y(7);
s <= xm+ym;
r1 <= xm-ym;
r2 <= ym-xm;
sig <= sx xor sy;
m <= '1' when (xm>ym) else '0';
suma(6 downto 0) <= s when (sig = '0') else r2 when (m
= '0') else r1;
suma(7) <= sy when (m = '0')else sx;
end Behavioral;
```

Note que la magnitud de la suma queda en 7 *bits*, si hay un *overflow* se pierde, no se registra.

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 4

1. Los Xoxo son ha*bit*antes del planeta X123, ellos solo tienen cuatro dedos (dos en cada mano) a eso se debe que su sistema numérico sea base 4. Luego de aterrizar en una conocida universidad pidieron que los alumnos de un curso de Diseño Digital que les construyan algunos dispositivos. Se le solicita a usted diseñarles una calculadora que sume y reste dos números sin signo en base 4. Para alimentar los datos se contará con 8 *switches*, cuatro para cada número, como se muestra en la figura.

Figura 4.9 Descripción gráfica de calculadora base 4

Para que esta calculadora funcione el usuario deberá poner en “on” los *switches* correspondientes a los dos números que se deseen sumar (un *switch* para cada número).

Ejemplos de sumas son 34+24 = 114, 34+34=124, 14+34=104, 24+14=34

Ejemplos de restas son 34-24 = 14, 34-34=04, 14-34=-24, 24-14=14

©Editorial Digital Tecnológico de Monterrey

Muestre el resultado de la suma y la resta en LEDs (al mismo tiempo) en binario base 4 (sistema numérico que también entienden los Xoxo).

Para la suma use cuatro LEDs: el 114 lo puede iluminar como 01 01, el 124 lo puede iluminar como 01 10. Para la resta use tres LEDs, uno para el signo, por ejemplo -24= (1)104.

2. Se le pide diseñar un sistema digital al cual se conectarán a su entrada 20 *switches* y a su salida dos *display*s de 7 segmentos para realizar una suma.

El funcionamiento del sistema es el siguiente:

- Con los primeros 10 *switches* se elige un dígito (el *switch* 0 corresponde al 0, el *switch* 1 corresponde al 1, y así hasta el 9, como si fueran diez teclas de un primer teclado),
- Con los otros 10 *switches* se elige un segundo dígito (de igual forma el *switch* 0 corresponde al 0, el *switch* 1 corresponde al 1, y así hasta el 9, como si fueran las teclas de un segundo teclado)
- Con los dos *display*s se ilumina la respuesta en decimal de la suma de los dos dígitos seleccionados con los *switches*.

El sistema completo que consta de cinco componentes se ilustra a continuación:

©Editorial Digital Tecnológico de Monterrey

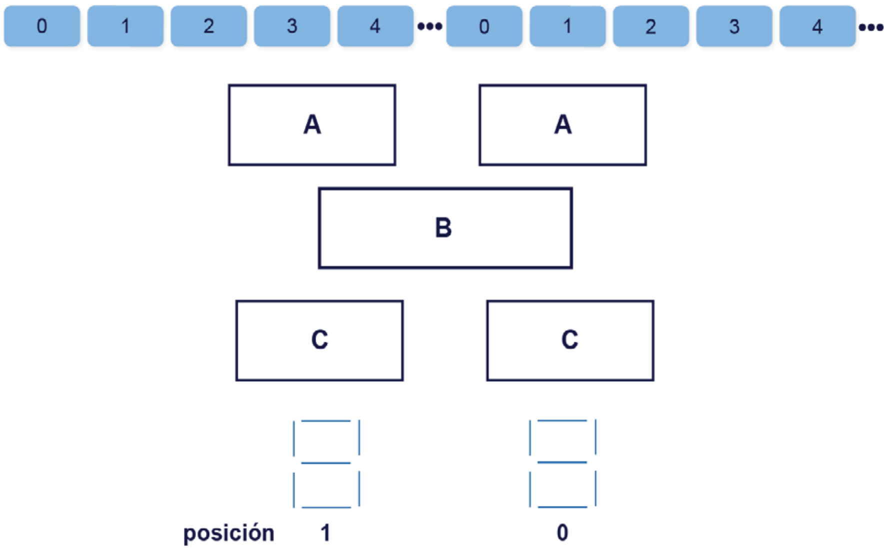


Figura 4.10

a) A la gráfica le faltan los conectores, agréguelos a este dibujo usando la forma:

Sustituya X por el número adecuado.

b) Diseñe en VHDL el componente A, tanto el puerto como la arquitectura. Utilice puntos suspensivos del *switch* 4 o 5 en adelante para que no codifique todo porque es mucho, solo una porción que sea representativa.

c) Diseñe el circuito esquemático del componente B, utilice sumadores de números de cuatro *bits*, comparadores, *multiplexers*, restadores.

©Editorial Digital Tecnológico de Monterrey

d) Diseñe en VHDL el circuito esquemático que diseñó en el inciso anterior.

e) Diseñe en VHDL el componente C, tanto el puerto como la arquitectura. Utilice puntos suspensivos para que no codifique todo porque es mucho, solo una porción que sea representativa.

f) Diseñe la arquitectura del componente completo en VHDL en nivel estructural, solo la arquitectura a partir del begin. No definir los "component" ni signals.

3. Un fabricante de máquinas despachadoras nos pide implementar el circuito de una máquina de café.

La máquina cuenta con un tablero de ocho *switches*, como se muestra a continuación:

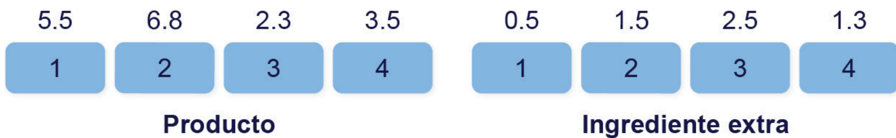


Figura 4.11

Como información al margen, los productos son:

1. Café capuchino

2. Café moka

3. Café expreso

4. Café americano

©Editorial Digital Tecnológico de Monterrey

Los ingredientes extras son:

1. Azúcar

2. Sustituto de azúcar

3. Crema

4. Leche descremada

El costo de cada opción se encuentra señalado en la parte superior de cada botón.

El usuario puede elegir un producto (usted decide qué hacer si elige dos o más, usted puede o no desplegar un mensaje de error).

Por otra parte, el usuario puede elegir solo de cero a cuatro ingredientes extras.

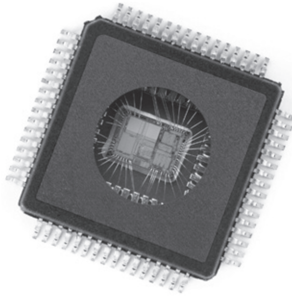
Su misión es… iluminar en *displays* el costo total (en decimal) de la elección del usuario. Puede iluminar el punto decimal con un LED.

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 4

Además de haber analizado la aritmética BCD, en este capítulo se introdujo la aplicación de los circuitos *encoder*s para interpretar información que proviene de botones que tienen algún significado asociado. De igual manera, se integraron los conocimientos y habilidades desarrolladas en los primeros capítulos, ya que los problemas requirieron el uso de los circuitos estudiados en los apartados previos. Este es el último capítulo dedicado a circuitos combinacionales, los siguientes se dedican a los circuitos secuenciales que tendrán un cambio significativo de enfoque, por lo que es importante dominar los primeros capítulos antes de avanzar al próximo.

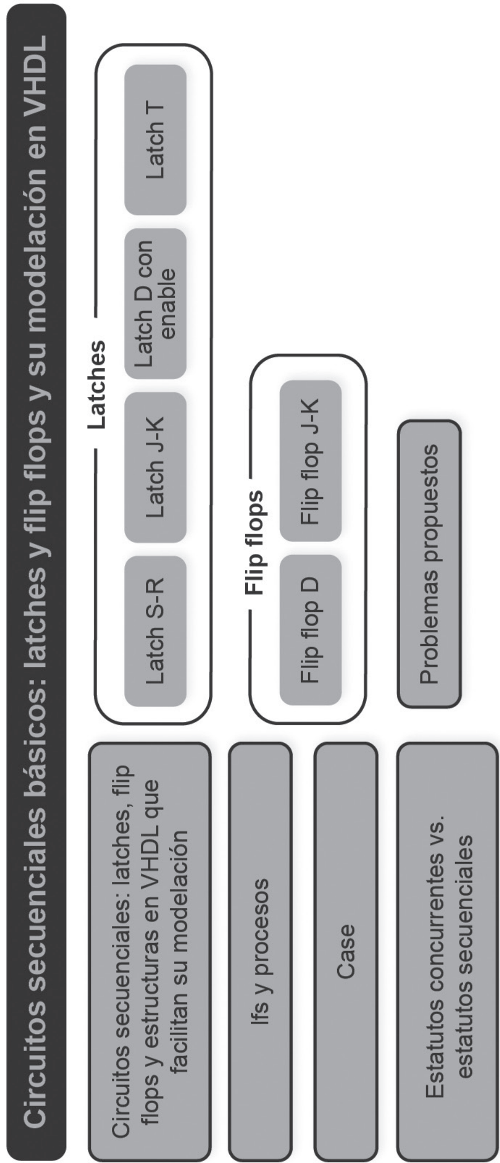


©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 5. Circuitos secuenciales básicos: latches, flip flops y su modelación en VHDL



©Editorial Digital Tecnológico de Monterrey

5.1 Introducción a los circuitos secuenciales

Cuando se requiere un circuito que presente una secuencia de operaciones, entonces el tipo de circuito a diseñar es el secuencial.

Un circuito secuencial se diferencia de uno combinacional porque en el circuito secuencial existe retroalimentación, es decir, una o más de las salidas del circuito son también entradas. La retroalimentación permite tomar en cuenta la salida anterior y de esa manera se logran establecer secuencias.

Figura 5.1

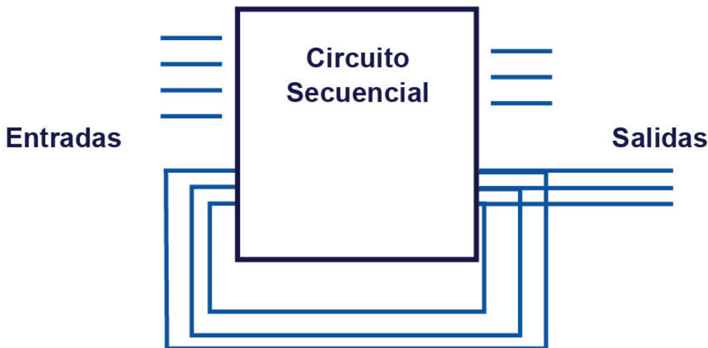


Figura 5.2

Cualquier circuito secuencial se construye valiéndose de un número reducido de circuitos comunes (tipificados con un nombre, y que existen en circuitos integrados TTL). Estos son: *latches*, *flip flops* y registros. Un elemento adicional que se requiere son las señales periódicas que se generan por circuitos osciladores (como el *timer 555*) que se utilizan para sincronizar, o bien, para establecer una frecuencia de operación.

©Editorial Digital Tecnológico de Monterrey

Para desarrollar la habilidad de construir circuitos secuenciales analizaremos diversos elementos, tanto para comprender su operación como para modelarlos. Este capítulo está dedicado a los *latches* y a los *flip flops*.

Se inicia este capítulo con las estructuras en **VHDL** que facilitarán la modelación del comportamiento de componentes secuenciales.

5.2 Estructuras en VHDL para el modelado de circuitos secuenciales

En el capítulo 2 se abordaron estructuras condicionales para la modelación de circuitos combinacionales. Ahora se cubrirán estas estructuras con mayor detalle para utilizarlas en la definición de circuitos secuenciales.

5.2.1 Estructura if y procesos

El lenguaje **VHDL** cuenta con opciones para modelar el comportamiento de un circuito a partir de su funcionamiento utilizando estructuras condicionales. Una de estas estructuras está dada por el estatuto **if**:

```
IF <condición> then <estatutos secuen-
ciales> elsif <condición> then <estatutos
estatutos secuenciales> … elsif <condi-
ción>  then <estatutos secuenciales> else
<estatutos secuenciales> end if
```

©Editorial Digital Tecnológico de Monterrey

Los estatutos secuenciales son: la asignación de una ecuación, el *IF* y el *CASE*.

Los estatutos se separan por el símbolo ";". Los ***elsif*** permiten anidar condiciones. Un ***if*** finaliza con las palabras ***end if***. Es opcional incluir condiciones anidadas. Se genera un *multiplexer* que contiene las compuertas lógicas requeridas para seleccionar las condiciones.

También es posible codificar condiciones anidadas con nuevos ***ifs***, sin utilizar la opción ***elsif***, pero esto requerirá cerrar con ***end if*** cada ***if*** que se haya abierto.

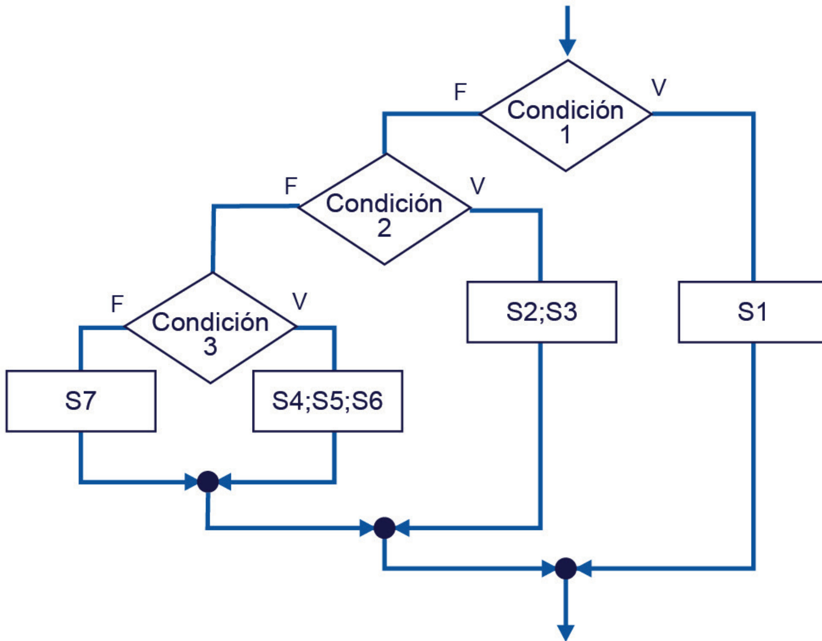


Figura 5.3

©Editorial Digital Tecnológico de Monterrey

Puede ser convertida a VHDL de dos formas:

a) Con elsif

```
If Condición1 then S1;
    Elsif Condición2 then S2;S3;
           Elsif Condición3 then S4;S5;S6;
                     Else S7;
End if;
```

b) Sin elsif

```
If Condición1 then S1;
    Else if Condición2 then S2;S3;
           Else if Condición3 then S4;S5;S6;
                     Else S7;
                  End if;
         End if;
End if;
```

Ambas opciones son equivalentes, aunque probablemente la primera tiene una construcción más sencilla.

Del entramado de condiciones se modela una sección del circuito. No se debe confundir la codificación de un algoritmo de programación en C o en Java, por ejemplo, con la descripción de un circuito. Con ***ifs*** se está separando en partes una red ***and-or*** del circuito; por un lado, están las condiciones y por otro, las salidas.

Aunque el estatuto ***if*** permite omitir la opción ***else***, es recomendable cerrar todas las opciones para que no queden condiciones indeterminadas. Si bajo alguna condición alguna señal debe quedar inalterada resulta equivalente indicar **señal<= señal**, o bien, ***null***.

©Editorial Digital Tecnológico de Monterrey

Otra estructura condicional utilizada dentro de un proceso es la estructura "CASE", la cual se explica a continuación.

Aunque el estatuto ***if*** permite omitir la opción ***else***, es sano cerrar todas las opciones para que no queden condiciones indeterminadas. Si alguna señal debe quedar inalterada resulta equivalente indicar **señal<= señal** y **null**.

La construcción de un circuito con ***ifs*** requiere ser incluida dentro de un ***process***. Hacerlo fuera de un ***process*** representaría un error sintáctico.

Process (**lista de señales que alteran de inmediato las salidas cuando tienen un cambio**)
begin
...
end process;

5.2.2 Estructura CASE

El estatuto *CASE* permite presentar un conjunto de condiciones de manera muy ordenada. Recordemos la modelación de un selector (**mux**) de dos entradas de control.

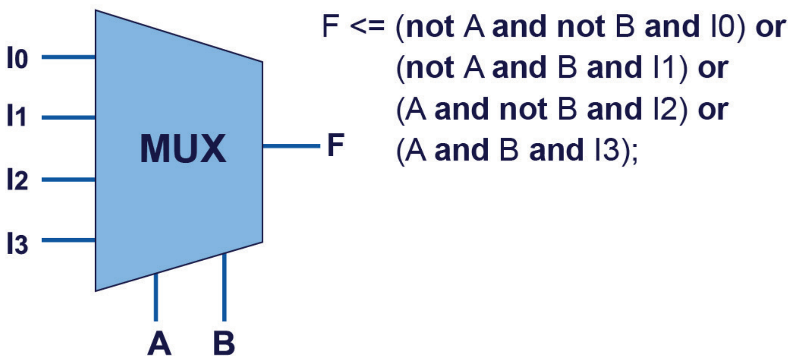


Figura 5.4

©Editorial Digital Tecnológico de Monterrey

La sintaxis general es la siguiente:

```
case señal is
when opción1 => estatutos secuenciales 1
when opción2 => estatutos secuenciales 2
. . .
[when others => estatutos secuenciales]
end case;
En términos de ifs, este case es equivalente a:

if señal = opción 1 then estatutos secuenciales 1;
elsif señal = opción 2 then estatutos secuenciales 2;
…
end if;
```

El requisito para modelar con *CASE* es que las condiciones estén planteadas para solo una señal, que puede ser un *bit* o un vector. Algunos ruteadores configuran mal las opciones del **when others**, por precaución es mejor dejar esa opción con **null** y cubrir en las opciones todos los casos que puedan ocurrir.

```
entity caso is
  Port ( Sel : in std_logic_vector(1 downto 0);
      I0 : in std_logic;
      I1 : in std_logic;
      I2 : in std_logic;
      I3 : in std_logic;
      F  : out std_logic);
end caso;
architecture Behavioral of caso is
begin
process (Sel,I0,I1,I2,I3)
```

©Editorial Digital Tecnológico de Monterrey

```
begin
case Sel is
  when "00" => F <= I0;
  when "01" => F <= I1;
  when "10" => F <= I2;
  when "11" => F <= I3;
  when others => null;
end case;
end process;
end Behavioral;
```

5.2.3 Estatutos secuenciales y estatutos concurrentes

Un **circuito digital** es un conjunto de interconexiones lógicas basadas en compuertas. Cada conexión une una, dos o más entradas y una salida, que a su vez puede ser una entrada a otra conexión.

Hemos llamado **componente** a un bloque lógico que tiene una **interfaz** de entradas y salidas, que en VHDL es llamada puerto, y una arquitectura que consta de circuitos.

En VHDL es posible definir los circuitos de los que consta un componente con expresiones lógicas que involucren una o varias compuertas. Es posible definir toda la red de conexiones que genera una salida final del componente en una sola expresión, o bien, es posible definir el componente por partes, describiendo las salidas intermedias y luego utilizándolas como entradas de otros subcircuitos. En caso de segmentar el circuito, el orden en que se describa no tiene importancia. Todos los segmentos (partes) de un circuito se definen por medio de instrucciones, que en VHDL se llaman **estatutos concurrentes**.

©Editorial Digital Tecnológico de Monterrey

```
architecture nivel_de_descripción of  nombre_circuito is
{signal señales_auxiliares : tipo;}
begin
estatuto concurrente 1;
estatuto concurrente 2;
.
.
.
estatuto concurrente n;
end nivel_de_descripción;
```

Un **estatuto** concurrente puede describir a un circuito combinacional o a un circuito secuencial. Los circuitos pueden estar aislados uno de otro, o pueden estar interconectados a través de sus señales.

Hay cuatro maneras de describir un circuito (que a su vez resume el tipo de *estatutos* concurrentes de las que se dispone en VHDL):

1. Asignaciones a señales de salida:

```
Señal <= expresión booleana
```

2. Asignación condicional:

```
Señal<= valor1 when expresión_condicional1 else
        valor2 when expresión_condicional2 else
   ... valorn
```

©Editorial Digital Tecnológico de Monterrey

Que es equivalente a un circuito and-or como el que sigue:

```
Señal<= valor1 and expresión_condicional1 or
        valor2 and expresión_condicional2 or
   ... valorn and expresión condicional complementaria
```

3. Proceso: En un proceso es posible describir uno o varios circuitos. Su sintaxis

```
Process(lista-sensitiva)
begin
estatuto secuencial 1;
estatuto secuencial 2;
.
.
.
estatuto secuencial n;
end process;
```

4. Port map: Una instancia de un componente. Esta opción ya se revisó y se utilizó en ejemplos. Es una manera de incorporar un componente estableciendo las conexiones a su puerto.

Por otra parte, los **estatutos secuenciales** son:

```
>> if condición then … elsif … then … elsif… end if
>> case
>> Señal <= expresión booleana
```

Note que **Señal ← expresión booleana** es tanto un estatuto concurrente como secuencial. Se debe considerar que las instrucciones secuenciales son concurrentes entre sí.

Con esta introducción se tienen las estructuras necesarias de modelación en VHDL para iniciar la definición de circuitos secuenciales.

©Editorial Digital Tecnológico de Monterrey

5.3 Circuitos almacenadores (*latches*)

Un ***latch*** es la unidad mínima de memoria. Un ***latch*** es un circuito que "almacena" un *bit*. Existe una variedad de ***latches*** que realizan funciones comunes de almacenamiento que existen encapsulados en circuitos TTL. Estos ***latches*** "comerciales" prácticamente cubren todas las funciones requeridas para el almacenamiento de un *bit*. Sin embargo, al describir un circuito en VHDL no es necesario restringirse a un circuito común, ya que usando VHDL el circuito se configura con compuertas, no con circuitos conocidos, así que se puede especificar cualquiera. Más adelante se presentan ejercicios para especificar *latches* que no existen como circuitos integrados.

Como ejercicio introductorio, para conocer los *latches* comunes se recomienda ubicar los números de parte de circuitos **TTL** que corresponden a ***latches*** "comerciales" (que se venden en circuitos integrados). Todos los ***latches*** comerciales tienen una o dos entradas, pueden o no tener ***enable*** y tienen dos salidas, una llamada **Q** que es la principal, y otra que es la misma salida **Q** pero complementada.

La salida **Q'** apoya el logro de la salida **Q**, como se muestra en el primer ***latch*** a analizar, cuya arquitectura interna se muestra a continuación.

5.3.1 Latch S-R

Un circuito muy simple que consigue con solo dos compuertas el comportamiento de un *latch*, es el siguiente:

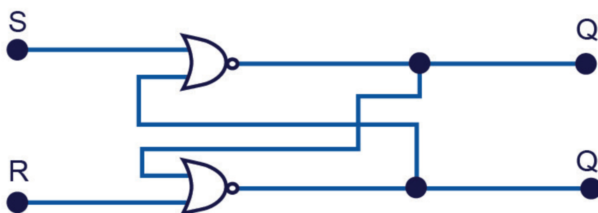


Figura 5.5 Circuito de latch S-R

©Editorial Digital Tecnológico de Monterrey

Este circuito es conocido como ***latch*** **S-R**. La **S** abrevia "***set***" y la **R** "***reset***". El circuito esquemático (en bloque, sin detallar la arquitectura interna) es:

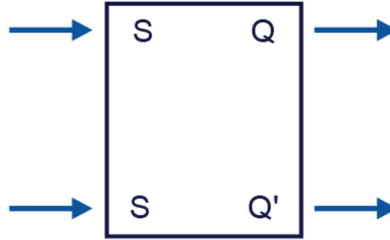


Figura 5.6 Bloque esquemático de latch S-R

No es común dibujar el sentido de las líneas de entrada o salida. En este caso se hizo para enfatizar cuáles son entradas y cuáles son salidas.

Al analizar este tipo de circuitos, necesitamos suponer un estado inicial para Q que pasará a un nuevo estado $\mathbf{Q^+}$. El nuevo valor de la salida ($\mathbf{Q^+}$) se obtiene en el delta de tiempo de retraso de las compuertas que intervengan en el circuito, que en este caso es una.

En narrativa, el funcionamiento de este circuito es: cuando ingresa un **1** por la entrada **S**, la salida **Q** se vuelve **1**. Aunque **S** pase a un valor de **0**, la salida **Q** sigue en **1**. Cuando por la entrada **R** ingresa un **1**, la salida **Q** se va a **0** y permanece en **0** hasta que ocurra de nuevo una entrada de **1** por **S**.

Gráficamente el funcionamiento del ***latch*** **S-R** es (mostrando su funcionamiento con valores lógicos de ejemplo):

©Editorial Digital Tecnológico de Monterrey

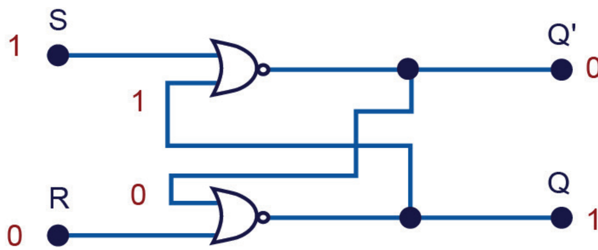


Figura 5.7 Funcionamiento de latch S-R

No importa si la salida **Q** está en **0** o en **1** (usted lo puede comprobar, ya que (1+0)'=(1+1)'=0) el valor final de **Q'** es **0** y eso ocasiona que **(0+0)'=1**.

A continuación, se muestra que con la entrada **R** en **1** la salida **Q** queda en **0**, sin importar si anteriormente era **0** o **1**, como podrá usted comprobar al seguir el circuito hasta que se estabilice, es decir, que ya no cambie su salida.

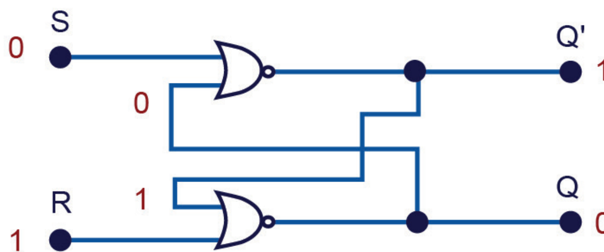


Figura 5.8 Funcionamiento de latch S-R

También es posible comprobar que con ceros en ambas entradas la salida permanece igual.

©Editorial Digital Tecnológico de Monterrey

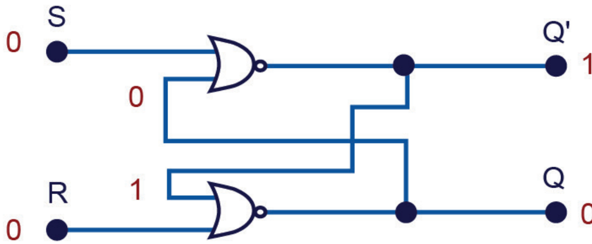


Figura 5.9 Funcionamiento de latch S-R

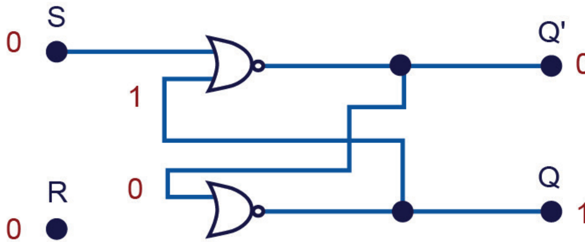


Figura 5.10 Funcionamiento de latch S-R

El problema con este circuito es que con las entradas **S=1** y **R=1** ambas salidas **Q** y **Q'** se vuelven **0** y a continuación las salidas se vuelven inestables, es decir, se quedan fluctuando entre **0** y **1**.

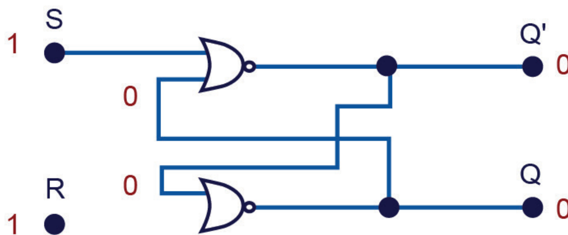


Figura 5.11 Funcionamiento de latch S-R

©Editorial Digital Tecnológico de Monterrey

En resumen, la tabla de verdad de este *latch* es:

S	R	Q	Q^{+}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	NO
1	1	1	NO

Tabla 5.1 Tabla de verdad de latch S-R

En VHDL, la modelación de este *latch* a partir de su circuito basado en compuertas, es:

Q<=R nor Qn;

Qn<=S nor Q;

Resulta complicado describir el comportamiento de este circuito en VHDL porque habría que separar la condición de que haya ocurrido la entrada **1 1** y que las salidas sean **0 0**.

Cabe señalar que un circuito similar se consigue con compuertas **nand**. En este caso, las entradas indeseables son las **0 0**.

5.3.2 Latch J-K

Este es un ***latch*** común con muchas aplicaciones. En el diseño de este *latch* se corrige el inconveniente del ***latch*** **S-R**, aunque con un mayor número de compuertas.

©Editorial Digital Tecnológico de Monterrey

En este latch la entrada **J** actúa como un **set**, es decir, elevar a **1** la salida **Q**, y la entrada **K** cuya función es bajar a **0** la salida **Q**. Cuando ambas entradas están en **0** la salida sigue igual, y cuando ambas están en **1** la salida se invierte. El dibujo esquemático con el que se representa este *latch* es:

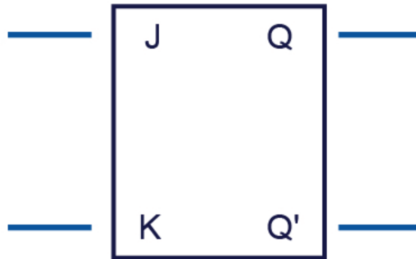


Figura 5.12 Bloque esquemático del latch J-K

El comportamiento que se ha descrito, detallado en una tabla de verdad, es:

J	K	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Tabla 5.2 Tabla de verdad de latch J-K

©Editorial Digital Tecnológico de Monterrey

La expresión resultante ya simplificada es:

Q+=JQ'+K'Q

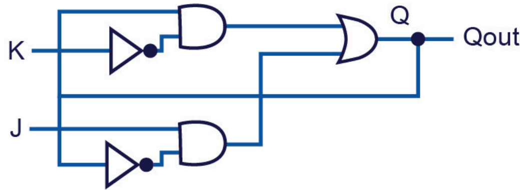


Figura 5.13 Circuito del latch J-K

Los nombres **Qout** y **Q** corresponden a la misma señal, pero se indican dos nombres para dar el sentido que **Q** se retroalimenta, entonces es una señal **inout**, y **Qout** es solo salida.

La descripción en VHDL es:

a) Por la expresión resultante:

```
Q <=( J and not Q) or (not K and Q);
Qout<=Q; --opcional, solo en caso que
se requiera una señal sólo de salida
```

b) La descripción por su comportamiento resulta extensa, y esta puede ser:

©Editorial Digital Tecnológico de Monterrey

```
Process(J,K)
Begin
 If (J='0') and (K='0') then null;
Elsif  (J='0') and (K='1') then Q<='0';
Elsif  (J='1') and (K='0') then Q<='1';
Elsif  (J='1') and (K='1') then Q<=not Q;
Else null;
 End if;
```

Este *latch* funciona bien excepto con las entradas puestas en J=1 y K=1, en este caso las salidas no se estabilizan porque se niegan constantemente, de hecho, no es posible simular este circuito. Para simularlo habrá que cambiar la condición cuando las entradas son 1-1, eliminar que se niegue la salida y dejar que se quede igual.

5.3.3 Latch D con enable

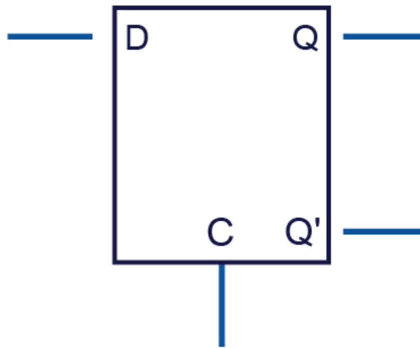


Figura 5.14 Bloque esquemático del latch D con enable

Este circuito tiene una entrada **D** y una entrada de control **C** para habilitar que se almacene la entrada. La salida **Q** refleja a la entrada **D** cuando la entrada *enable* **C** es **1**, si **C** es **0** la salida **Q** permanece inalterada. Esta misma descripción descrita en una tabla de verdad, es la siguiente:

©Editorial Digital Tecnológico de Monterrey

J	K	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Tabla 5.3 Tabla de verdad del latch D

Cabe recordar que la señal **Q** y **Q^+** es la misma señal, en la tabla se diferencian para hacer notar que **Q** es la salida actual y **Q^+** es la siguiente salida. **Q^+** obtenida de la tabla de verdad resulta:

```
Q += C'D'Q + C'DQ + CDQ' + CDQ =C'Q + CD
```

La expresión simplificada refleja la condición lógica, si **C=0** la salida es **Q**, si **C=1** la salida es **D**.

El circuito es:

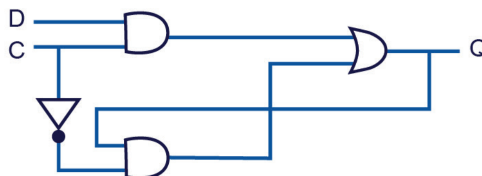


Figura 5.15 Circuito del latch D con enable

©Editorial Digital Tecnológico de Monterrey

En VHDL hay dos formas principales para describir este circuito, una es utilizando la expresión booleana, simplificada o sin simplificar. Aquí se muestra simplificada:

Q=<(NOT C AND Q) OR (C AND D);

La otra forma describe el *latch* a partir de su comportamiento:

```
Process(C,D)
Begin
If C=’1’ then
     Q<=D;
     Else
     Null;
End if;
End process;
```

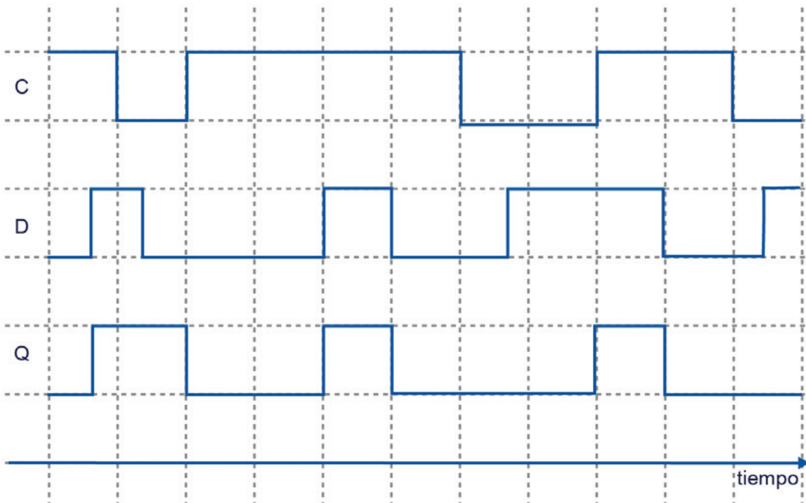


Figura 5.16 Gráfica de tiempo del funcionamiento del latch D con enable

El comportamiento de este circuito se observa con mejor precisión con esta gráfica, cuando **C** es **1**, la salida **Q** refleja a la entrada **D**, cuando **C** es **0**, **Q** refleja el último valor que se almacenó en

©Editorial Digital Tecnológico de Monterrey

C=1. La respuesta **Q** es inmediata, solo tiene el retraso del circuito combinacional.

Lo que se debe apreciar en la gráfica es que, mientras **C** está en **1**, **Q** refleja el valor de **D** y, cuando **C=0**, **Q** permanece igual. Esta es la gráfica que produce tanto el circuito como cualquiera de los dos modelos hechos en VHDL. En el caso del modelo hecho con la expresión booleana, los cambios en **C** o **D** reflejarán un cambio en **Q**. En el caso del modelo hecho con un proceso, un cambio en **C** o **D** provocan que se active el proceso (si se fuera a simular) y este es el artificio de VHDL para la simulación de un proceso, su contenido se evalúa cada vez que hay cambios en las señales incluidas en la lista de sensibilidad. Para este *latch* puede haber cambios en la salida cuando hay cambios en la entrada **D** o en la entrada **C**.

5.3.4 Latch T

El ***latch*** **T** es otra forma de memoria de un *bit*. Es un circuito conocido, encapsulado y comercial. La **T** viene de toggle. Esta palabra no tiene una palabra equivalente en español. La palabra más cercana podría ser conmutar, pero la traducción más precisa sería alternar entre dos estados. Con el comportamiento del *latch* T se precisa cómo funciona: cuando la entrada **T** es **0** la salida permanece igual, cuando es **1** la salida se invierte. En una tabla de verdad, el *latch* T se describe así:

T	Q	Q^+
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 5.4 Comportamiento latch T

©Editorial Digital Tecnológico de Monterrey

Hay dos modelos básicos en VHDL de este circuito, el que se describe con su ecuación:

```
T<=T Xor Q;
```

Y el que se describe por su comportamiento:

```
Process (T)
Begin
If T='1' then Q<=not Q;
        Else null;
End if;
End process;
```

Responde la siguiente actividad.

Diseñe un nuevo *latch* S-R, al que llamaremos A-B, con la siguiente descripción (este no es un circuito comercial): las entradas son A, B y salida Q. Su funcionamiento es el siguiente: que ponga en 1 su salida cuando A=1 y en 0 cuando B=1. Si A=0 y B=0 o A=1 y B=1, la salida permanece igual.

©Editorial Digital Tecnológico de Monterrey

5.4 Flip flops

En un ***latch***, la entrada se almacena de inmediato. Existe otro tipo de circuito, llamado ***flip flop***, en el que la entrada se almacena en el momento en que otra entrada adicional, a la que llamaremos entrada de reloj, sufre un cambio. Algunos *flip flops* son sensibles a un cambio positivo, es decir, de **0** a **1**. Otros a un cambio negativo, es decir, de **1** a **0**. Esta operación tiene muchas aplicaciones y permite sincronizar circuitos, ya que, conectando la misma señal de reloj a múltiples circuitos, es posible que todos registren sus entradas o sus cambios al mismo tiempo. La construcción de un *flip flop* con la conexión de dos *latches* se explicará utilizando dos ***latches*** **D** con ***enable*** **C**.

5.4.1 Flip flop D

De la sección anterior se estudió que el ***latch*** **D** con entrada habilitadora **C** se modela con la ecuación:

```
Q<=C'Q + CD
```

Si se conectan dos *latches* D de este tipo en cascada, sería de la siguiente manera:

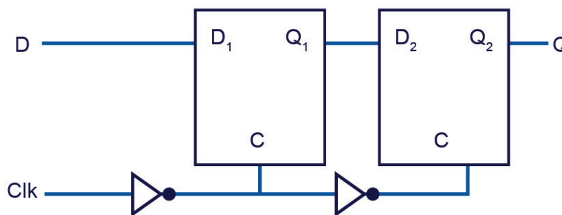


Figura 5.17 Circuito interno del flip flop D

©Editorial Digital Tecnológico de Monterrey

Se observará que cuando la señal llamada **Clk** esté en **0**, el primer ***latch*** memorizará la entrada **D**, al momento en que **CLk** suba a **1**. Lo último que se haya registrado en **Q1** antes de que **CLk** subiera a **1**, se registrará en la salida del segundo ***latch***. Esta es la actuación de un componente llamado ***flip flop***.

Este es un componente conocido, encapsulado y comercial e indispensable para la construcción de registros y de la sincronización de eventos.

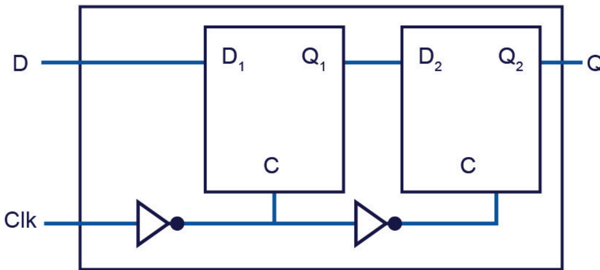


Figura 5.18 Circuito interno del *flip flop* D

Y se conoce con un esquemático específico, que es el siguiente:

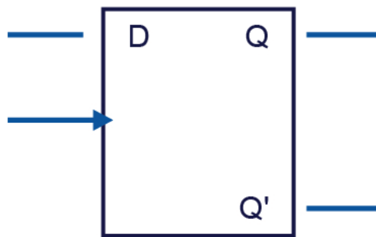


Figura 5.19 Dibujo esquemático del flip flop D

Con un triángulo se distingue la señal a la que se distinguirá su transición. El comportamiento de este *flip flop* D se muestra en la figura 5.20.

©Editorial Digital Tecnológico de Monterrey

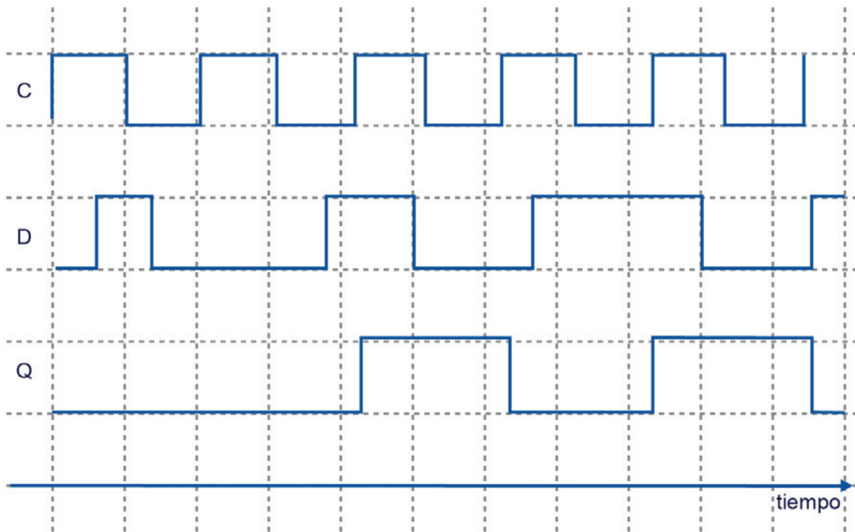


Figura 5.20 Diagrama de tiempo del flip flop D

Tal como se puede observar en esta figura, cada vez que hay una transición de **0** a **1** de la señal **Clk**, se registra la señal **D** en **Q** (el valor que tenga **D** justo antes de la transición). Por lo tanto, los cambios de **Q** se efectúan solo en las transiciones de **CLk**.

Este componente se modela con las ecuaciones **Q1<=Clk Q1 + Clk'D**; **Q<= Clk'Q + ClkQ1** que describe la conexión de los dos ***latches*** conectados en configuración maestro-esclavo, pero esta manera de describir el ***flip flop*** resulta ser un poco complicada. Lo afortunado en VHDL es que un ***flip flop*** **D**, que responde a la transición positiva de un reloj C, puede ser definido en diferentes niveles, entre ellos su comportamiento. La definición de un *flip flop* D puede ser descrita a través de su comportamiento utilizando la estructura ***process***. De hecho, hay descripciones que podemos considerar como **plantillas** (o patrones) que son reconocidas por el sintetizador.

©Editorial Digital Tecnológico de Monterrey

Por ejemplo, para el *flip flop* D la plantilla es:

```
process (Clk);
begin
 if Clk'event and (Clk='1') then
     Q<= D;
end if;
end process;
```

En VHDL el sufijo **'event** se añade a una señal cuyas transiciones se desean reconocer. **'event** indica tanto al simulador como al sintetizador la ocurrencia de una transición de la señal que le precede, y se utiliza tanto una transición positiva como negativa. Para distinguir si la transición que se va a considerar es positiva o negativa se incluye la condición **Clk='1'** o bien **Clk='0'**. La señal puede tener cualquier nombre.

Por otra parte, se ha dicho que en la lista sensitiva del *process* se deben incluir todas las entradas del circuito que describe.

```
process (Clk, D);
begin
 if Clk'event and (Clk='1') then
     Q<= D;
end if;
end process;
```

Es una buena práctica incluir la opción else null; sin embargo, el sintetizador permite reconocer el *flip flop* D sin incluir esa opción.

Así que esta es la descripción completa. No obstante, la descripción anterior en la que en la lista solo se encuentra **Clk** y no **D** también se reconoce y es la que es considerada como la plantilla del ***flip flop* D**. Lo que podemos enfatizar es que las entradas que se incluyen en la lista son las que pueden provocar un cambio en la

©Editorial Digital Tecnológico de Monterrey

salida cuando presentan una variación. Bajo esta perspectiva revisemos de nuevo la gráfica y constatemos que la salida cambia cuando hay un cambio en **Clk** y no un cambio en **D**, por eso, la entrada que se considera crítica, o bien la salida, es sensible a **Clk** y no a **D**. Cualquiera de las dos descripciones es correcta.

```
Started process "Synthesize".

================== * HDL Compilation * ==================

Compiling vhdl file C:/Xilinx/bin/modulo2/ffd.vhd in Library work.
Entity <ffd> (Architecture <Behavioral>) compiled.

================== * HDL Analysis * ==================

Analyzing Entity <ffd> (Architecture <Behavioral>).
Entity <ffd> analyzed. Unit <ffd> generated.

================== * HDL Synthesis * ==================

Synthesizing Unit <ffd>. Related source file is C:/Xilinx/bin/modulo2/ffd.vhd.
Found 1-bit register for signal <Q>.
Summary: inferred 1 D-type flip-flop(s).

Unit <ffd> synthesized.

=====================================

HDL Synthesis Report Macro Statistics # Registers : 1 1-bit register : 1
```

©Editorial Digital Tecnológico de Monterrey

```
process (Clk, D);
begin
 if Clk'event and (Clk='1') then
    Q<= D;
end if;
end process;
```

Otra forma de definir un *flip flop* D a nivel comportamiento es:

```
Q<=D when (Clk'event and (Clk='1')) else Q;
```

Aunque esta no es la plantilla y el sintetizador no reconoce esta descripción, el simulador sí.

Existen *flip flops* encapsulados comerciales de todos los tipos de *latches*, y su construcción interna lleva la misma filosofía: una conexión *maestro-esclavo*. En todos los casos, la descripción más sencilla es la conexión por comportamiento utilizando un *process*.

A continuación, nos enfocaremos en la descripción de un ***flip flop*** **JK**, ya que es un componente útil y común.

5.4.2 Flip flop JK

El dibujo esquemático del ***flip flop*** **JK** es:

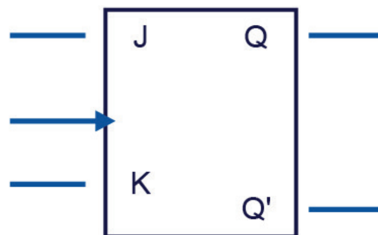


Figura 5.21 Bloque esquemático del flip flop J-K

©Editorial Digital Tecnológico de Monterrey

Su descripción por comportamiento en VHDL si la entrada de la señal de reloj es **Clk** es:

```
Process(Clk)
Begin
If Clk='1' and Clk'event then
  If (J='0') and (K='0') then null;
     Elsif  (J='0') and (K='1') then Q<='0';
     Elsif  (J='1') and (K='0') then Q<='1';
     Elsif  (J='1') and (K='1') then Q<=not Q;
     Else null;
   End if;
  Else null;
 End if;
End process;
```

Otra opción más simple, pero que requiere recordar la ecuación booleana es:

```
Process(Clk)
Begin
If Clk='1' and Clk'event then
Q <=( J and not Q) or (not K and Q); ***
Else null;
 End if;
End process;
```

En la lista sensible no se requiere J y K porque las salidas se renuevan con cambios de Clk y no con cambios de J y K, así que esta es la plantilla completa.

©Editorial Digital Tecnológico de Monterrey

5.4.3 Flip flop JK con entradas de preset y clear asincrónicas

Ahora analizaremos otro tipo de ***flip flop*** **JK**. También es conocido y existe como circuito integrado **TTL** comercial, cuenta con dos entradas extras, una llamada **PRE'** (de *preset* que se activa en cero) y otra **CLR'** (de *clear* que se activa en cero). En el momento en que ***preset*** baja a **0**, la salida **Q** sube a **1**. En el momento en que ***clear*** baja a **0**, la salida **Q** baja a **0**. Como ***preset*** y ***clear*** no dependen de la señal de reloj para efectuar su función, se dice que son funciones asincrónicas.

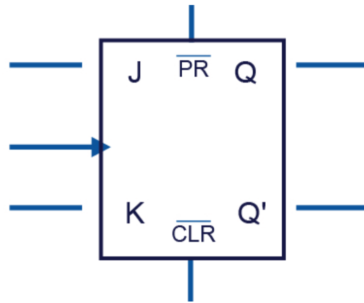


Figura 5.22 Bloque esquemático del *flip flop* J-K con *clear* y *preset*

El circuito interno de este *flip flop* es el siguiente:

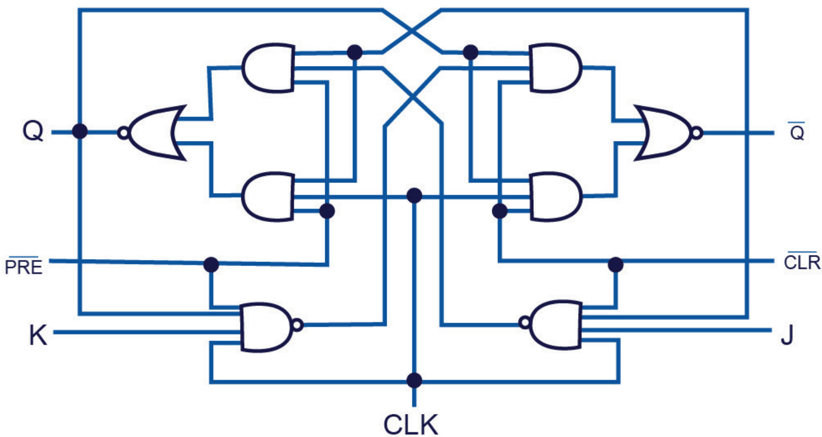


Figura 5.23 Circuito interno del *flip flop* J-K

©Editorial Digital Tecnológico de Monterrey

Usted puede analizar esta arquitectura y observará que su comportamiento es el siguiente:

Entradas					Salidas		Función
$\overline{CLR}$	$\overline{PRE}$	J	K	CLK	Q	Q	
0	0	X	X	X	1	1	-
0	1	X	X	X	0	1	Clear
1	0	X	X	X	1	0	Preset
1	1	0	0	Transición negativa	Q_0	Q_0	Sin cambios
1	1	0	1	Transición negativa	0	1	-
1	1	1	0	Transición negativa	1	0	-
1	1	1	1	Transición negativa	Q_0	Q_0	Conmutación
1	1	X	X	1	Q_0	Q_0	Sin cambio

Tabla 5.5 Tabla de verdad del flip flop J-K con entradas preset y clear

©Editorial Digital Tecnológico de Monterrey

El ***preset*** y el ***set*** son como señales de reloj alternas que fuerzan a la salida a cambiar de estado. Para modelar este circuito en VHDL no es necesario describir esta arquitectura, ya que es posible hacerlo por su comportamiento; tan solo es necesario seguir un patrón de modelación que se describe a continuación:

Para modelar este *flip flop* y cualquier otro circuito secuencial, es preciso seguir las siguientes reglas:

1) Primero modelar las funciones asincrónicas.

2) A continuación, modelar las funciones sincrónicas utilizando solo una expresión condicional que haga referencia al reloj.

Siguiendo estas reglas, la modelación queda como sigue, tomando en cuenta que el circuito muestra una señal de reloj negativa.

```
Process(PRE,CLR,Clk)
Begin
  If PRE='0' then Q<='1';
     Elsif CLR='0' then Q<='0';
        Elsif Clk='0' and Clk'event then
     Q <=( J and not Q) or (not K and Q);
else null;
end if;
end process;
```

Para finalizar este capítulo, a continuación, se proponen ejercicios para modelar latches y flip flops.

5.4.4. Flip flop T

Anteriormente en este capítulo se describió un latch T, ahora se define como *flip flop*, respondiendo a la transición positiva de la señal de reloj y con dos entradas asíncronas: PRE y CLR, a las que se responde cuando sean 1.

©Editorial Digital Tecnológico de Monterrey

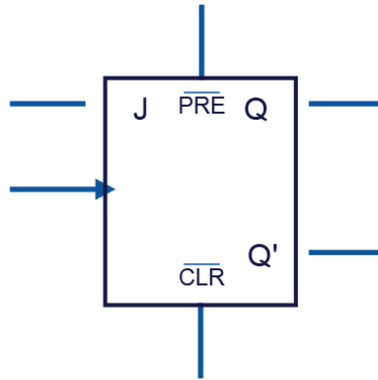


Figura 5.24 Bloque esquemático del flip flop T

Siguiendo las reglas de descripción de señales sincrónicas y asincrónicas y considerando que un *flip flop* T (que es un circuito comercial conocido) cambia la salida cuando la entrada T es 1, de otra manera permanece igual, se describe en VHDL de la siguiente manera:

```
Entity ffT is
Port (T, PRE, CLR, clk: in std_logic;
          Qout: out std_logic);
end ffT;
architechture comportamiento of ffT is
signal Q:std_logic:='0';
begin
process (T, PRE, CLR, clk)
begin
if PRE='1' then Q<='1';
    elsif CLR='1' then Q<='0';
    elsif clk='1' and clk'event then
          if T='1' then Q<=not Q;
                   else null;
          end if;
     else null;
end if;
Qout<=Q;
end process;
End ecuacion;
```

©Editorial Digital Tecnológico de Monterrey

Se agregó la inicialización de Q

```
signal Q:std_logic:='0';
```

para que este componente pueda ser simulado, porque de otra manera esta salida quedaría como *undefined*, a menos que se diera un 1 al *preset* o al *clear*, lo cual ya definiría la salida Qout.

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 5

Para los siguientes problemas, se requiere la definición de un *latch* J-K.

```
Entity latchJK is
Port (J,K:in std_logic;
        Qout:out std_logic;
End latchJK;
Architecture ecuacion of latchJK is
Signal Q:std_logic;
begin
Q <=(J and not Q) or (not K and Q);
Qout<=Q;
End ecuacion;
```

1. Diseñar un circuito al cual ingresen las señales generadas por dos botones, uno de Start y otro de Stop.

Cuando *Start* se oprima se deberá iluminar un LED y permanecer iluminado, aunque *Start* se suelte. El LED se apagará cuando se oprima *Stop* y permanecerá apagado hasta que de nuevo se oprima *Start*.

Tome en cuenta que un botón es un dispositivo de entrada física y funcionalmente diferente a un interruptor ya que un interruptor permanece físicamente en la misma posición a menos que deliberadamente se cambie y un botón se oprime y al soltarlo vuelve a la posición original.

En realidad, al hacer contacto un botón sufre cambios de estados repetitivos de 0 a 1 hasta que queda estable, a esto se le llama rebote. En este apartado consideraremos un botón ideal (sin rebote).

©Editorial Digital Tecnológico de Monterrey

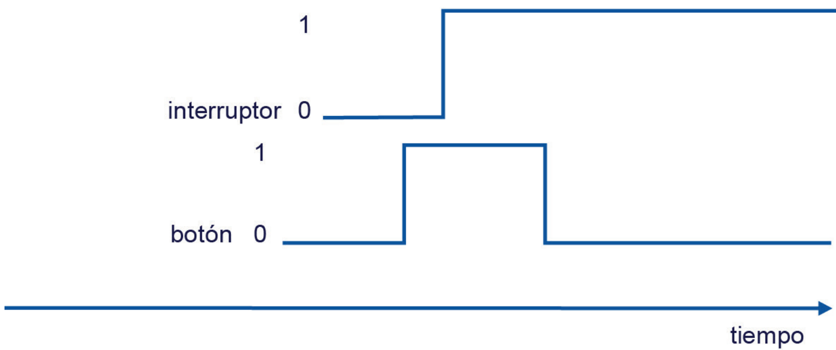


Figura 5.25

2. El *latch* T solo tiene una entrada, T, que cuando es 0 la salida permanece igual y cuando es 1 se invierte.

Aunque el *latch* T es un circuito comercial, se podría conseguir su funcionamiento con un J-K, ¿cómo?

Complete la conexión.

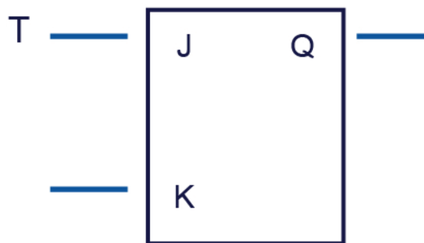


Figura 5.26

©Editorial Digital Tecnológico de Monterrey

3. El *latch* D solo tiene una entrada, D.

Cuando el circuito está habilitado, C=1, si D es 0 la salida permanece en cero y cuando D es 1 en 1. Si C=0 la salida permanece igual.

Aunque el *latch* D es un circuito comercial, se podría conseguir con un J-K, ¿cómo? Complete la conexión.

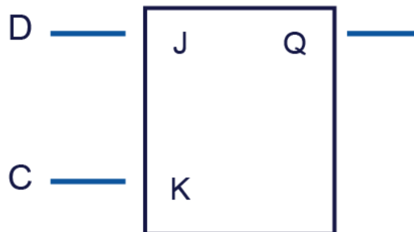


Figura 5.27

4. Diseñar un circuito al cual ingresen las señales generadas por un botón, un *Start/Stop*. Cuando *Start/Stop* se oprima se deberá iluminar un LED y permanecer iluminado, aunque *Start/Stop* se suelte. El LED se apagará cuando se oprima *Start/Stop* de nuevo y permanecerá apagado hasta que de otra vez se oprima el botón.

5. Diseñe la arquitectura interna (diseño esquemático), basada en compuertas lógicas, de un flip flop T que responda tanto a las transiciones positivas como a las negativas de una señal de reloj.

6. Diseñe la arquitectura interna (diseño esquemático), basada en compuertas lógicas, de un flip flop D que tenga dos entradas de reloj y que responda a la transición positiva de ambas (cualquiera de las dos).

©Editorial Digital Tecnológico de Monterrey

7. Para el siguiente circuito (formado por *latches* D con enable C):

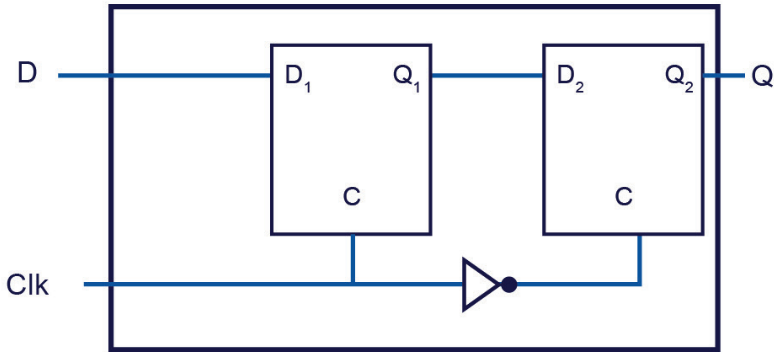


Figura 5.28

Dibuje en el siguiente diagrama de tiempo la salida Q. Suponga que en tiempo 0 la salida Q es 0.

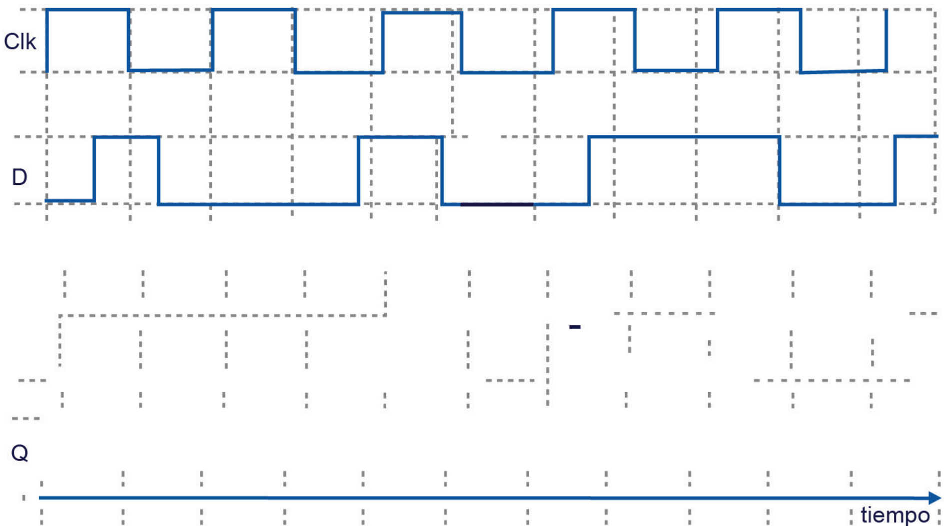
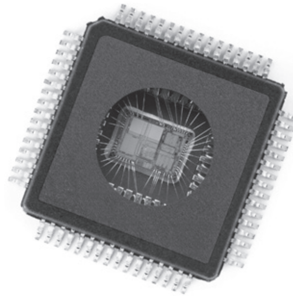


Figura 5.29

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 5

Un *latch* es la unidad mínima de memoria basada en electrónica digital, es un circuito que almacena un *bit*. Por su parte, los flips flops son los circuitos en los que se basan los circuitos secuenciales. Los circuitos que ejecutan algoritmos y sucesiones de pasos que toman decisiones lógicas, requieren *flip flop*s. Con *flip flop*s se construyen registros con funciones que facilitan el diseño de circuitos secuenciales y autómatas que se diseñan con flips flops individuales o en conjunto, es decir, con registros. Ya sea que se trate de un circuito con una continuación de pocos pasos, o de uno con secuencias complejas como son los microprocesadores o CPUs, su base de diseño es la misma: los *flip flop*s.



©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 6. Registros

Registros	
¿Qué es un registro?	Ejemplos de modelación de registros por sus funciones
Registros y su operación	Definición de un circuito que contiene más de un registro
Arquitectura interna de una muestra de registros	Ejemplos de un circuito que contiene más de un registro
Modelación de registros por sus funciones	Modelación incorrecta

El componente básico de un circuito secuencial es el registro. En este capítulo 6 se explica el diseño de los registros más comúnmente utilizados.

©Editorial Digital Tecnológico de Monterrey

6.1 ¿Qué es un registro?

Un registro es un conjunto de *flip flops* interconectados que realizan operaciones de carga de datos, almacenamiento o presentan en su salida una secuencia determinada, como es el caso de los contadores o de los registros internos de las máquinas de estados.

Una unidad central de procesamiento (CPU por sus siglas en inglés) cuenta con un conjunto de registros:

- El registro acumulador (AC),
- El registro contador de programa (PC),
- El registro de instrucción (IR),
- El registro de dirección de memoria (MAR),
- El registro de datos de memoria (MDR) y
- El registro de banderas (FR).

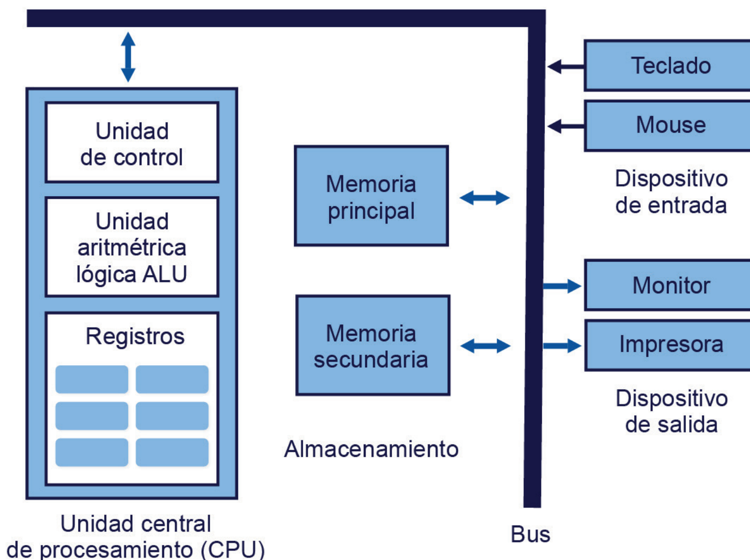


Figura 6.1 Diseño básico de un sistema computacional

©Editorial Digital Tecnológico de Monterrey

Probablemente el concepto que usted tenga de registro provenga de la arquitectura de Von Neumann.

Los registros son la base del diseño de los circuitos secuenciales. Como se mencionó, el diseño básico de un registro consta de un conjunto de *flip flops* de cualquier tipo y entradas de control que se conectan con lógica combinacional para lograr realizar alguna función.

Existen registros en circuitos integrados TTL, algunos solo realizan una función mientras que otros son multifuncionales. En general, las operaciones que pueden realizar los registros son:

- Carga paralela sincrónica
- Carga paralela asincrónica
- Carga serial por el bit más significativo con corrimiento a la derecha
- Carga serial por el bit menos significativo con corrimiento a la izquierda
- Corrimiento aritmético a la derecha (se repite al ingreso el bit más significativo)
- Corrimiento lógico a la derecha y a la izquierda (ingresa un cero)
- Incrementar
- Decrementar
- Llevar sus salidas a cero (*reset*) esta operación puede ser sincrónica o asincrónica.

Cabe resaltar que un registro puede realizar solo una operación en un pulso de reloj, o bien, una operación asincrónica de carga o de *reset*, una a la vez. Los registros requieren señales de control (con sus correspondientes entradas) para habilitar sus distintas funciones.

©Editorial Digital Tecnológico de Monterrey

Como el propósito principal del libro es utilizar VHDL para diseñar circuitos digitales, y en VHDL es posible definir registros a partir de su comportamiento, entonces no nos enfocaremos en diseñar la arquitectura interna de los registros basada en *flip flops*; sin embargo, sí se mostrarán algunos ejemplos básicos para tomar una idea de su diseño.

A continuación, se muestran los tipos de registro más comunes y su funcionamiento. Existen circuitos integrados con estos componentes. A continuación, se mostrará el diseño interno de algunos y su definición en VHDL.

©Editorial Digital Tecnológico de Monterrey

6.2 Registros y su operación

6.2.1 Registro de carga paralela sincrónica y salida paralela

Primero se muestra su funcionamiento ilustrándolo en algunos pasos en los que los registros interactúan con entradas y transforman sus salidas.

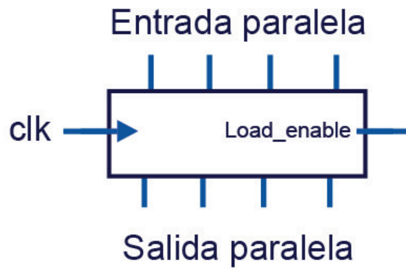


Figura 6.2 Funcionamiento de un registro de carga paralela

En este registro, mientras se le ingrese un 0 a la entrada de control *load enable* la salida continuará inalterada, si es ceros continúa en ceros.

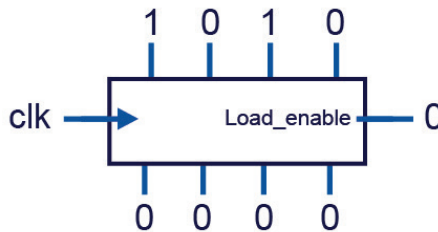


Figura 6.3 Funcionamiento de un registro de carga paralela

Si la entrada de control es 1, cuando ocurra la transición positiva de la señal clk entonces se registrará el dato que se encuentre en la entrada, reflejándose en la salida.

©Editorial Digital Tecnológico de Monterrey

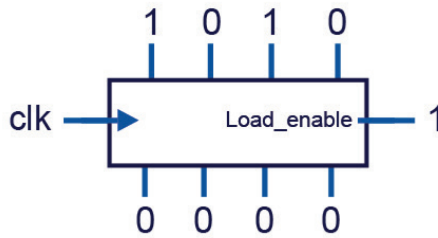


Figura 6.4 Funcionamiento de un registro de carga paralela

Al ocurrir la transición positiva de la señal clk ocurre lo siguiente:

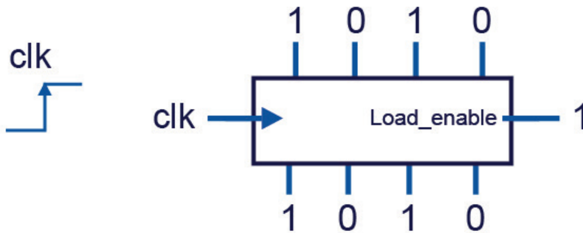


Figura 6.5 Funcionamiento de un registro de carga paralela

Si a continuación ocurre una transición de la señal de reloj, de nuevo ocurre la carga, y así sucesivamente.

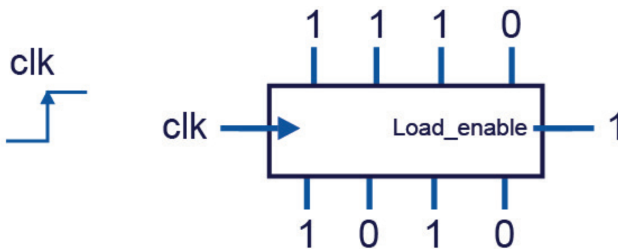


Figura 6.6 Funcionamiento de un registro de carga paralela

©Editorial Digital Tecnológico de Monterrey

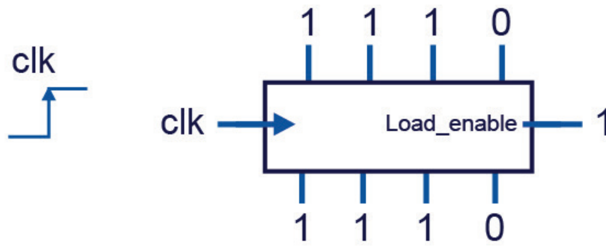


Figura 6.7 Funcionamiento de un registro de carga paralela

6.2.2 Registro con entrada serial sincrónica y salida serial con corrimiento (shift) a la derecha

También se mostrará por pasos la operación de este circuito, señalando del lado izquierdo la transición del reloj.

Figura 6.8 Funcionamiento registro serial

Figura 6.9 Funcionamiento registro serial

©Editorial Digital Tecnológico de Monterrey

Figura 6.10 Funcionamiento registro serial

6.2.3 Contador ascendente

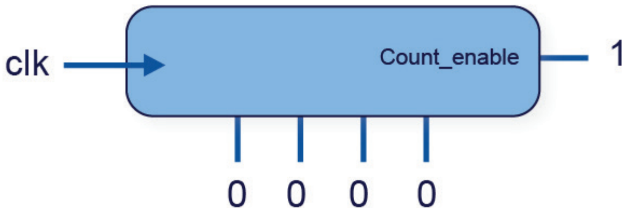


Figura 6.11 Funcionamiento contador ascendente

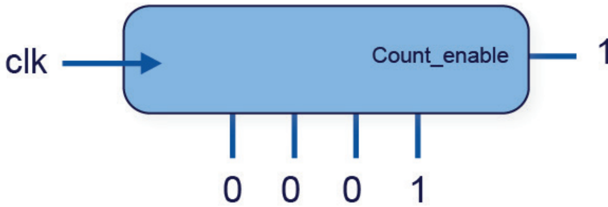


Figura 6.12 Funcionamiento contador ascendente

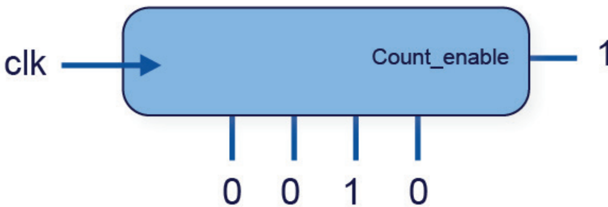


Figura 6.13 Funcionamiento contador ascendente

©Editorial Digital Tecnológico de Monterrey

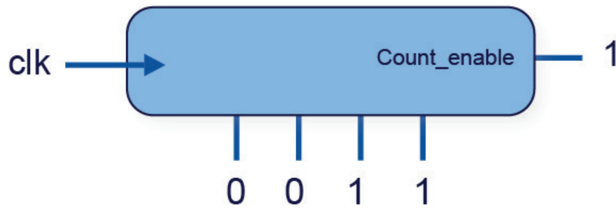


Figura 6.14 Funcionamiento contador ascendente

6.2.4 Contador ascendente con reset asincrónico

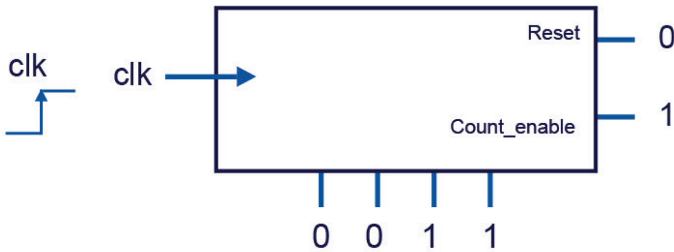


Figura 6.15 Funcionamiento contador ascendente

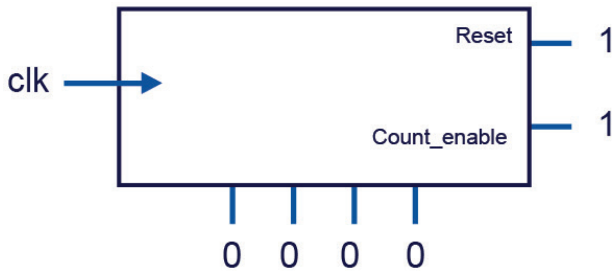


Figura 6.16 Funcionamiento contador ascendente

©Editorial Digital Tecnológico de Monterrey

6.3 Arquitectura interna de una muestra de registros

Como ya se mencionó, un registro se forma con la interconexión de *flip flops*, lógica adicional y entradas de controlk, bnm, uj.

6.3.1 Registro con carga paralela sincrónica

Por ejemplo, este circuito corresponde a un registro de tres *bits* y cuenta con una entrada de *enable* llamada *load*. Como puede observar, cuando load=1 las entradas E2, E1 y E0 ingresan a los tres *flip flops* D y en caso contrario Q se retroalimenta conservando su valor.

©Editorial Digital Tecnológico de Monterrey

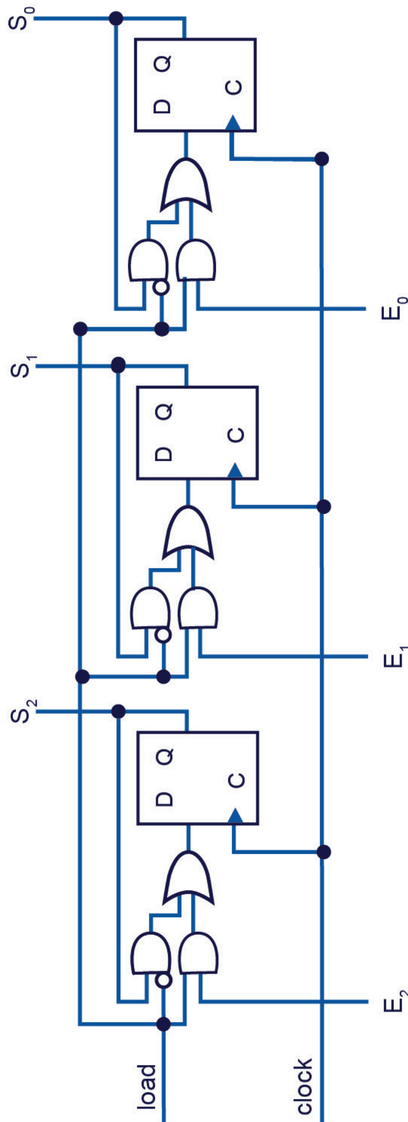


Figura 6.17 Registro con carga paralela sincrónica

La definición de este registro puede realizarse modelando los tres *flip flops*.

Una opción para el código es la siguiente, da clic aquí para conocerlo.

©Editorial Digital Tecnológico de Monterrey

```
Architecture comportamiento of shiftregister is
Signal Q2,Q1,Q0:std_logic;
begin
Process(clk , load)
Begin
 If clk ='1' and clk 'event then
    Q2<=(load and E2) or (not load and Q2);
    Else null;
End if;
End process;
S2<=Q2;

Process(clk , load)
Begin
 If clk ='1' and clk 'event then
     Q1<=(load and E1) or (not load and Q1);
     Else null;
End if;
End process;
S1<=Q1;

Process(clk , load)
Begin
 If clock='1' and clock'event then
    Q0<=(load and E0) or (not load and Q0);
    Else null;
End if;
End process;
S0<=Q0;
End comportamiento;
```

Cabe señalar que S2, S1 y S0 son señales de salida del puerto, por esta razón se definen Q2, Q1 y Q0 como internas, para que sean inout.

Otra opción es visualizar el funcionamiento del circuito como un componente completo y no como una interconexión de *flip flops*.

©Editorial Digital Tecnológico de Monterrey

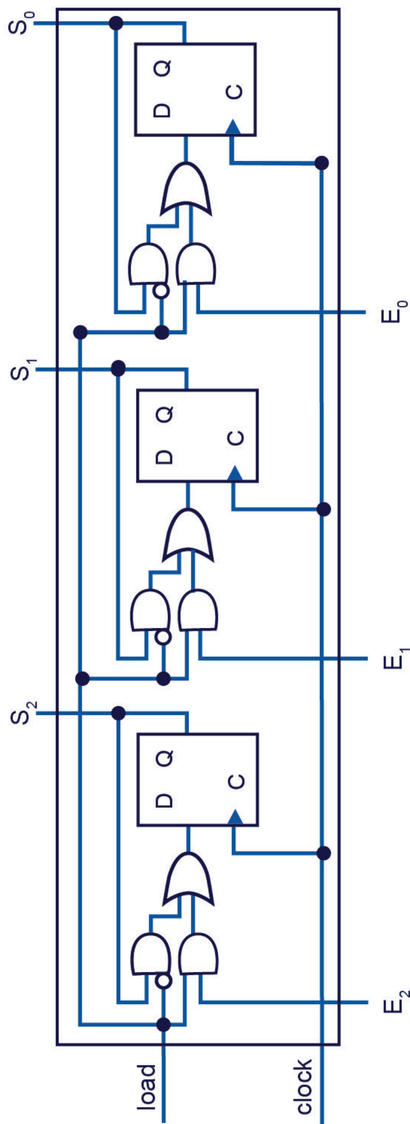


Figura 6.18 Registro con carga paralela sincrónica

Para esto, lo más común es no detallar su arquitectura interna, solo su funcionamiento a partir de las señales de control.

©Editorial Digital Tecnológico de Monterrey

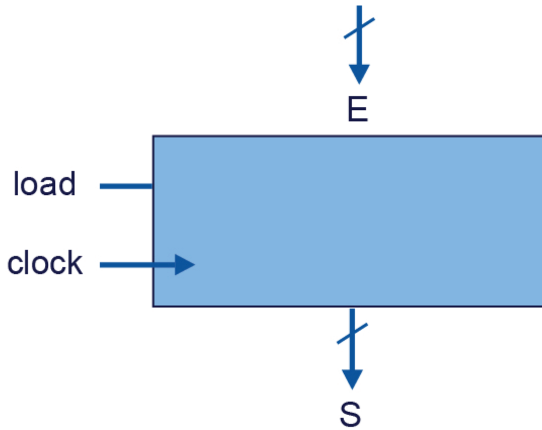


Figura 6.19 Registro de carga paralela

La definición de este registro, visualizándolo como un circuito integrado y haciéndolo a partir de su comportamiento es (modelando la arquitectura):

```
Architecture comportamiento of shiftregister is
Signal Q:std_logic_vector(2 downto 1);
begin
Process(clk , load)
Begin
 If clock='1'a and clock'event and (load='1') then
    Q<=E;
    Else null; --o lo que es lo mismo, Q<=Q
End if;
End process;
S<=Q;
End comportamiento;
```

Nótese que para resumir la descripción se utiliza S, Q y E como vectores y no como *bits* independientes. Esto conduce a que el circuito se visualice como un componente registro y no como un conjunto de *flip flops*.

©Editorial Digital Tecnológico de Monterrey

6.3.2 Registro con carga paralela asincrónica

Otro circuito que se utilizará como ejemplo es el de un registro que permite carga asincrónica en paralelo, la cual se realiza aprovechando las entradas asincrónicas que en este ejemplo llamaremos *Reset* (PR) y *Clear* (CL), que al detectar un cero, dado que están negados, llevan la salida Q a 1, en caso de que el 0 sea detectado en PRE y a 0, si el 0 se detecta en CLR, sin importar la señal de reloj.

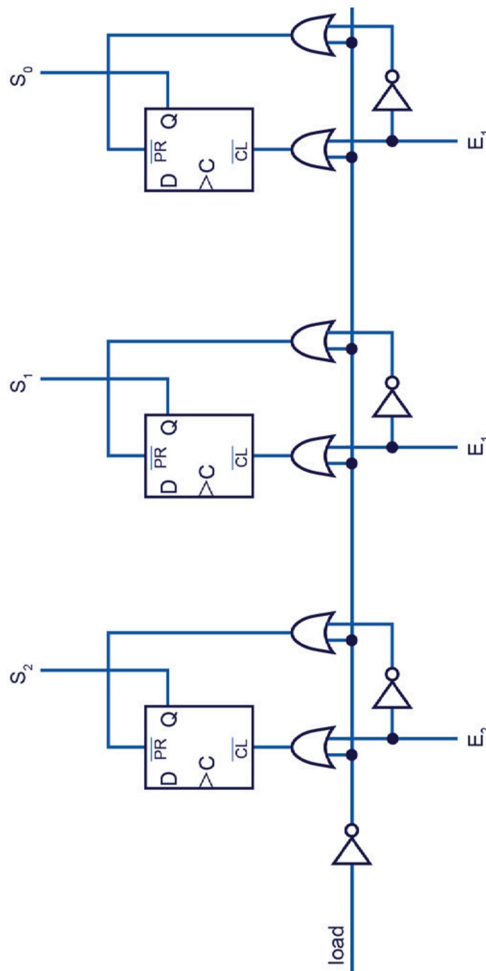


Figura 6.20 Registro de carga paralela asincrónica

©Editorial Digital Tecnológico de Monterrey

Que se puede resumir como:

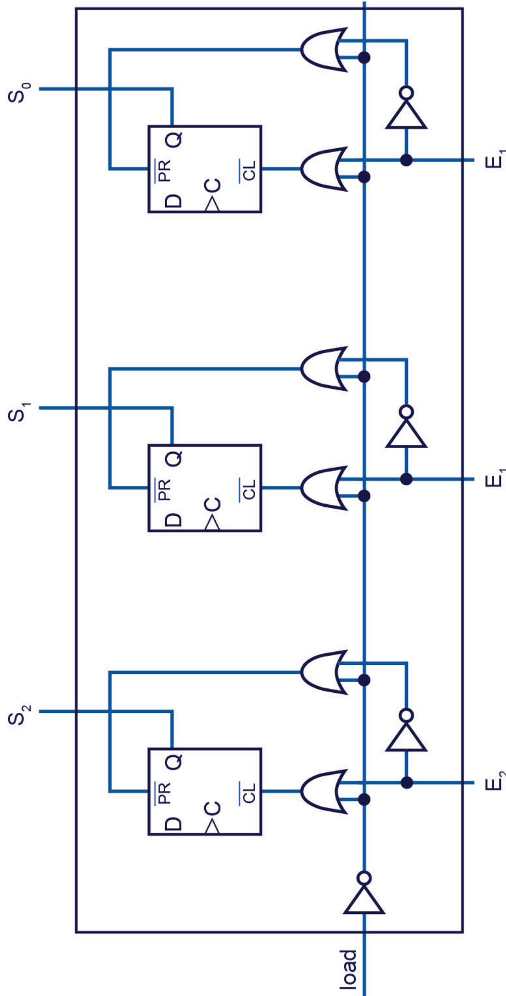


Figura 6.21 Registro de carga paralela asincrónica

©Editorial Digital Tecnológico de Monterrey

O bien, como:

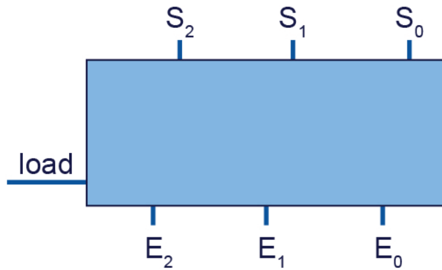


Figura 6.22 Registro de carga paralela asincrónica

O de manera más breve como:

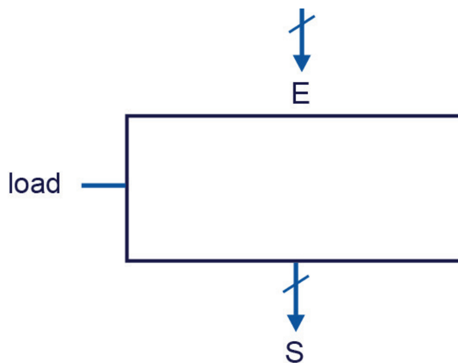


Figura 6.23 Registro de carga paralela asincrónica

Al analizar este circuito, es posible resumir su comportamiento como se presenta a continuación (modelando el elemento en forma independiente):

```
Process(load)
Begin
 If load='1' then S<=E;
   Else null;
 End if;
End process;
```

©Editorial Digital Tecnológico de Monterrey

En esta descripción se utilizan E y S como vectores, y no *bit* a *bit*.

Este circuito, que solo almacena *bits* sin aplicarles ninguna función, podría conformarse de latches y no de *flip flops*, dado que no se utiliza el reloj.

6.3.3 Registro con carga serial sincrónica y corrimiento a la derecha

Otro ejemplo útil es un registro con carga serial que realiza corrimiento hacia la derecha.

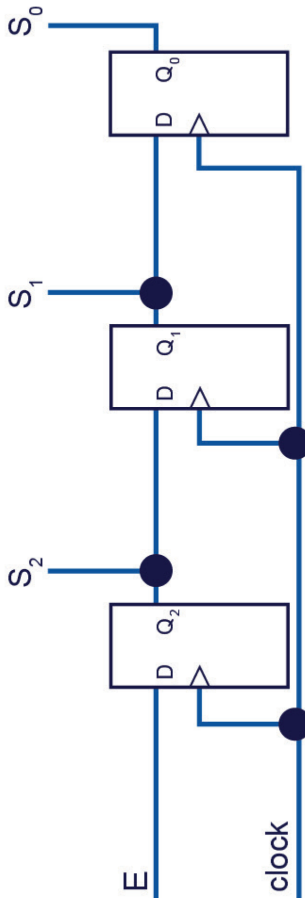


Figura 6.24 Registro con carga serial sincrónica y corrimiento

©Editorial Digital Tecnológico de Monterrey

Hay varias formas de modelar este registro, primero se hará con *flip flops* en procesos independientes (modelando la arquitectura):

```
Architecture flipflops of shiftregister is
Signal Q2, Q1, Q0:std_logic;
begin
Process(clk )
Begin
 If clk ='1' and clk 'event then
    Q2<=E;
    Else null;
End if;
End process;
S2<=Q2;
```

```
Process(clk )
Begin
 If clk ='1' and clk 'event then
    Q1<=Q2;
    Else null;
End if;
End process;
S1<=Q1;

Process(clk )
Begin
 If clock='1' and clock'event then
    Q0<=Q1;
    Else null;
End if;
End process;
S0<=Q0;
End comportamiento;
```

©Editorial Digital Tecnológico de Monterrey

Ahora se modelará el mismo registro, también describiendo la arquitectura interna con *flip flops*, pero utilizando solo un proceso (que es válido).

```
Architecture comportamiento of shiftregister is
Signal Q2,Q1,Q0:std_logic_vector;
begin
Process(clk )
Begin
 If clk ='1' and clk 'event then
    Q2<=E;
    Q1<=Q2;
    Q0<=Q1;
    Else null;
End if;
End process;
S2<=Q2;
S1<=Q1;
S0<=Q0;
End comportamiento;
```

Una descripción equivalente, utilizando solo un proceso es:

```
Architecture comportamiento of shiftregister is
Signal Q:std_logic_vector(2 downto 0);
begin
Process(clk )
Begin
 If clk ='1' and clk 'event then
    Q(2)<=E;
    Q(1)<=Q(2);
    Q(0)<=Q(1);
    Else null;
End if;
End process;
S<=Q;
End comportamiento;
```

©Editorial Digital Tecnológico de Monterrey

Por último, se muestra la descripción de este registro usando vectores:

```
Architecture comportamiento of shiftregister is
Signal Q:std_logic_vector(2 downto 0);
begin
Process(clk )
Begin
 If clk ='1' and clk 'event then
    Q<=E&Q(2 downto 1);
    Else null;
End if;
End process;
S<=Q;
End comportamiento;
```

El operador **&** se utiliza para expresar concatenación de buses y es muy útil para modelar corrimientos. Nótese que E reemplaza al *bit* más significativo y se pierde el menos significativo al realizar el corrimiento.

6.3.4 Contador ascendente

En esta sección de análisis, diseño y modelación de registros comunes se presenta un contador cuya arquitectura interna puede estar diseñada de dos formas: un diseño que se dirija a generar una cuenta (cuyo método de diseño es tema del capítulo 9), o un registro de carga paralela y un sumador, como se muestra a continuación:

©Editorial Digital Tecnológico de Monterrey

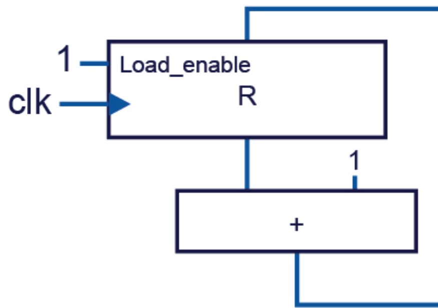


Figura 6.25 Contador ascendente

Este contador puede ser modelado, por su comportamiento, de la siguiente manera:

```
Process(clk )
Begin
If clk ='1' and clk 'event and (Load_enable='1' ) then
   R<=R+1;
   Else null;
End if;
End process;
```

6.3.5 Contador BCD con reset asincrónico

Un contador BCD cuenta de 0 a 9, luego de 0 a 9 y así sucesivamente. Este contador puede tener un diseño interno preparado para que del 9 pase al 0 (que se estudiará en el capítulo 9), o bien, de una manera simple es posible utilizar un contador binario (que cuenta hasta 15) con *reset* sincrónico o asincrónico.

Para un *reset* asincrónico se detecta la salida 10102 y se lleva una señal al *reset*. La salida solo está en 10 durante el tiempo que tarda en propagarse esta señal al *reset*, el retraso de una compuerta lógica. Si el período de la señal de reloj se acerca al retraso de una compuerta lógica, esta no es una solución adecuada, habría que diseñar un

©Editorial Digital Tecnológico de Monterrey

contador que solo cuente de 0 a 9 sin tratar de pasar por el 10, o bien, utilizar un diseño con un *reset* sincrónico. De otro modo esta es una solución adecuada. El diseño y modelación bajo esta opción utiliza dos señales de control: una llamada Inc, que habilita el incremento y otra de *reset* para efectuar el *reset* asincrónico.

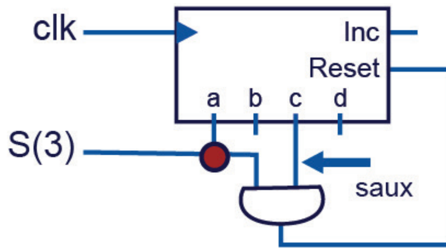


Figura 6.26 Contador BCD con reset asincrónico

La salida es s, en la figura se señala su *bit* más significativo s(3), mientras que saux es la salida que se retroalimenta, la salida interna. Es común asociar el nombre de la salida interna al nombre del registro, como se muestra a continuación:

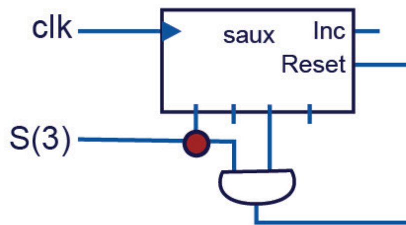


Figura 6.27 Contador BCD con reset asincrónico

El comportamiento de este contador se aprecia en su diagrama de tiempo figura 6.28. El *reset* asincrónico causa que la salida 10 sea instantánea e invisible a la vista.

©Editorial Digital Tecnológico de Monterrey

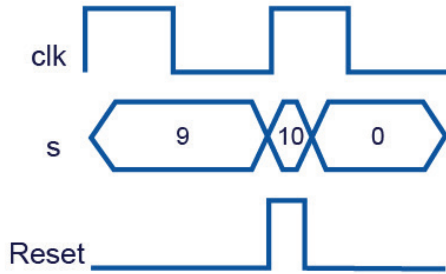


Figura 6.28 Diagrama de tiempo del contador ascendente

Su definición en VHDL es:

```
entity contador is
Port (clk, Inc: in std_logic;
        s: out std_logic_vector (3 downto 0));
end contador;

architecture Behavioral of contador is
signal saux: std_logic_vector(3 downto 0):=”0000”;
signal Reset: std_logic;
```

La inicialización es válida tanto para la simulación como para la síntesis, pero si en el sistema de desarrollo que se esté usando no fuera válida, entonces se requeriría un pulso de *reset* para que se asegure que el contador esté en ceros.

```
begin
process (clk, Inc, Reset)
begin
if Reset=’1’ then saux<=”0000”;
elsif clk=’1’and clk’event and (Inc=’1’) then saux<=
saux+1;
end if;
end process;
s <=saux;
Reset<=saux(3) and saux(1);
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Nótese que en esta modelación el vector s está en el puerto como salida, y si se requiere retroalimentar al *reset*, entonces se necesita un bus auxiliar inout que es saux para que internamente este bus se utilice de entrada al circuito combinacional interno cuya salida se conecta al *reset*. Otro detalle que vale la pena remarcar es que para detectar el diez es suficiente utilizar los pines 3 y 1 y no los cuatro pines, porque al contar 1001, 1010, 1011… la primera vez que se hacen 1 al mismo tiempo, los pines 3 y 1 es en el 10, así que no habría que especificar los ceros de los pines 2 y 0. La señal Inc es un *enable* externo, si no se requiere se omite.

6.3.6 Contador BCD con reset sincrónico

Si el contador binario con el que se diseña el contador BCD tiene *reset* sincrónico, entonces la condición que hay que detectar es la cuenta igual a 9, para que al finalizar el ciclo con cuenta igual a 9 el registro pase a ceros. El diseño se muestra a continuación:

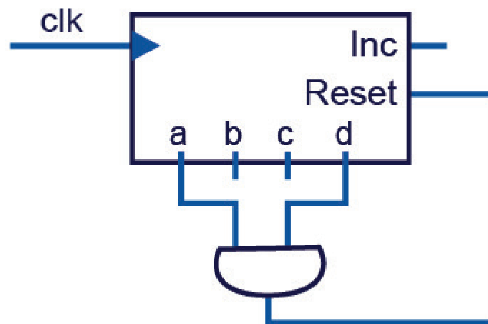


Figura 6.29 Contador BCD con reset asincrónico

©Editorial Digital Tecnológico de Monterrey

6.4 Modelación de registros por sus funciones

En la sección anterior se mostraron definiciones de registros de una función. En esta sección se muestra con mayor detalle la descripción por comportamiento de un registro de múltiples funciones.

Para inferir un registro, el sintetizador busca una definición que sigue cierto patrón y que contiene los siguientes elementos:

- Proceso
- Vector de datos
- Señal de reloj

Y la descripción debe seguir las siguientes reglas:

1. Primero se presentan las condiciones para las operaciones asincrónicas.

2. Luego las sincrónicas a partir de una expresión que indique la transición de la señal de reloj.

La siguiente figura muestra la interfaz para un registro con carga serial y paralela.

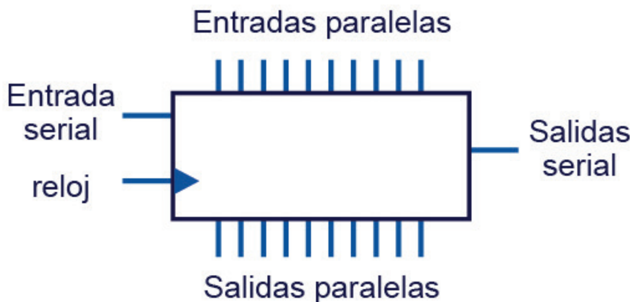


Figura 6.30 Interfaz de un registro

©Editorial Digital Tecnológico de Monterrey

Para modelar un registro se requiere definir un vector de datos que defina la salida paralela del registro. El *bit* menos significativo de este vector representa la salida serial, por ejemplo, para definir un registro de ocho *bits*, se define el vector R:

R: std_logic_vector (7 downto 0)

La figura muestra un registro con entrada serial por el *bit* más significativo. Para proporcionar un dato al *bit* más significativo habría que hacer una asignación del siguiente tipo:

R(7)<= entrada_serial.

Esta asignación debe encontrarse dentro de un proceso que sea sensible a la señal de reloj del registro, y a su vez dentro de un *if* que verifique si hay una transición, ya sea positiva o negativa (dependiendo del registro que se desee definir).

Así que, la definición completa es:

```
process (reloj)
begin
if (reloj='1') and reloj'event then
R(7)<= entrada serial
end process;
```

Por supuesto, esta asignación solo modifica a la entrada serial. Si se desea que ocurra un *shift* a la derecha de la salida del registro al ocurrir la entrada serial, se acude al operador &, con el que se puede describir la salida del registro. La modelación en VHDL de la entrada serial con *shift* a la derecha en la transición positiva de la señal de reloj es:

```
process (reloj)
begin
if (reloj='1') and reloj'event then
R <=  entrada serial&R(7 downto 1);
end process;
```

©Editorial Digital Tecnológico de Monterrey

De nuevo, la salida serial está disponible en R(0) y queda actualizada un delta después de la transición.

Por otra parte, si en vez de usar la entrada serial se desea cargar datos utilizando la entrada paralela en la transición positiva del reloj, la modelación en VHDL sería:

```
process (reloj)
begin
if (reloj='1') and reloj'event then
R<= entrada paralela;
end process;
```

La salida paralela queda en R, y queda disponible y actualizado un delta después de la transición.

Las descripciones anteriores en VHDL mostraron de manera aislada algunas funciones que pueden tener los registros. Integraremos ahora estas funciones en un registro de 8 *bits* y agregaremos otras funciones: un *reset* asincrónico y el incremento en 1 para que actúe como contador. Para que sea posible seleccionar cuál de las funciones se realizará, agregaremos dos señales de control: c1 y c0.

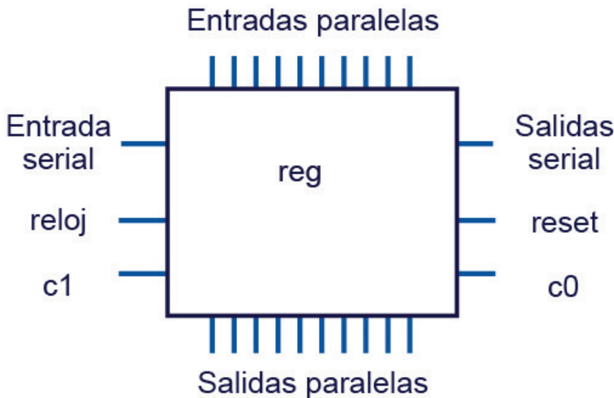


Figura 6.31 Modelo esquemático de un registro multifuncional

©Editorial Digital Tecnológico de Monterrey

```
c1 c0  función
0 0    entrada serial sincrónica por el bit
       más significativo (MSB)
0 1   entrada paralela sincrónica
1 0   incremento en 1 sincrónico
1 1   reset asincrónico
```

Para este registro, el puerto es:

```
Port ( reloj   : in std_logic;
         c1    : in std_logic;
         c0    : in std_logic;
         dato_serial : in std_logic;
         dato_paralelo    : in std_logic_vector(7 downto 0);
         salida_serial    : out std_logic;
         salida_paralela  : out std_logic_vector(7 downto 0));
```

Esta definición de puerto corresponde a la interfaz del registro, pensando, por ejemplo, en los pines de entrada y salida de un circuito integrado de un registro. Notemos que la salida serial y la salida paralela (0) provienen de la misma señal, pero se han separado en dos "pines" de salida del registro. Con el puerto definido de esta manera, requerimos una definición para la salida interna del registro, sobre la cual se puedan hacer operaciones:

signal reg: std_logic_vector (7 downto 0);

Si se recuerda la arquitectura de un registro conformada por flip flops, tendremos en mente que no existe una estructura de registro aislada de las entradas y las salidas, ya que los *flip flops* no son más que circuitos con entradas y salidas, por lo tanto, podríamos modelar las operaciones del registro utilizando solamente la salida paralela. Sin embargo, cuando internamente la salida se retroalimenta, las salidas tendrían que definirse como inout, lo cual no es correcto, así que el vector auxiliar se vuelve necesario.

©Editorial Digital Tecnológico de Monterrey

Siguiendo las reglas de modelación de registros, su definición correcta es la siguiente:

```
architecture Behavioral of ejer5_5 is
signal reg: std_logic_vector(7 downto 0);
begin
salida_serial<=reg(0);
salida_paralela<=reg;
process(reloj, c1, c0)
begin
if (c1='1') and (c0='1') then
   reg<="00000000";
   elsif (reloj='1') and reloj'event then
       if (c1='0') and (c0='0') then
              reg<=dato_serial&reg(7 downto 1);  --se modela la función shift right
              elsif (c1='0') and (c0='1') then
              reg<=dato_paralelo;
              elsif (c1='1') and (c0='0') then
              reg<=reg + 1;
          else null;
        end if;
else null;
end if;
end process;
```

©Editorial Digital Tecnológico de Monterrey

Esta definición es reconocida correctamente por el sintetizador.

Por ejemplo, con la siguiente definición:

```
if (reloj='1') and reloj'event then
      if (c1='0') and (c0='0') then
       reg<=dato_serial&reg(7 downto 1);  --se hace shift right
           elsif (c1='0') and (c0='1') then
            reg<=dato_paralelo;
                elsif (c1='1') and (c0='0') then
           reg<=reg + 1;
                    else null;
        end if;
elsif (c1='1') and (c0='1') then
  reg<="00000000";
end if;
end process;
```

El sintetizador marca el siguiente error:

ERROR:Xst:827 - C:/xilinx_webpack/bin/modul3/ejer5_5.vhd (Line 21).
Signal reg cannot be synthesized, bad synchronous description.

Por último, mencionaremos que de acuerdo a la sintaxis de VHDL, el mismo registro podría ser descrito como se muestra a continuación:

©Editorial Digital Tecnológico de Monterrey

```
architecture Behavioral of registrocon is
signal reg: std_logic_vector(7 downto 0);
begin
reg<=  "00000000" when (c1='1') and (c0='1') else
   dato_serial&reg(7 downto 1) when (reloj='1') and reloj'event and (c1='0') and (c0='0')
else
   dato_paralelo when (c1='0') and (c0='1') and (reloj='1') and reloj'event else
    reg + 1 when (c1='1') and (c0='0') and (reloj='1') and reloj'event
        else reg;
   end Behavioral;
```

Es decir, una descripción sin proceso. Esta descripción, así como la anterior, pueden simularse, pero no sintetizarse. El error que marca el sintetizador es el mismo:

ERROR:Xst:827 - C:/xilinx_webpack/bin/modul3/ejer5_5.vhd (Line 21).

Signal reg cannot be synthesized, bad synchronous description.

©Editorial Digital Tecnológico de Monterrey

6.5 Ejemplos de modelación de registros por sus funciones

6.5.1 Definición de registro multifuncional

Definir en VHDL un registro de 8 *bits* que cuente con las siguientes funciones:

1. Entrada serial sincrónica por el *bit* más significativo (MSB)

2. Entrada serial sincrónica por el *bit* menos significativo (LSB)

3. *Shift right* lógico sincrónico

4. *Shift left* lógico sincrónico

5. Entrada paralela sincrónica

6. Decremento en 1 sincrónico

7. Incremento en 1 sincrónico

8. *Reset* asincrónico

El registro tiene tres entradas de control para realizar las funciones descritas. Con el control = 0 se realiza la primera función, con control = 1 la segunda y así sucesivamente. El registro cuenta con salida paralela que siempre está disponible. Aunque la entrada serial por el MSB corresponde a la entrada paralela de la posición 7, se cuenta con dos pines distintos de entrada. Lo mismo para la entrada serial por el LSB y la entrada paralela de la posición 0 que, aunque representan la misma entrada, tienen pines distintos.

©Editorial Digital Tecnológico de Monterrey

La definición de este registro es la siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--control=0:entrada serial sincrónica por el bit más significativo (MSB)
--control=1:entrada serial sincrónica por el bit menos significativo (LSB)
--control=2:shift right lógico sincrónico
--control=3:shift left lógico sincrónico
--control=4:entrada paralela sincrónica
--control=5:decremento en 1 sincrónico
--control=6:incremento en 1 sincrónico
--control=7:Reset asincrónico

entity ejer5_5 is
Port ( reloj: in std_logic;
c2,c1,c0 : in std_logic;
dato_serial_izq, dato_serial_der: in std_logic;
dato_paralelo_in : in std_logic_vector(7 downto 0);
salida_paralela: out std_logic_vector(7 downto 0));
end ejer5_5;

architecture Behavioral of ejer5_5 is
signal reg: std_logic_vector(7 downto 0);
begin
salida_paralela<=reg;
```

©Editorial Digital Tecnológico de Monterrey

```
process(reloj, c2, c1, c0)
begin
  if (c2='1') and (c1='1') and (c0='1') then --Reset asincrónico
  reg<="00000000";
  elsif (reloj='1') and reloj'event then
    if (c2='0') and (c1='0') and (c0='0') then
    reg<=dato_serial_izq&reg(7 downto 1); --entrada serial por el MSB
    elsif (c2='0')and (c1='0') and (c0='1') then
    reg<=reg(6 downto 0)&dato_serial_der; --entrada serial por el LSB
    elsif (c2='0') and (c1='1') and (c0='0') then
    reg<='0'&reg(7 downto 1);--shift right
    elsif (c2='0') and (c1='1') and (c0='1') then
    reg<=reg(6 downto 0)&'0';--shift left
    elsif (c2='1') and (c1='0') and (c0='0') then
    reg<=dato_paralelo_in;--entrada paralela
    elsif (c2='1') and (c1='0') and (c0='1') then
    reg<=reg - 1;--dec
    elsif (c2='1') and (c1='1') and (c0='0') then
    reg<=reg + 1; --inc
    else null;
    end if;
    else null;
  end if;
end process;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

El reporte de síntesis que se genera para esta definición se presenta a continuación:

```
==============* HDL Compilation *==============
Compiling VHDL  file c:/xilinx/bin/modulo3/registro.vhd in Library work.
Entity <ejer5_5> (Architecture <behavioral>) compiled.

===============* HDL Analysis *===============
Analyzing Entity <ejer5_5> (Architecture <behavioral>).
Entity <ejer5_5> analyzed. Unit <ejer5_5> generated.

===============* HDL Synthesis *===============
Synthesizing Unit <ejer5_5>.
Related source file is c:/xilinx/bin/modulo3/registro.vhd.
Found 8-bit addsub for signal <$n0006>.
Found 8-bit register for signal <reg>.
Summary:
inferred 8 D-type flip-flop(s).
inferred 1 Adder/Subtracter(s).
Unit <ejer5_5> synthesized.
```

©Editorial Digital Tecnológico de Monterrey

```
=========================================================================
HDL Synthesis Report

Macro Statistics
# Registers : 1
8-bit register : 1
# Adders/Subtractors : 1
8-bit addsub : 1

=========================================================================
==========================* Advanced HDL Synthesis *==========================

INFO:Xst:1767 - HDL ADVISOR - Resource sharing has identified that some arithmetic
operations in this design can share the same physical resources for reduced device
utilization. For improved clock frequency you may try to disable resource sharing.

=========================================================================

Minimum period: 6.698ns (Maximum Frequency: 149.298MHz)
Minimum input arrival time before clock: 8.417ns
Maximum output required time after clock: 7.193ns
Maximum combinational path delay: No path found

=========================================================================
Completed process "Synthesize".
```

©Editorial Digital Tecnológico de Monterrey

6.6 Definición de un circuito que contiene más de un registro

El siguiente circuito contiene dos registros. En cada transición de reloj, A carga Pin (parallel in), B carga A y en todo momento Pout (parallel out) muestra el contenido de B.

Si antes de una transición de reloj Pin=0011, A=1001, B=0110, Pout=0110, después de la transición de reloj los cambios que ocurren son: A=0011, B=1001, Pout=1001. Una opción para describir este circuito puede ser modelando por separado los dos registros:

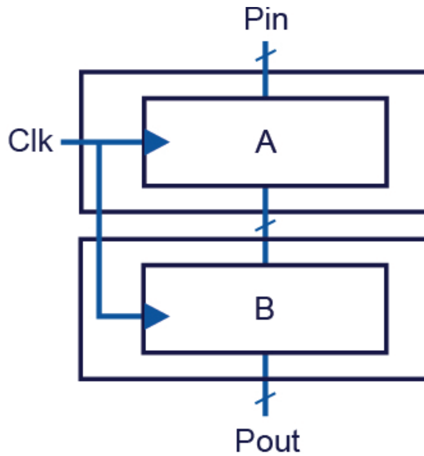


Figura 6.32 Diseño esquemático de un circuito con dos registros

©Editorial Digital Tecnológico de Monterrey

```
architecture Behavioral of pruebas is
signal A, B:STD_LOGIC_VECTOR (3 downto 0);
begin

process(clk )
 begin
  if clk ='1' and clk 'event
   then
     A<=Pin;
   else
     null;
   end if;
 end process;

process(clk )
 begin
  if clk ='1' and clk 'event
   then
     B<=A;
   else
     null;
   end if;
 end process;
 Pout<=B;
end Behavioral;
```

Otra opción es modelar ambos registros en el mismo *process*, el circuito es el mismo, así como su funcionamiento.

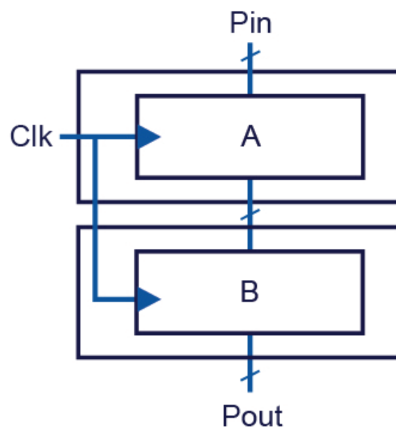


Figura 6.33 Diseño esquemático conjunto de los dos registros

©Editorial Digital Tecnológico de Monterrey

```
entity pruebas is
    Port ( clk  : in STD_LOGIC;
             pin : in  STD_LOGIC_VECTOR (3 downto 0);
           pout : out  STD_LOGIC_VECTOR (3 downto 0));
end pruebas;
architecture Behavioral of pruebas is
signal A, B:STD_LOGIC_VECTOR (3 downto 0);
begin
 process(clk )
 begin
  if clk ='1' and clk 'event
  then
    A<=Pin;
    B<=A;
    else
     null;
   end if;
 end process;
 Pout<=B;
end Behavioral;
```

A continuación, se presenta otro ejercicio en el que se deducirá el circuito que se obtiene a partir de una definición en VHDL. La definición es la siguiente:

```
Signal A,B,C, D: std_logic_vector(3 downto 0);
Begin
 Process(clk , e, f)
  Begin
         If clk ='1' and clk 'event then
         If (e='0') and (f='0') then A<=B;
         elsif (e='0') and (f='1') then A<= C;
                                B<=A;
         elsif (e='1') and (f='0') then A<=D
         else null;
         end if;
         else null;
         end if;
   end process;
```

©Editorial Digital Tecnológico de Monterrey

El circuito que se genera es equivalente a este:

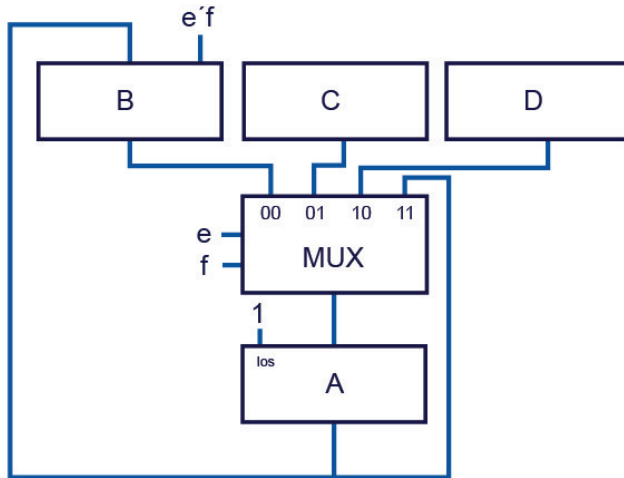


Figura 6.34 Diseño esquemático del ejemplo

En cada uno de estos registros se muestra esquemáticamente la entrada en la parte superior del registro y la salida en la parte inferior.

©Editorial Digital Tecnológico de Monterrey

6.7 Ejemplos de un circuito que contiene más de un registro

6.7.1 Registros conectados en cascada

A continuación, se propone simular las salidas de los registros A y B en las primeras tres transiciones de reloj.

```
Architecture behavioral of circuito is
Signal A,B:std_logic_vector(3 downto 0):="0001";
Begin
If clk ='1' and clk 'event then
    A<=A+1;
    B<=A;
Else null;
End if;
End process;
```

Las salidas son:

```
                                  A          B
Primera transición:             0010       0001
Segunda transición:             0011       0010
Tercera transición:             0100       0011
```

Para la siguiente definición, considerando que en los FPGAs actuales las salidas de los registros inician en ceros, las salidas son:

©Editorial Digital Tecnológico de Monterrey

```
Process(clk )
  If clk ='1' and clk 'event then
     A<= A+1;
     B<= A;
  end if;
end process;
```

En tiempo = 0, A tiene un valor de 0000 (antes de la primera transición).

Después de la primera transición de clk
A = 0001
B = 0000
Después de la segunda transición de clk
A = 0010
B = 0001
Después de la tercera transición de clk
A = 0011
B = 0010

6.7.2 Contador de pulsos

En este ejemplo se diseña un sistema digital que incluye un registro contador de dos *bits* que se incrementa en 1 cuando se oprime un botón <Enter>. Solo se deberá incrementar en una unidad por cada vez que se oprima el botón <Enter>, sin importar cuánto tiempo se quede oprimido. El sistema también cuenta con un *reset* asincrónico para poner a ceros el contador. Suponga que la señal <Enter> ingresa sin rebote.

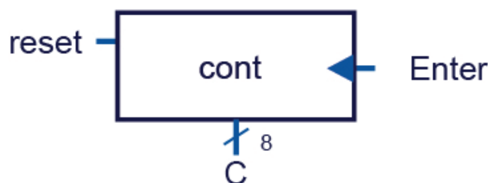


Figura 6.35 Diseño esquemático del ejemplo

©Editorial Digital Tecnológico de Monterrey

```
entity SDE is
    Port ( ENTER, CLK , RESET: in  STD_LOGIC;
           C : out  STD_LOGIC_VECTOR (7 downto 0));
end SDE;
architecture Behavioral of SDE is
signal


begin

end Behavioral;
```

Una solución es la siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity problema2 is
  Port ( ENTER, CLK , RESET: in  STD_LOGIC;
       C : out  STD_LOGIC_VECTOR(1 downto 0));
end problema2;

architecture Behavioral of problema2 is
signal cont: std_logic_vector(7 downto 0):="00000000";

begin
  process(Reset,enter)
         begin
             if Reset ='1' then
                    cont<="00000000";
             elsif enter='1' and enter'event then
                    cont<=cont+1;
             end if;
  end process;
c<=cont;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

6.8 Modelación incorrecta

El mayor problema que enfrentan los futuros ingenieros que están aprendiendo VHDL, es utilizar VHDL como si fuera un lenguaje de programación como C o Java.

VHDL no es un lenguaje de programación, es un lenguaje descriptor de *hardware*, que es muy distinto. Cuenta con estructuras que hacen posible expresar el comportamiento de un circuito, pero no hay que confundir estos artificios con la programación algorítmica (la que traduce algoritmos que se efectúan paso a paso).

Un buen ejercicio para librarse de errores de descripción de circuitos consiste en observar descripciones erróneas y corregirlas. Los circuitos secuenciales en general se basan en registros, así que hay que cuidar que su modelación quede correcta. En un registro solo se puede efectuar una función en una transición de reloj, un incremento, un *shift*, una carga o *reset*. La clave para un buen diseño es observar con qué señales de control se realiza qué función, y si esta es sincrónica o no.

Todos los códigos que se presentan a continuación muestran errores comunes. En ocasiones no ha sido muy claro lo que hay detrás de la mente de los diseñadores, pero se sugiere ponerse en su lugar para hacer la descripción de la manera correcta.

Los siguientes circuitos no se comportan de la manera en que su diseñador pensó que lo harían.

Para realizar correcciones hay que tomar en cuenta que un *flip flop* solo puede ingresar un *bit* a la vez, esto conlleva a que un registro solo puede realizar una función a la vez, o hacer un incremento, un corrimiento o una carga. El ejercicio que se le propone realizar con cada uno de los siguientes problemas consiste en analizar la simulación y el circuito esquemático que se infiere a partir de un diseño en VHDL para corregirlo. Otra alternativa es analizar el circuito que se solicita y modelarlo.

©Editorial Digital Tecnológico de Monterrey

6.8.1 Dos incrementos en la misma transición (1)

Con este código el diseñador pretende que el contador cont se incremente 1 con el pulso (transición positiva) de a y luego incremente 3 si f es 1.

```
entity s is
  Port ( a,f : in  STD_LOGIC; c:out std_logic_vector(3
downto 0) );
end s;
architecture Behavioral of s is

signal cont:std_logic_vector(3 downto 0):="0011";
begin
process(a)
begin
if a='1' and a'event then
  cont<=cont+1;
  if f='1' then
     cont<=cont+3;
end if;
end if;
end process;
c<=cont;  end Behavioral;
```

De esta descripción, el circuito que interpreta el sistema de desarrollo es:

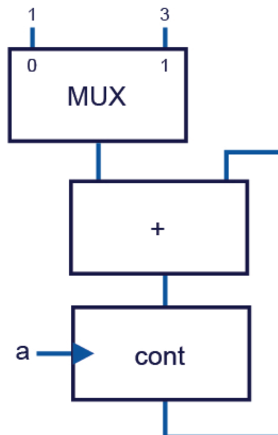


Figura 6.36 Circuito descriptivo

©Editorial Digital Tecnológico de Monterrey

Que no representa lo que espera el diseñador.

El error en esta descripción es pretender que con una transición de la señal de reloj se realicen dos funciones en secuencia. Esto es imposible. La solución puede ser la siguiente (interpretando los requerimientos del diseñador y realizando solo una operación por transición del reloj).

Señal de control	Función
f=1	Incrementar cont en 4
f=0	Incrementar cont en 1

Tabla 6.1

El código corregido es el siguiente:

```
entity s is
    Port ( a,f : in  STD_LOGIC; c:out std_logic_vector(3
downto 0) );
end s;
architecture Behavioral of s is
Solución:
signal cont:std_logic_vector(3 downto 0):="0011";
begin
process(a)
begin
if a='1' and a'event then
   if f='1' then
   cont<=cont+4;
   else
  cont<=cont+1;
end if;
end if;
end process;
c<=cont;
end Behavioral;
```

La señal f puede o no incluirse en la lista del *process* porque la salida no cambia de inmediato si cambia f.

©Editorial Digital Tecnológico de Monterrey

6.8.2 Dos incrementos en la misma transición (2)

Con el siguiente código el diseñador pretende que el contador cont se incremente 1 con el pulso de a, y posteriormente se incremente en 2 si es que f sea 1 si no que se incremente en 4.

```
entity s is
    Port ( a,f : in  STD_LOGIC; c:out std_logic_vec-
tor(3 downto 0));
end s;



architecture Behavioral of s is
signal cont:std_logic_vector(3 downto 0):="0011";
begin
process(a)
begin
if a='1' and a'event then
   cont<=cont+1;
   if f='1' then
             cont<=cont+2;
       else
       cont<=cont+4;
end if;
end if;
end process;
c<=cont;
end Behavioral;
```

En este diseño la simulación e implementación del circuito ignora el incremento en 1 de cont y solo se realiza el incremento en 2 o en 4. Como en el problema anterior, si lo que se desea es que se realice el incremento en 1 y luego en 2 o en 4, se puede diseñar un registro que realice la siguiente operación:

Señal de control	Función
f=1	Incrementar cont en 3
f=0	Incrementar cont en 5

Tabla 6.2

©Editorial Digital Tecnológico de Monterrey

La solución es la siguiente:

```
architecture Behavioral of s is
signal cont:std_logic_vector(3 downto 0):="0011";
begin
process(a)
begin
if a='1' and a'event then
  if f='1' then
          cont<=cont+3;
      else
      cont<=cont+5;
end if;
end if;
end process;
c<=cont;
end Behavioral;
```

6.8.3 Borrar e incrementar en la misma transición

Aquí el diseñador necesita que el contador cont se borre en el preciso instante que tome el valor de 3 y que con el pulso (transición positiva) de a se incremente en 1. Lo anterior no es lo que está descrito en el siguiente código:

```
entity s is
    Port ( a,f : in  STD_LOGIC; c:out std_logic_vector(3
downto 0));
end s;
architecture Behavioral of s is
signal cont:std_logic_vector(3 downto 0):="0000";
begin
process(a)
begin
if a='1' and a'event then
  cont<=cont+1;
  if cont=3 then
    cont<=(others => '0');
     else
     null;
end if;
end if;   end process;
c<=cont;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

El diseño de este registro deja ver la intención de realizar dos operaciones simultáneas: un incremento y cuando cont=3, un *reset* que resulta sincrónico porque ocurre en la transición positiva de a.

Debe tomarse en cuenta que, si el *reset* es sincrónico, entonces durante un período completo de reloj cont tendrá el valor de 3. Si la restricción es que al momento que cont tome el valor de 3 se le aplique la operación de *reset*, entonces el reset tendrá que ser asincrónico.

Bajo esta perspectiva, el diseño es el siguiente:

Señal de control	Función
cont = 3	**reset asincrónico**
cont ≠ 3	**Incrementar cont en 1**

Tabla 6.3

```
architecture Behavioral of s is
signal cont:std_logic_vector(3 downto 0):="0000";
begin
process(a, cont)
begin
if cont=3 then
   cont<=(others => '0');
elsif a='1' and a'event then
 cont<=cont+1;
   else
   null;
end if;
end if;   end process;
c<=cont;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

6.8.4 Definición de registros en diferentes estructuras (1)

En este diseño se está modelando un registro en dos procesos, en uno describiendo el *reset* y en otro el incremento.

```
process(Reset)
        begin
if Reset='1' then
cont1<="0000";
cont2<="0000";
  End if;
end process;
 process(clk )
  begin
 if clk ='1' and clk 'event then
 cont1<=cont1 + 1;
 cont2<=cont2 + 1;
 end if;
end process;
```

No es posible modelar un registro en dos procesos distintos, el error que detecta el sistema de desarrollo es: "multiple source for a signal". Para que cont1 quede bien modelado, su descripción debe integrarse de la siguiente manera:

```
process(Reset, clk )
begin
if Reset='1' then
cont1<="0000";
cont2<="0000";
 elsif clk ='1' and clk 'event then
 cont1<=cont1 + 1;
 cont2<=cont2 + 1;
else null;
end if;
end process;
```

©Editorial Digital Tecnológico de Monterrey

6.8.5 Definición de registros en diferentes estructuras (2)

```
process(clock_25, Reset, r_y, ck) begin
       if rising_edge(clock_25) then
         if (cont_nivel = "0001") then
            ck <= count2(24)
        end if;
       end if;
    end process;
   process(cont_nivel)
   begin
   if (cont_nivel = "0010") then
          ck <= count2(23);
          elsif (cont_nivel = "0011") then
          ck <= count2(22);
         end if;
    end process;
```

Este código tiene el mismo error que el de la descripción anterior, y su corrección se presenta a continuación:

```
process(clock_25, Reset, r_y, ck) begin
       if rising_edge(clock_25) then
         if (cont_nivel = "0001") then
               ck <= count2(24)
       elsif (cont_nivel = "0010") then
              ck <= count2(23);
       elsif (cont_nivel = "0011") then
              ck <= count2(22);
              else null;
         end if;
    else null;
    end if;
    end process;
```

©Editorial Digital Tecnológico de Monterrey

6.8.6 Diferentes operaciones en la misma transición

El siguiente código es erróneo. Lo que se propone es que los registros hagan lo siguiente: cont2 solo se debe incrementar cuando cont1 llega a 800, pero cuando cont1 es 800 y cont2 es 521 cont2 se debe borrar. Cont1 se incrementa cuando no es 800.

```
Signal cont:integer;
begin
   process(clk )
   begin
   if (clk ='1' and clk 'event) then
        cont1<=cont1+1;
        if (cont1=800) then cont1<=0;
            cont2<=cont2+1;
            if (cont2=521) then
            cont2<=0;
            end if;
          end if;
         end if;
        end process;
```

Como se puede ver en este código, se pretende que ante una transición de la señal clk, cont1 se incremente; y si cont1 es 800, también se borre. Además, tal parece que se pudiera tener instantáneamente el valor modificado de cont1 después de incrementarse en una sola transición de reloj, pero esto es imposible.

©Editorial Digital Tecnológico de Monterrey

```
Signal cont:integer;
begin
   process(clk )
   begin
   if (clk ='1' and clk 'event) then
        if (cont1=800) then
 cont1<=0;
          if (cont2=521) then
                 cont2<=0;
           else cont2<=cont2+1;
           end if;
   else cont1<=cont1+1;
end if;
        else null;
       end if;
      end process;
```

Para modificar esta descripción hay que resumir la operación que se debe realizar ante una transición de reloj. La modelación correcta queda de la siguiente manera:

```
entity s is
    Port ( a: in  STD_LOGIC; c:out std_logic_vector(3 downto 0));
end s;
architecture Behavioral of s is
signal cont:std_logic_vector(3 downto 0):="0000";
begin
process(a)
begin
if a='1' and a'event then
  cont<=cont+1;
end if;
  c<=cont;
end process;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

6.8.7 Modelación de la salida incorrecta

En este diseño se pretende que c sea la salida paralela de cont.

```
entity s is
    Port (a: in STD_LOGIC; c:out std_logic_vector(3 downto 0));
end s;
architecture Behavioral of s is
signal cont:std_logic_vector(3 downto 0):="0000";
begin
process(a)
begin
if a='1' and a'event then
   cont<=cont+1;
end if;
   c<=cont;
end process;
end Behavioral;
```

Hay dos soluciones, la primera corresponde a la plantilla de descripción de registros.

```
architecture Behavioral of s is
signal cont:std_logic_vector(3 downto 0):="0000";
begin
process(a)
begin
if a='1' and a'event then
  cont<=cont+1;
end if;
end process;
c<=cont;
end Behavioral;
```

O esta, que funciona bien para el simulador, pero no corresponde a la plantilla de definición de registros, así que el sintetizador no necesariamente reconoce esta definición.

©Editorial Digital Tecnológico de Monterrey

```
architecture Behavioral of s is
signal cont:std_logic_vector(3 downto 0):="0000";
begin
process(a, cont)
begin
if a='1' and a'event then
  cont<=cont+1;
end if;
c<=cont;
end process;
end Behavioral;
```

6.8.8 Contador sin señal de reloj

Un contador no puede modelarse sin reloj, no se estabiliza, por lo que es necesario agregar una entrada de reloj para que su conexión quede correcta.

```
process(enable)
begin
if enable='1' then
  cont<=cont+1;
end if;
```

```
process(enable, clk )
begin
if clk ='1' and clk 'event and (enable='1') then
  cont<=cont+1;
else null;
end if;
```

©Editorial Digital Tecnológico de Monterrey

6.8.9 Modelación incorrecta de señal de control

No es posible modelar un registro ni ningún componente en dos estructuras distintas. Cuando se diseña con ifs lo que se está modelando es un *multiplexer*, quizá especial, cuya salida alimenta a un componente. Eso sí es válido, una modelación bajo una sola estructura.

La siguiente modelación es incorrecta:

```
process(Reset)
begin
 if Reset='1' then cont<="0000"; else null;
 end if;
end process;
process (clk )
begin
  if clk ='1' and clk 'event then
    cont<= cont + 1;
  else null;
  end if;
end process;
```

La modelación correcta es la siguiente:

```
process(Reset,clk )
   begin
         if Reset='1' then
         cont<="0000";
         elsif clk ='1' and clk 'event then
         cont<= cont+1;
         else null;
         end if;
end process;
```

©Editorial Digital Tecnológico de Monterrey

Después de esta intensa práctica con registros, y antes de continuar con el diseño de circuitos digitales basados en registros (capítulo 8), en el siguiente capítulo se discutirán a profundidad señales de botones y de reloj.

a) Corrija el código de cada inciso de tal manera que resulte un registro bien modelado.

```
process(Reset)
being
 if Reset=´1´then cont<=”000”; else null;
 end if;
end process;
process (clk)
begin
if clk =´1´and clk´event then
   cont<= cont + 1;
  else null;
  end if;
end process;
```

b)

```
 process (Enable)
 begin
if Enable='1' then cont<=cont+1; end if;
 end process
```

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 6

1. Corregir la siguiente descripción de registro.

El propósito del siguiente contador es que se incremente en dos cada ciclo y borrarse a cero cuando llega a 14 (debe permanecer en 14 un ciclo completo).

```
 Process(clk )
 begin
if clk ='1' and clk 'event then
cont<=cont+1;
if cont = 14 then cont<="0000" else cont<=cont +1;
end if;
 end process;
```

Solución:

```
 Process(clk )
 begin
if clk ='1' and clk 'event then
 if cont = 14 then
 cont<="0000";
 else
 cont<=cont + "0010";
 end if;
end if;
 end process;
```

©Editorial Digital Tecnológico de Monterrey

2. Muestre en un diagrama esquemático de bloques (a nivel muy general) del circuito que se genera de la siguiente sección en **VHDL**:

```
 process(clk )
 begin
If clk ='1' and clk 'event and a='1' and b='1' then
         Registro <=Registro+1;
end if;
 end process;
```

3. Indique los errores que encuentre en los siguientes segmentos de código en **VHDL** y corríjalos.

```
 Architecture comportamiento of shiftregister is
 Signal Q2,Q1,Q0:std_logic;
 begin
 Process(clk , load)
 Begin
If clk ='1' and clk 'event then
         Q2<=(load and E2) or (not load and Q2);
         Else null;
 End if;
 End process;
 S2<=Q2;

 Process(clk , load)
 Begin
 If clk ='1' and clk 'event then
         Q1<=(load and E1) or (not load and Q1);
         Else null;
 End if;
 End process;
 S1<=Q1;
```

©Editorial Digital Tecnológico de Monterrey

```
Process(clk , load)
Begin
If clock='1' and clock'event then

        Q0<=(load and E0) or (not load and Q0);
        Else null;
End if;
End process;
S0<=Q0;
End comportamiento;
```

4. La descripción del circuito integrado 74194, que es un registro de corrimientos (***shift*** register) bidireccional de cuatro *bits*, es la siguiente:

La entrada **CLRb** pone en ceros la salida, es asincrónica, se activa en 1 y cuando está en 1 no se toma en cuenta ninguna otra entrada. El resto de las funciones se aplican en la transición positiva del reloj. Si la entrada de control **S1=S0=1**, el registro se carga en paralelo. Si **S1=1** y **S0=0** el registro se recorre a la derecha y la salida serial derecha (SDR, por sus siglas en inglés) se refleja en **Q3**. Si **S1=0** y **S0=1**, el registro se recorre a la izquierda y la salida serial izquierda se refleja en **Q0**. Si **S1= S0=0** no ocurre ninguna acción.

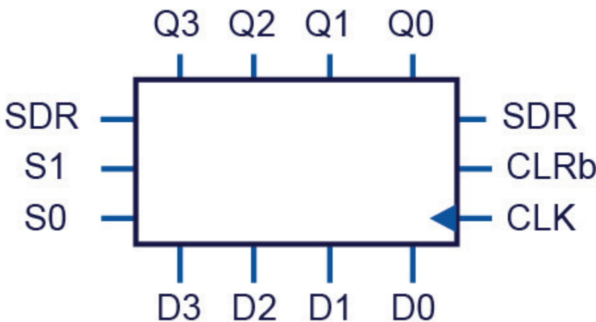


Figura 6.37

©Editorial Digital Tecnológico de Monterrey

Modele en VHDL este registro.

5. Conecte dos de circuitos 74194 para conseguir un registro de 8 *bits* con la misma funcionalidad. Muestre:

a) el diseño esquemático, y

b) la descripción en VHDL.

6. Diseñe en VHDL un registro REG de 8 *bits* que cuenta con dos entradas de *Enable*: E1 y E2. Cuando E1 está en uno se carga sincrónicamente la entrada paralela PI1 en los cuatro *bits* más significativos. Cuando E2 está en 1 se carga la entrada paralela PI0 en los cuatro *bits* menos significativos. La salida paralela de 8 *bits* es PO.

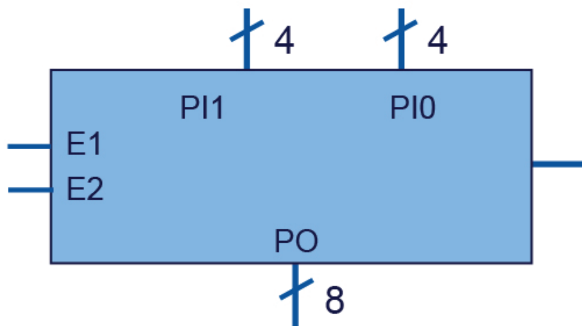


Figura 6.38

©Editorial Digital Tecnológico de Monterrey

El puerto del registro es el siguiente:

```
entity REG is
  Port ( N : in STD_LOGIC_VECTOR (4 downto 0);
             CLK , E1 , E2: in STD_LOGIC;
         PO : out STD_LOGIC_VECTOR (7 downto 0));
end REG;
```

7. Conecte tres circuitos 74194 para lograr un registro de 12 *bits* con la misma funcionalidad que el 74194.

8. Este problema consiste en colocar un número binario de tres *bits* en micro *switches*, oprimir un botón que haga la función de <ENTER> e iluminar el número en tres leds. Luego regresar los *switches* a 0 y hacer que el número siga iluminándose en los leds. Si se configura otro número y se vuelve a oprimir <ENTER> se deberá iluminar el nuevo número. Esto seguirá ocurriendo cada vez que se oprima <ENTER>.

Utilice un botón y un registro para resolver este problema.

Tome en cuenta que el funcionamiento del botón es diferente al de los *switches*.

Simule el circuito.

Al utilizar el simulador puede utilizar el siguiente comando para asociar un comportamiento en el tiempo a una señal de entrada:

force señal valor tiempo, valor tiempo, valor tiempo …
por ejemplo
force boton 0 0, 1 10, 0 30, 1 44, 0 100

©Editorial Digital Tecnológico de Monterrey

Aquí se asocia 0 a un tiempo=0, la señal botón permanece en 0 hasta tiempo 10 en el que cambia a 1 y permanece en 1 hasta 30, en ese momento cambia a 0 y en tiempo 44 vuelve a cambiar a 1 hasta que en tiempo 100 cambia a 0 y así permanece hasta el resto de la simulación.

Considere en la solución conectar a la entrada de un registro con carga paralela los *microswitches*, los Leds conéctelos a la salida y el botón al que se ha llamado <ENTER> a la entrada de reloj.

9. Dibuje el diseño esquemático, basado en *flip flops* D (que no cuentan con funciones de *reset* ni *enable*), de un registro de 2 *bits* que tiene tres entradas de control: *enable, load* y *reset*. El circuito realiza las siguientes funciones:

Si *enable*=1 y *load*=1 entonces se realiza una carga paralela sincrónica.

Si *enable*=1 y *load*=0 entonces se realiza una carga serial sincrónica por el *bit* más significativo con shift a la derecha.

Si *enable* =0 y *reset*=1 pone las salidas en ceros en forma sincrónica.

Usted defina los nombres de las entradas y salidas.

10. Construya el diseño esquemático de un registro de 4 *bits* (el diseño a nivel de flip flops D que no tienen señal de *reset* ni de enable) que cuente con las siguientes señales de control y sus correspondientes funciones:

a) *Load*p: carga sincrónica paralela.

b) *Load*s: carga sincrónica serial por la izquierda.

c) *Reset* sincrónico.

©Editorial Digital Tecnológico de Monterrey

El circuito debe contar con los pines necesarios para la entrada y salida paralela, así como la entrada y salida serial. También debe contar con las entradas de control y la del reloj.

11. Busque el circuito integrado comercial más similar al circuito que resolvió en el problema anterior.

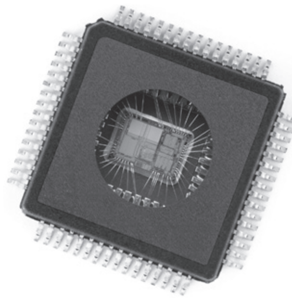
©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 6

Los registros constituyen la base del diseño sincrónico secuencial. Cualquier problema que se quiera resolver, que exija una secuencia de pasos y realizar diversas funciones puede ser construido interconectando registros. Es necesario conocer con todo detalle los diferentes registros que es posible modelar para estar preparados para el diseño de circuitos secuenciales.

Después de esta práctica, antes de continuar con el diseño de circuitos digitales basados en registros (capítulo 8) en el siguiente capítulo, se discutirán a profundidad señales de botones y de reloj.



©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 7. Circuitos monoestables y astables

Circuitos monoestables y astables

Señales de reloj

División de frecuencias en potencias de 2

Obtención de frecuencias específicas

Eliminación de ruido eléctrico (rebotes)

Circuito "one shot" y circuito eliminador de rebotes

One shot

Eliminación de rebotes

Como preparación previa al diseño de circuitos secuenciales basados en registros, este capítulo se dedica a estudiar elementos relacionados con la operación y diseño de este tipo de circuitos.

Entre los elementos relacionados con la operación de circuitos secuenciales se encuentran los componentes astables y los monoestables.

Un astable es un circuito multivibrador que no tiene ningún estado estable, oscila entre un nivel y otro, lo que significa que posee dos estados "quasi-estables" entre los que conmuta, permaneciendo en cada uno de ellos un tiempo determinado. El tiempo que transcurre en un estado y el siguiente es denominado período. Por el patrón que sigue la repetición de su conmutación pertenece al tipo de señales periódicas (que repiten un patrón de oscilación en el tiempo).

Existen circuitos integrados que generan señales astables, como el timer 555. La frecuencia de conmutación depende, en general, de la carga y descarga de condensadores. Entre las múltiples aplicaciones de estos circuitos se encuentran la generación de ondas periódicas (generador de reloj) y de trenes de impulsos. En este capítulo se trabaja con señales astables que sean útiles como señales de reloj.

©Editorial Digital Tecnológico de Monterrey

El monoestable es un circuito multivibrador que realiza una función consistente en que, al recibir una excitación exterior, cambia de estado y se mantiene en este durante un periodo que viene determinado por una constante de tiempo. Transcurrido dicho período, la salida del monoestable vuelve a su estado original. Por tanto, tiene un estado estable (de aquí su nombre) y un estado casi estable. El tipo de circuito monoestable que se analiza en este capítulo es el *one shot*, en especial el que genera un estado que perdura un ciclo de una señal de reloj.

El ruido eléctrico es una señal astable de alta frecuencia producida al hacer contacto un circuito electromecánico, entre otras causas. En el capítulo se analiza la aplicación de circuitos monoestables para la eliminación de ruido eléctrico.

©Editorial Digital Tecnológico de Monterrey

7.1 Señales de reloj

Una señal de reloj es una señal astable que fluctúa y tiene dos estados 0 y 1 lógico (con los niveles de voltaje y corriente que soporte el circuito al que se va a conectar).

Si se recuerda del capítulo 4, un *flip flop* está construido a partir de dos *latches* en conexión maestro-esclavo, uno se habilita en 0 y el otro en 1. El segundo se habilita cuando el primero se deshabilita, respondiendo así cuando ocurre una transición en la señal conectada a los habilitadores. La señal que se conecte a estos habilitadores debe tener transiciones, es decir, cambios de estado. Sabemos que en conjunto este circuito maestro-esclavo es un *flip flop*, y la señal que se debe conectar a los habilitadores se conoce como señal de reloj.

Por otra parte, la interconexión de *flip flops* consigue registros. La conexión de una señal de reloj a diversos componentes que tienen entrada de reloj (*flip flops* y registros) consigue su sincronización, es decir, que reciban y produzcan datos a un mismo tiempo.

No es posible construir un registro que haga transformaciones sobre los datos que almacena sin conectarle una señal de reloj, ya sea periódica o que al menos tenga cambios de estado, que tenga transiciones. En general los circuitos configurables no cuentan con circuitos internos que generen señales de reloj, así que hay que conectar circuitos externos. Si el reloj externo no cuenta con la frecuencia necesaria para resolver algún problema, es posible manipular su frecuencia y obtener una menor. A continuación, se muestra un conjunto de ejercicios destinados a este fin. Este tema se presenta para un aprendizaje inductivo, por lo que a partir de ejemplos es posible generalizar un método a seguir.

Una señal de reloj periódica tiene una frecuencia. La frecuencia de una señal periódica se mide en hertz.

$$1 \text{ Hertz} = \frac{\text{un ciclo}}{\text{segundo}}$$

©Editorial Digital Tecnológico de Monterrey

Por otra parte, el período de una señal periódica es:

```
Periodo = 1 / frecuencia

que tiene como unidades = segundos / ciclo
```

7.1.1 División de frecuencias en potencias de 2

Se cuenta con un reloj (clk) que tiene una frecuencia de 50 MHz y se desea obtener, a partir de este, otras señales periódicas de menor frecuencia.

A continuación, se presenta la descripción de circuitos en VHDL para obtener:

a. clk/2 = 25 MHz

b. clk/4 = 12.5 MHz

c. clk/8 = 6.25 MHz

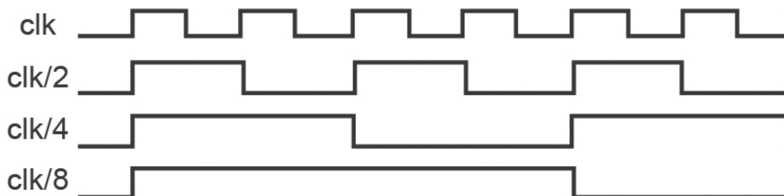


Figura 7.1

©Editorial Digital Tecnológico de Monterrey

Si se utiliza un contador de tres *bits* que se incremente con la frecuencia de este reloj de 50 MHz, se logra la secuencia:

```
000        100        000
001        101        001
010        110        Etc.
011        111
```

Si se grafica el *bit* 0 de salida del contador, se observa que corresponde a una señal periódica que tiene la mitad de la frecuencia del reloj que se utiliza para incrementar el contador; el *bit* 1 tiene la cuarta parte de la frecuencia reloj (la mitad de la anterior) y el *bit* 2 la octava parte (la mitad de la anterior).

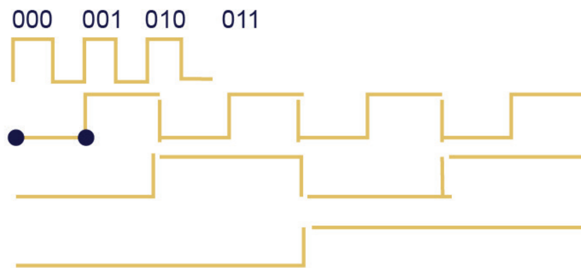


Figura 7.2

©Editorial Digital Tecnológico de Monterrey

El código para obtener estas tres señales es el siguiente:

```
signal contador:std_logic_vector(2 downto 0);
signal clk1,clk2,clk3:std_logic;
begin
process(clk)
begin
  if clk='1' and clk'event then
     contador<=contador+1;
  end if;
end process;
clk1<=contador(0);
clk2<=contador(1);
clk3<=contador(2);
end Behavioral;
```

De esta manera, con un contador es posible obtener una división de frecuencia que sea una potencia de 2. En general, en el *bit* de la posición i-1 de la salida del contador tendremos una frecuencia que sea la fracción 1/2i de la frecuencia original. Al simular el código anterior se obtendría el siguiente diagrama de tiempo:

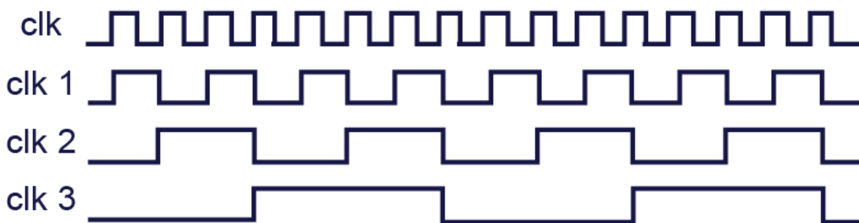


Figura 7.3

©Editorial Digital Tecnológico de Monterrey

7.1.2 Obtención de frecuencias específicas

A partir de clk de 50 MHz se requiere una frecuencia que tenga un período de 1 seg. (es decir, frecuencia de 1 Hz).

En este caso, para obtener un Hertz habría que dividir entre 50,000,000 a la frecuencia original, lo cual no es una potencia de dos. Entonces se puede utilizar otro enfoque para obtener esta frecuencia. Como en 50,000,000 de ciclos el contador contaría hasta 50,000,000 en 1 seg., es posible esperar a la mitad de la cuenta para negar un nuevo reloj y así obtener una señal cuadrada de 1 Hz.

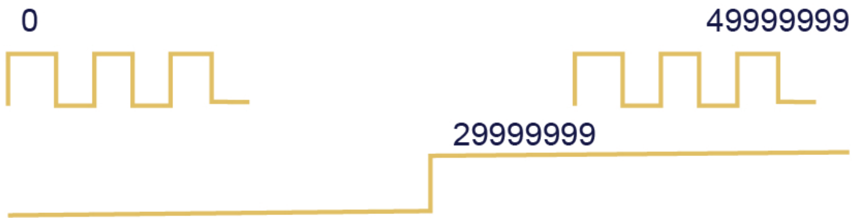


Figura 7.4

```
 signal contador: integer;
 signal clk1hz: std_logic;
 begin
 process(clk)
 begin
if clk='1' and clk'event then
          if contador=24999999 then
               contador<=(others=>'0');
               clk1hz<= not clk1hz;
          else
               contador<=contador+1;
            end if;
          else null;
end if;
 end process;
```

©Editorial Digital Tecnológico de Monterrey

Por otra parte, si también a partir de clk de 50 MHz se requiere una frecuencia que tenga un período de 2 seg. (es decir, frecuencia de 0.5 Hz). La división en este caso es 50,000,000/0.5 = 100,000,000. Para obtener una señal cuadrada, contando hasta 50,000,000 se tendrá la mitad del ciclo.

Figura 7.5

```
signal contador:integer;
signal clkmediohz:std_logic;
begin
process(clk)
begin
if clk='1' and clk'event then
        if contador=49999999 then        contador<=(o-
        thers=>'0');
clkmediohz<= not clkmediohz;
                        else        contador<=contador+1;
          end if;
        else null;
end if;
end process;
```

Si se desea obtener la frecuencia más aproximada a 200 KHz utilizando una señal periódica clk de 50 MHz. La división 50,000,000/200,000 = 250 proporciona las veces que ocurre la frecuencia buscada en la frecuencia disponible. Como es una división exacta es posible obtener los 200KHz con un contador que cuente hasta 125 para obtener la mitad del ciclo.

©Editorial Digital Tecnológico de Monterrey

```
signal contador:std_logic_vector(7 downto 0);
signal clk200KHz:std_logic;
begin
process(clk)
begin
  if clk='1' and clk'event then
     if contador=124    then    contador<=(others=>'0');
clk200KHz<= not clk200KHz;
                    else   contador<=contador+1;
       end if;
     else null;
  end if;
end process;
```

Si el compilador que utiliza no acepta esta descripción puede cambiar el contador a tipo integer o bien expresar 124 en binario.

Responde la siguiente actividad.

1. Diseñe en VHDL un circuito que obtenga una frecuencia de 1MHz utilizando una señal **clk** de 50 MHz.

2. En el siguiente código, si **clk** tiene una frecuencia de 50 MHz, ¿cuál es la frecuencia de **clkX**?

```
signal contador:std_logic_vector(5 downto 0);
signal clkX:std_logic;
begin
process(clk)
begin
if clk='1' and clk'event then
        contador<=contador+1;
end if;
end process;
clkX<=contador(5);
```

©Editorial Digital Tecnológico de Monterrey

3. Otra forma de lograr una frecuencia específica es con un registro de rotación, tal y como se muestra en este ejercicio.

Si clk tiene una frecuencia de 50 MHz,

a) ¿Cuál es la frecuencia de clkX ?

b) ¿Cuál es la forma de la señal de clkX ?

```
signal reg:std_logic_vector(4 downto 0):=”000001”;

signal clkX:std_logic;

begin
process(clk)
begin
if clk=’1’ and clk’event then
        reg<=reg(0)&reg(5 downto 1);
end if;
end process;
clkX<=reg(4);
```

En este ejercicio podrá observar que el tamaño de registro de rotación determina el divisor de la frecuencia. Cualquiera de los *bits* del registro de rotación puede ser utilizado como señal periódica.

©Editorial Digital Tecnológico de Monterrey

7.2 Eliminación de ruido eléctrico (rebotes)

Un gran número de sistemas digitales interactúan con botones. Las características electromecánicas de los botones provocan que, al hacer contacto, generen ruido eléctrico conocido como rebote. Esto puede representar un problema para algunas aplicaciones. Además, si se desea que al accionar un botón se habilite una función de un registro para realizar alguna operación, convendría que la duración del pulso generado por el botón fuera el de un ciclo de reloj. Estos temas se tratan en la siguiente sección.

7.2.1 Circuito one shot y circuito eliminador de rebotes

La señal que proviene de botones genera un 1 lógico mientras el botón se encuentre oprimido, y generalmente, tanto el cambio de 0 a 1 como de 1 a 0, tiene transitorios que son cambios de alta frecuencia de 1 a 0 y a 1 por un corto tiempo hasta que finalmente la señal se estabiliza. El transitorio recibe el nombre de rebote. Generalmente el rebote tiene una frecuencia menor a 50 MHz. La señal de reloj que se utilice para conectarse al *one shot* debe tener una frecuencia menor que el ruido, porque de otro modo el rebote deja de ser ruido y se convierte en una señal periódica.

A continuación, se muestran dos posibles tratamientos que se pueden aplicar a la señal Din producida por un botón.

One shot

Hay circuitos integrados que generan señales monoestables. Generalmente son circuitos capacitivos. El tipo de señal monoestable que analizaremos es la que pueda ser utilizada para dos propósitos: a) eliminar rebotes y b) generar un pulso por un tiempo específico.

Dentro de los circuitos monoestables están los que generan señales *one shot*. El tipo de circuito one-shot que analizaremos no es el capacitivo encapsulado en un circuito integrado, sino el que se logra con la interconexión de *flip flops*.

©Editorial Digital Tecnológico de Monterrey

En particular, el *one shot* a analizar genera un pulso que permanece en 1 durante un ciclo completo de reloj, sin importar cuántos ciclos (siempre y cuando sean más que tres) permanezca oprimido el botón.

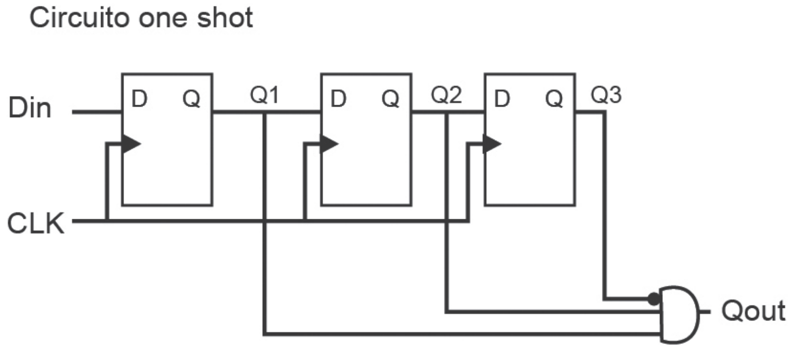


Figura 7.6

Su comportamiento se ilustra a continuación:

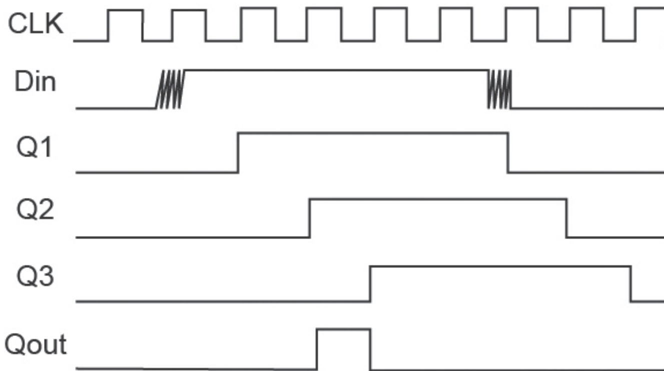


Figura 7.7

©Editorial Digital Tecnológico de Monterrey

La modelación de este circuito en VHDL es la siguiente:

```
entity one_shot is
    Port ( Din, clk : in  STD_LOGIC;
           Qout : out  STD_LOGIC);
end one_shot;

architecture Behavioral of one_shot is
signal Q1, Q2, Q3: std_logic;
begin
process(clk)
begin
 if clk='1' and clk'event then
      Q1<=Din;
      Q2<=Q1;
      Q3<=Q2;
 end if;
end process;
Qout<=Q1 and Q2 and not Q3;
end Behavioral;
```

Eliminación de rebote

Otra opción para esta señal es conservar su duración aproximada y solo “limpiar” la señal, es decir, eliminarle el rebote. Para esto, el siguiente circuito es adecuado.

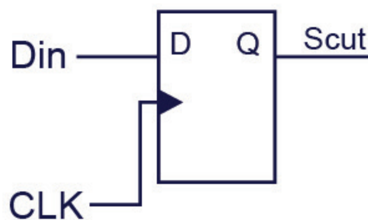


Figura 7.8

©Editorial Digital Tecnológico de Monterrey

La condición para este circuito es que la entrada dure al menos un ciclo de reloj (sin contar los rebotes). La salida resulta limpia, como se muestra a continuación.

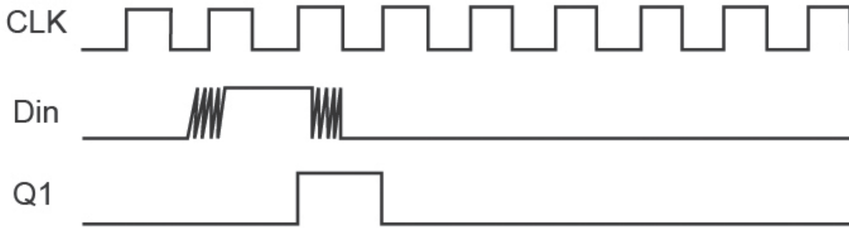


Figura 7.9

Si por algún requerimiento del circuito se desea alargar la duración de la salida, es posible agregar *flip flops* en cascada. Tanto el número de *flip flops* como de períodos que se deseen prolongar, por ejemplo, para alargar por dos períodos se agregan dos *flip flops* y el circuito queda de la siguiente manera:

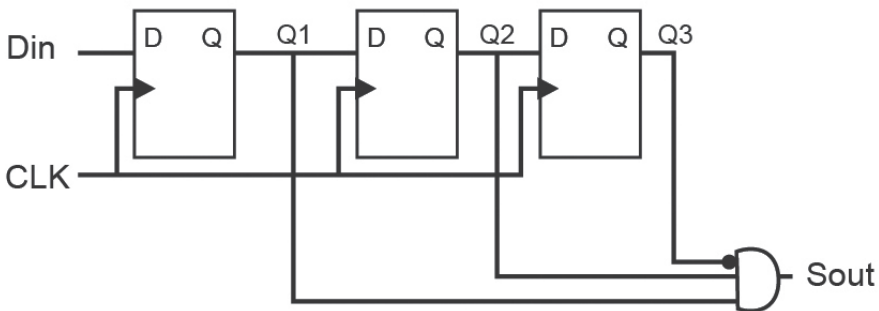


Figura 7.10

©Editorial Digital Tecnológico de Monterrey

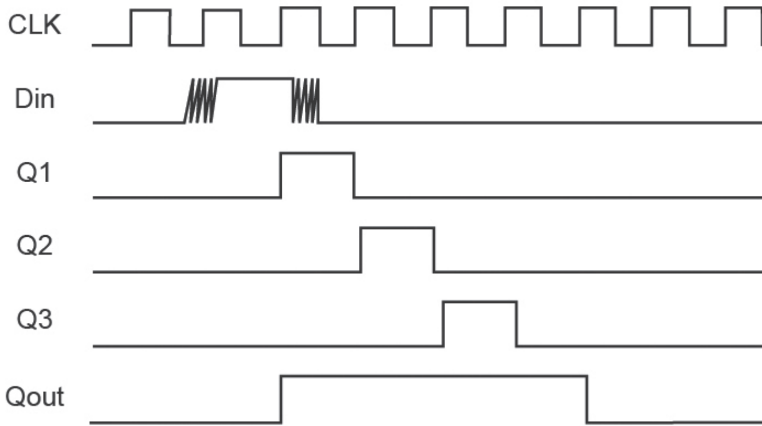


Figura 7.11

Para observar la aplicación de estas alternativas obsérvese el siguiente problema. Si se quisiera contar el número de veces que se ha oprimido un botón, es posible utilizar un registro contador con una entrada de *enable* que incremente en uno en cada transición positiva del reloj. Las posibles conexiones que podrían realizarse son:

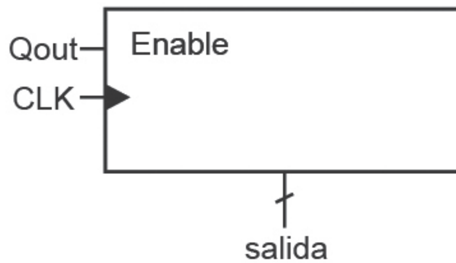


Figura 7.12

©Editorial Digital Tecnológico de Monterrey

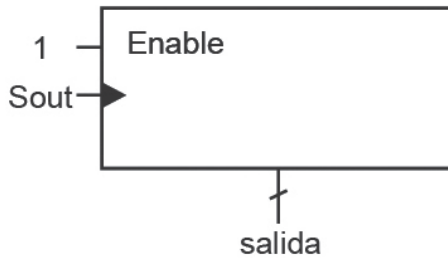


Figura 7.13

En esta última opción podría utilizarse Qout en vez de Sout y tendría el mismo efecto.

Al terminar de estudiar este capítulo el lector tiene los elementos necesarios para diseñar un sistema secuencial, tema del siguiente capítulo.

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 7

1. Se tiene el siguiente código en VHDL.

```
signal contador:std_logic_vector(4 downto 0);
signal clk1:std_logic;
begin
process(clk)
begin
if clk='1' and clk'event then
        contador<=contador+1;
end if;
end process;
clk1<=contador(4);
```

Si la frecuencia de clk es de 64MHz, ¿cuál es la frecuencia de clk1?

a) 1 MHz

b) 2 MHz

c) 4 MHz

d) 8 MHz

e) 16 MHz

e) 32 MHz

©Editorial Digital Tecnológico de Monterrey

2. Se tiene el siguiente código en VHDL.

```
signal contador:std_logic_vector(23 downto 0);
signal clk1hz:std_logic;
begin
process(clk)
begin
if clk='1' and clk'event then
        if contador=31999999 then contador<=(others=>'0');
clk1<= not clk1;
            else                    contador<=conta-
            dor+1;
          end if;
        else null;
end if;
end process;
```

Si la frecuencia de clk es de 64MHz, ¿cuál es la frecuencia de clk1?

a) 1 MHz

b) 2 MHz

c) 4 MHz

d) 8 MHz

e) 1 Hz

f) 2 Hz

g)8 Hz

h)16 Hz

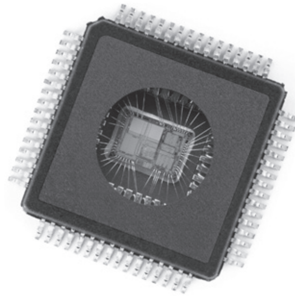
i) 32 Hz

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 7

El manejo de señales de reloj permite preparar la sincronización de circuitos secuenciales, cuyo diseño inicia en siguiente capítulo. Se analizaron circuito para disminuir su frecuencia, que en aplicaciones tales como el diseño de un cronómetro, son de utilidad.

Por otra parte, la conexión de periféricos, como interruptores y botones a circuitos secuenciales requiere que sus señales se condicionen. En este capítulo se estudiaron circuitos para generar señales por tiempo específico y para filtrar el ruido de entrada.

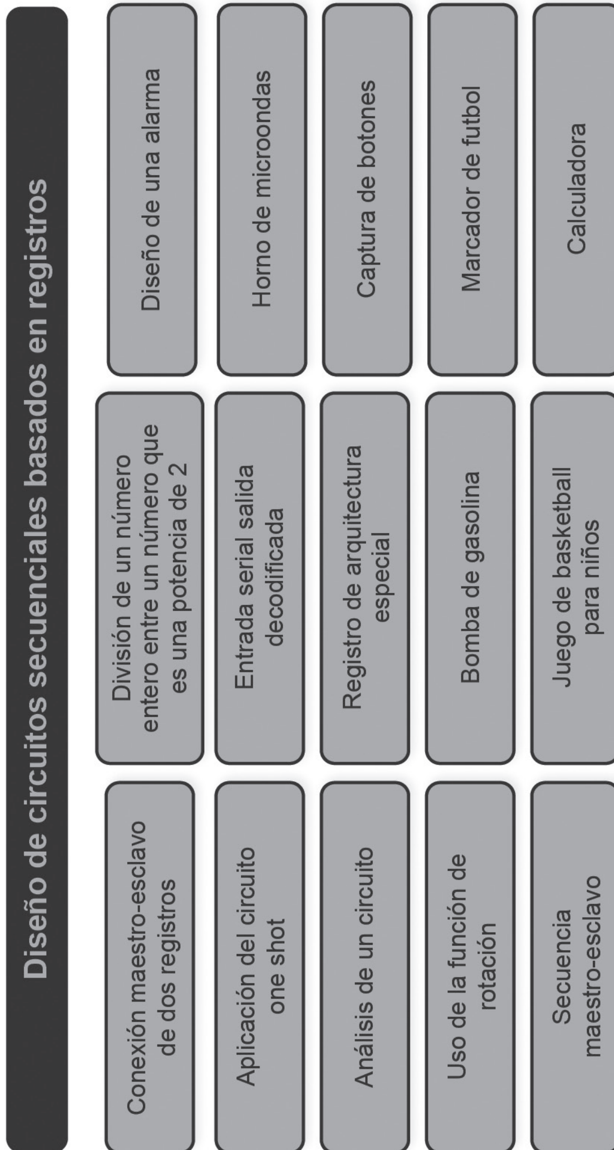


©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 8. Diseño de circuitos secuenciales basados en registros



©Editorial Digital Tecnológico de Monterrey

Este es un capítulo de práctica, ya que se presentarán problemas en cuya solución intervienen circuitos basados en registros, *latches*, *flip flops* y circuitos combinacionales. Se espera que el lector desarrolle la habilidad de diseñar circuitos secuenciales de este tipo al analizar y resolver ejemplos.

©Editorial Digital Tecnológico de Monterrey

8.1 Conexión maestro-esclavo de dos registros

Segundero. Se iniciará con un problema en el que la solución en VHDL se presenta de diferentes maneras para que el lector elija un estilo. El problema es el siguiente:

Se le solicita diseñar en VHDL un fragmento de un cronómetro consistente en un segundero cuyo ciclo sea de 00 a 59, pasando por 00, 01, 02… 09, 10, 11, 12… 19, 20, 21… 29… 57, 58, 59, 00.

Para la solución de este problema conviene utilizar dos contadores BCD, como el que ya se diseñó en el capítulo 6, que cuenten con una entrada Inc para habilitar el incremento y otra de *reset* para llevar el contador a ceros, puede funcionar en forma asincrónica o sincrónica. El primer diseño que se muestra cuenta con *reset* asincrónico.

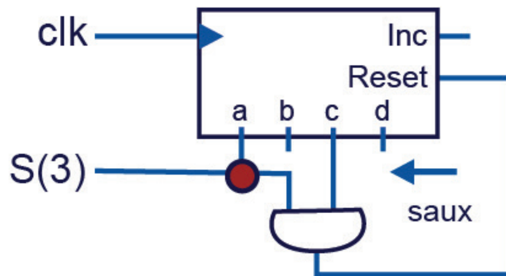


Figura 8.1

Que también se representó como:

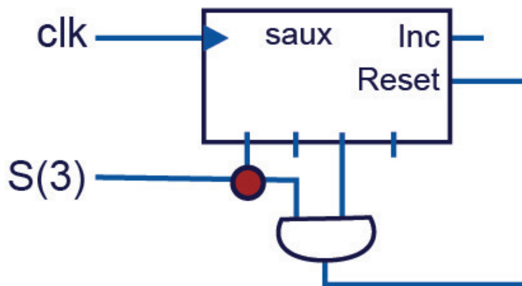


Figura 8.2 Contador con reset asincrónico

©Editorial Digital Tecnológico de Monterrey

Cuya modelación en VHDL es:

```
entity contador is
    Port ( clk, inc, reset : in std_logic;
                s: out std_logic_vector(3 downto 0));
end contador;

architecture Behavioral of contador is
signal saux:std_logic_vector(3 downto 0):=”0000”; --la
inicialización es válida tanto
--para la simulación como para la síntesis, pero si en
el sistema de desarrollo que esté
--usando no fuera válida entonces se requeriría un
pulso de reset para que se asegure
--que el contador esté en ceros
begin
process(clk, inc, reset)
begin
if reset='1' then s<="0000";
   elsif clk=’1’and clk'event and (inc='1') then
saux<= saux+1;
end if;
end process;
s<=saux;
end Behavioral;
```

La solución utiliza un registro para las unidades y otro para las decenas. Cuando las unidades sean 9, el habilitador del incremento Inc sube a 1 y se incrementan las decenas en la siguiente transición de reloj. A continuación, se presenta el diseño y su funcionamiento.

©Editorial Digital Tecnológico de Monterrey

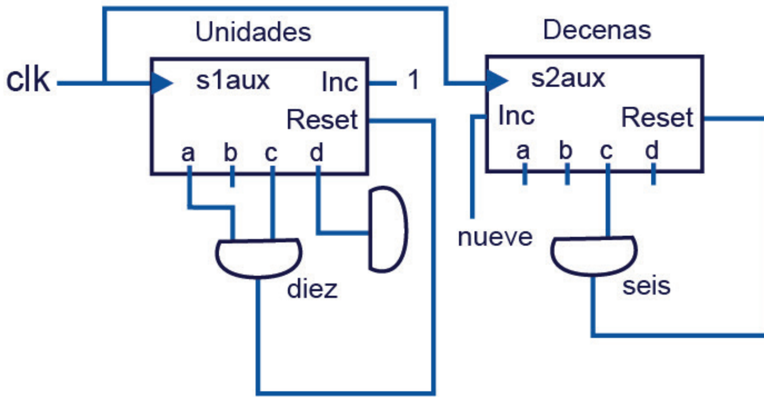


Figura 8.3 Segundero BCD

El análisis del comportamiento de las señales se muestra a continuación:

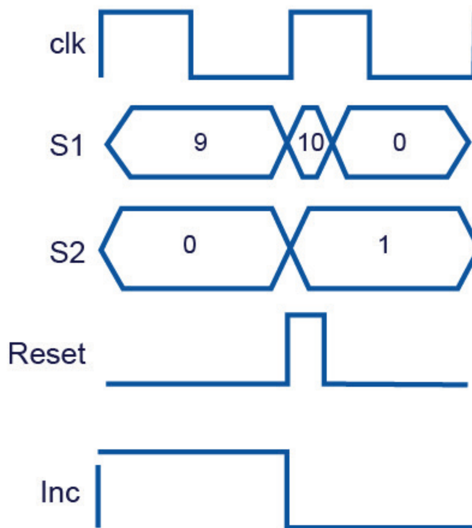


Figura 8.4 Diagrama de tiempo del segundero BCD

©Editorial Digital Tecnológico de Monterrey

También es posible hacer una definición estructural de este circuito:

```
entity segundero is
    Port(s1: out std_logic_vector(3 downto 0);
         s2: out std_logic_vector(3 downto 0);
         clk: in std_logic);
end segundero;

architecture Behavioral of segundero is
signal s1aux, s2aux:std_logic_vector(3 downto 0): = ”0000”;
component contador
Port(clk, inc, reset: in std_logic;
      s: out std_logic_vector(3 downto 0));
end component;
signal diez, nueve, seis: std_logic;
begin
e1:contador port map (clk,‘1’, diez, s1aux);
     --estas salidas auxiliares permiten utilizar
--estas salidas como entradas de
--los circuitos cominacionales
     --requeridos para detectar el
--seis, el diez y el nueve
e2: contador port map (clk, nueve, seis, s2aux);
nueve <= s1aux(3 and s1aux(0)
diez <= s1aux(3) and s1aux(1);
seis <= s2aux(2) and s2aux(1);
s1 <= s1aux;
s2 <= s2aux;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Otra opción no estructural de descripción requiere volver a definir el contador BCD y hacerlo para dos instancias, las unidades y las decenas. La definición puede ser la siguiente:

```
entity seg_procesos is
    Port(s1: out std_logic_vector(3 downto 0);
         s2: out std_logic_vector(3 downto 0);
         clk: in std_logic);
end seg_procesos;

architecture Behavioral of seg_procesos is
signal nueve, diez, seis: std_logic;
signal s1aux, s2aux:std_logic_vector(3 downto 0): = ”0000”;
begin
process (clk, diez)
begin
  if diez = '1' then s1aux <= "0000";
  elsif (clk =' 1') and clk'event then s1aux <= s1aux + 1;
  end if;
end process;
process(nueve, seis, clk)
begin
  if seis = '1' then s2<="0000";
  elsif and clk = ’1’ and clk'event and (nueve = '1') then s2aux <= s2aux + 1;
  end if;
end process;
nueve <= s1aux(3) and s1aux(0)
diez <= s1aux(3) and s1aux(1);
seis <= s2aux(2) and s2aux(1);
s1 <= s1aux;
s2 <= s2aux;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

El circuito de segundero que se resolvió no tiene funciones, contaría sin parar sin poder detener su funcionamiento. Para construir un sistema más funcional se propone agregar la conexión de tres botones. El problema queda propuesto como sigue:

Agregar al problema que ya resolvió, las siguientes entradas provenientes de botones:

Start: el segundero no iniciará su operación hasta que se oprima el botón Start, pero no tiene que quedarse oprimido para que opere el dispositivo.

Stop: el segundero detiene su operación.

Reset_segundero: mientras esté oprimido este botón, el segundero genera ceros a la salida.

Se ha planteado que **Start**, *stop* y reset_segundero sean botones y no interruptores porque los cronómetros comerciales generalmente se operan con solo una mano, y los botones son más fáciles de operar con una mano que los interruptores.

Estas funciones de *start, stop* y reset_segundero se pueden lograr agregando un *latch* JK y un par de compuertas or al circuito que ya se diseñó. El *latch* es necesario porque *start* y *stop* son botones que bajarán a 0 cuando se suelten, así que no pueden conectarse directo a Inc, que es el *enable* del incremento porque el contador se deshabilitaría cuando se suelte el botón.

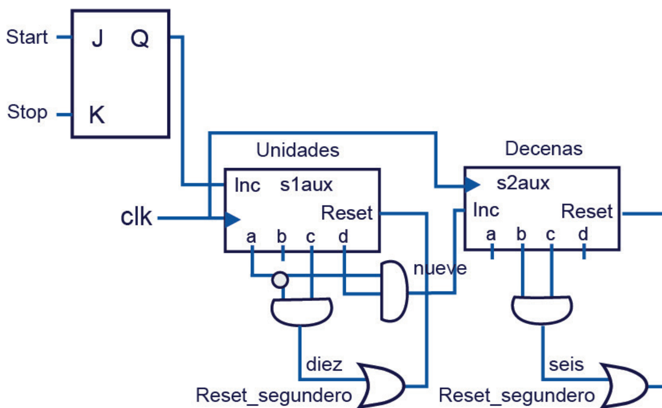


Figura 8.5 Diagrama esquemático de segundero funcional

©Editorial Digital Tecnológico de Monterrey

Note que en el código se agregó un and con Q al, cuando se detecta el 9 para evitar que las decenas avancen en caso de que se llegase a oprimir el *stop* cuando el contador de unidades está en 9. Esto no está en la figura, solo en el código.

Enseguida se presentan tres descripciones diferentes.

Definición estructural:

```
entity segun_con_botones is
Port (s1, s2: out std_logic_vector(3 downto 0);
        clk, Start, Stop, Reset_segundero : in std_logic);
end segun_con_botones;


architecture Behavioral of segun_con_botones is
component contador
Port (clk, inc, reset: in std_logic;
            s: out std_logic_vector(3 downto 0));
end component;
signal Q, diez, nueve, seis: std_logic;
signal s1aux, s2aux:std_logic_vector(3 downto 0):=”0000”;
begin
Q <= (Start and not Q) or ((not Stop) and Q);
e1:  contador port map (clk, Q, diez, s1aux);
--estas salidas auxiliares permiten utilizar
--los circuitos cominacionales requeridos para detectar el
--seis, el diez y el nueve
e2:  contador port map (clk, nueve, seis, s2aux);
nueve<= s1aux(3) and s1aux(0) and Q;
diez<= s1aux(3) and s1aux(1);
seis<= s2aux(2) and s2aux(1);
s1<=s1aux;
s2<=s2aux;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Definición por comportamiento:

```
entity segun_con_botones is
Port (s1, s2: out std_logic_vector(3 downto 0);
        clk, Start, Stop, Reset_segundero : in std_logic);
end segun_con_botones;


architecture Behavioral of segun_con_botones is
signal s1aux, s2aux:std_logic_vector(3 downto 0):=”0000”;
signal Q, diez, seis:std_logic:=’0’;
begin
Q <= (Start and not Q) or ((not Stop) and Q);
process (clk, diez, Reset_segundero)
begin
if(diez=’1’)or(Reset_segundero=’1’)then s1aux<=”0000”;
elsif (clk=’1’) and clk’event and (Q=’1’) then s1aux<=s1aux + 1;
end if;
end process;
process (clk, nueve, seis,  Reset_segundero)
begin
if(seis=’1’)or(Reset_segundero=’1’)then s2aux<=”0000”;
elsif clk=’1’ and clk’event and (nueve=’1’) and (Q=’1’)
        then s2aux<=s2aux + 1;
        else null;
end if;
end process;
nueve<= s1aux(3) and s1aux(0)
diez<= s1aux(3) and s1aux(1);
seis<= s2aux(2) and s2aux(1);
s1<=s1aux;
s2<=s2aux;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Otra solución por comportamiento:

```
 entity segun_con_botones is
Port (s1, s2: out std_logic_vector(3 downto 0);
         clk, Start, Stop, Reset_segundero : in std_logic);
 end segun_con_botones;


 architecture Behavioral of cronometro is
 signal  un_aux, dec_aux: std_logic_vector(3 downto 0);
 signal s1aux, s2aux:std_logic_vector(3 downto 0):=”0000”;
 signal Q:std_logic;
 begin
 Q <= (Start and not Q) or ((not Stop) and Q);
 process(clk, Reset_segundero, s1aux)
 begin
 if (Reset_segundero =’1’) or (s1aux=”1010”) then
                    s1aux<=”0000”;
                    elsif (clk=’1’) and clk’event and (Q=’1’) then
               s1aux<=s1aux+1;
          else null;
end if;
 end process;
 process (clk, reset, s2aux)
 begin
if (Reset_segundero =’1’) or (s2aux=”0110”)then
          s2aux<=”0000”;
               elsif (clk=’1’) and clk’event and (s1aux=”1001”) and
               (Q=’1’) then
          s2aux<=s2aux+”0001”;
else null;
end if;
 end process;
 s1<=s1aux;
 s2<=s2aux;
 end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

En una simulación, las señales de tipo std_logic inician indefinidas (U). Para simular un circuito como el del cronómetro existen varias alternativas:

1) Reemplazar el tipo std_logic por el tipo *bit*, ya que este inicia en ceros.

2) Provocar la inicialización a ceros desde la definición de las señales, por ejemplo:

```
signal s1,s2: std_logic_vector (3 downto 0):= "0000";
```

3) Incluir entradas de *reset*, y en la simulación asegurarse de que ocurra un *reset* que esté en 1 y luego regrese a 0. Para esto es posible proporcionar al simulador comandos como el siguiente:

```
force reset 0 0, 1 15, 0 40
```

Con el comando force la señal de reset inicia en 0, luego cambia a 1 en tiempo=15 y luego vuelve a 0 en tiempo=40.

Una vez que ocurra el *reset*, las salidas correspondientes estarán en ceros y podrá realizarse la cuenta. Para simular un *latch* también es preciso cambiar el valor indefinido (U) de la salida Q, de lo contrario, el simulador siempre marcará como indefinida esta salida. Lo anterior puede realizarse de la siguiente manera:

```
signal Q: std_logic:='0';
```

Por otra parte, para simular la señal de reloj puede hacer lo siguiente:

```
force clk 0 0, 1 10 -repeat 20
```

Aquí se está asociando a clk una señal que inicia en 0, cambia a 1 en tiempo=10 y tiene un período de 20 (con la palabra *repeat* no se indica la cantidad de ciclos, sino el tamaño del período de la señal).

Si tiene problemas para simular el *latch* JK puede modificar la ecuación del JK por una que no niegue la salida en las entradas 1-1, como se hizo con el *latch* SR modificado.

©Editorial Digital Tecnológico de Monterrey

8.2 Aplicación del circuito one shot

Conteo con botón. El propósito de este circuito es contar el número de veces que se oprima un botón. Este botón está conectado a la entrada del circuito, además de que también se cuenta con una señal de reloj y otro botón para la función de *reset*. El botón puede estar oprimido por cualquier cantidad de tiempo, pero solo se contabilizará una vez que se ha oprimido. La salida de este circuito son 8 *bits* que tienen la cuenta en binario de las veces que se oprimió el botón. Al oprimir el botón de *reset* la cuenta se debe poner en ceros.

Se requiere un registro contador. Por el rebote, el botón no se debe conectar a la entrada de reloj del contador. Como el botón estará oprimido por tiempo indefinido, es conveniente conectarlo a un circuito *one shot*. La señal de *reset* que proviene también de un botón puede conectarse en forma directa ya que se procesa de forma asincrónica, así que, al ocurrir el rebote, no importa si se capta en forma repetida, pues solo provocará la producción repetitiva de ceros durante el instante que perdura el rebote. Hasta que el *reset* quede en cero estable podrá iniciar la cuenta.

El diseño esquemático es el siguiente (E corresponde a enable y R a *reset*):

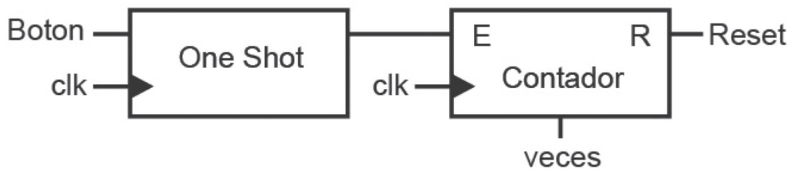


Figura 8.6 Diagrama esquemático del circuito que cuenta pulsos de un botón

©Editorial Digital Tecnológico de Monterrey

Hay algunas diferencias en la sintaxis que detecta el compilador de las distintas versiones de VHDL. Si el siguiente código le presenta un problema de sintaxis, intercambie botonOS='1' por (botonOS='1') que corresponde a las reglas originales en cuanto a que los operadores no tienen prioridad y se evalúan de izquierda a derecha en la simulación.

```
entity boton is
     port(clk, boton, reset: in std_logic;
               veces: out std_logic_vector(7 downto 0));
end boton;

architecture Behavioral of boton is

     component oneshot is
          Por(Din, clk : in  STD_LOGIC;
                       Qout : out  STD_LOGIC);
     end component;

signal contador: std_logic_vector(7 downto 0);
signal botonOS: std_logic;

begin

botonOneShot: oneshot port map(boton, clk, botonOS);

     process(clk,botonOS,reset)
          begin
               if reset =' 1' then
                    contador <= "00000000";
               elsif clk =' 1' and clk'event and botonOS = '1' then
                    contador <= contador+1;
               end if;
     end process;

veces <= contador;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Descripción del *one shot*:

```
entity one_shot is
    Port(Din, clk : in  STD_LOGIC;
         Qout : out  STD_LOGIC);
end one_shot;

architecture Behavioral of one_shot is
signal Q1, Q2, Q3: std_logic;
begin
process(clk)
begin
 if clk = '1' and clk'event then
           Q1 <= Din;
     Q2 <= Q1;
     Q3 <= Q2;
 end if;
end process;
Qout <= Q1 and Q2 and not Q3;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

8.3 Análisis de un circuito

Captura de dos números BCD. Se ha propuesto diseñar un fragmento de una bronto-calculadora que consista en un circuito y tenga como propósito que, con solo cuatro *switches*, se capturen dos números y cada número conste de dos dígitos BCD y los deje en dos registros de ocho *bits* (la calculadora haría operaciones con ambos números).

Para que con este circuito se capturen los dos números de dos dígitos BCD utilizando cuatro *switches*, se seguirían los siguientes pasos:

a) Configurar (en binario) un número BCD en los 4 *switches*

b) Oprimir <Enter>

c) Al repetir los pasos de los incisos a y b por 4 veces consecutivas los cuatro números BCD deben quedar acomodados en dos registros, el primer dígito se almacena en los cuatro *bits* más significativos del primer registro, el segundo dígito se almacena en los cuatro *bits* menos significativos del primer registro, el tercero en la parte más significativa del segundo registro y el cuarto en la menos significativa del segundo registro.

d) Si se volviera a oprimir <Enter> el proceso volvería a empezar re-escribiendo la parte más significativa del primer registro.

e) Al seleccionar un *reset* se reinicializa el proceso de captura conduciendo el procedimiento al paso a.

©Editorial Digital Tecnológico de Monterrey

Un posible diseño esquemático de este circuito de captura que se propone analizar, es el siguiente:

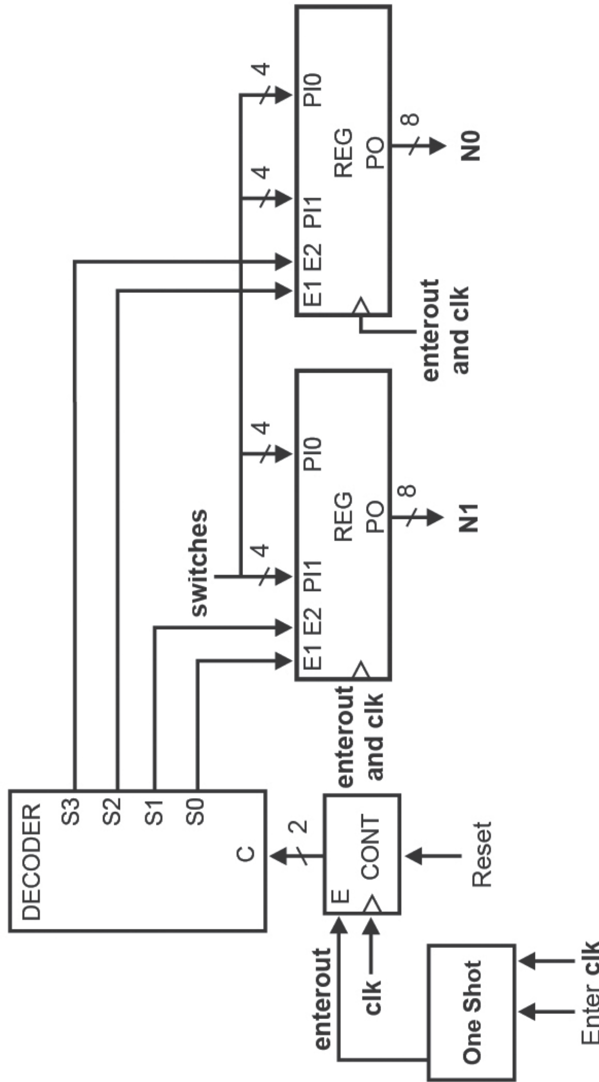


Figura 8.7 Diseño esquemático de circuito de captura de números BCD configurados en *switches*

©Editorial Digital Tecnológico de Monterrey

Análisis del circuito. El registro REG de 8 *bits* cuenta con dos entradas de Enable E1 y E2. Cuando E1 está en 1 se carga sincrónicamente la entrada paralela PI1 en los cuatro *bits* más significativos, cuando E2 está en 1 se carga la entrada paralela PI0 en los cuatro *bits* menos significativos. La salida paralela de 8 *bits* es PO.

El registro contador de dos *bits* se incrementa en 1 sincrónicamente cuando recibe una señal de <Enter> que proviene de un botón. Solamente se incrementa en una unidad por cada vez que se oprime el botón, sin importar cuánto tiempo se quede oprimido el botón <Enter>; lo anterior se logra con el *one shot*. El contador tiene un *reset* asincrónico para forzar que su salida sea cero, causando que S0 sea 1, que selecciona E1 del primer registro, primer paso del procedimiento.

El *one shot* hace que cuando se pulse <Enter> se produzca una señal que esté en 1 durante un período de reloj, lo que causará que en la siguiente transición de reloj:

a) El contador se incremente para que la salida del DECODER esté lista para la siguiente carga (ya que ocurre un uno en la entrada que habilita su incremento).

b) Se cargue el dígito BCD de los *switches* en la posición que corresponda, la menos significativa o la más significativa de alguno de los dos registros, dependiendo cuál salida del DECODER esté en 1 (al inicio y después de un *reset* es S0, ya que habilita la señal de reloj).

Como ejercicio se propone diseñar el diagrama de tiempo de este circuito para que se aprecie cómo al pulsar <Enter> se carga un dígito con la entrada habilitadora actual y se deja lista para la siguiente carga. Se realiza una definición híbrida, una parte estructural y otra por comportamiento.

©Editorial Digital Tecnológico de Monterrey

```
entity calculadora is
     Port ( ENTER, CLK, RESET: in  STD_LOGIC;
         switches : in  STD_LOGIC_VECTOR (3 downto 0);
         N1, N0 : out  STD_LOGIC_VECTOR (7 downto 0));
end calculadora;

architecture Behavioral of calculadora is

component oneshot is
Port ( Din, clk : in  STD_LOGIC;
           Qout : out  STD_LOGIC);
end component;

component REG is
     Port ( PI1, PI0 : in  STD_LOGIC_VECTOR (3 downto 0);
                CLK, E1, E2: in  STD_LOGIC;
                P0 : out  STD_LOGIC_VECTOR(7 downto 0));
end component;

component cont is
    Port ( E, CLK, RESET: in  STD_LOGIC;
          C : out  STD_LOGIC_VECTOR (1 downto 0));
end component;

signal c : STD_LOGIC_VECTOR (1 downto 0):=”11”;
signal s0, s1, s2, s3, aux: STD_LOGIC;

begin
```

©Editorial Digital Tecnológico de Monterrey

```
decoder: process (c) begin
                if c = "00" then
                     s0 <= '1';
                     s1<='0';
                     s2<='0';
                     s3<='0';
                elsif c = "01" then
                     s0 <= '0';
                     s1<='1';
                     s2<='0';
                     s3<='0';
                elsif c = "10" then
                     s0 <= '0';
                     s1<='0';
                     s2<='1';
                     s3<='0';
                elsif c = "11" then
                     s0 <= '0';
                     s1<='0';
                     s2<='0';
                     s3<='1';
                else null;
                end if;
             end process;

oneshot: oneshot port map (enter, clk, enterout);
     contador: cont port map(enterout, clk, reset,c);
     aux<=clk and enterout;
     reg1: reg port map(switches, switches, aux, s0, s1, N1);
     reg2: reg port map(switches, switches, aux, s2, s3, N0);

end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

A continuación, se definen los componentes estructurales:

```
                        cont<=cont+1;
                end if;
        end process;
c <= cont;
end Behavioral;

entity REG is
 Port(switches: in  STD_LOGIC_VECTOR (3 downto 0);
      CLK, E1, E2: in  STD_LOGIC;
      PO: out STD_LOGIC_VECTOR(7 downto 0));
end reg;

architecture Behavioral of REG is

signal PI1,PI0: std_logic_vector(3 downto 0);

begin

        process(clk,e1,e2)
            begin
                if clk = '1' and clk'event then
                    if e1 = '1' then
                        PI1 <= switches;
                    elsif e2 = '1' then
                        PI0 <= switches;
                    else null;
                    end if;
                    else null;
                end if;
        end process;

PO <= PI1&PI0;

end Behavioral;
```

Recuerde que mayúsculas y minúsculas son indistintas en el código, así que E1 y e1 simbolizan lo mismo.

©Editorial Digital Tecnológico de Monterrey

8.4 Uso de la función de rotación

Rotación. En este problema se diseña un circuito que genera la secuencia de un 1 rotando (este tipo de funciones se utilizan por los circuitos controladores de tableros de LEDs), la rotación ocurre en las transiciones positivas de una señal externa de reloj.

La secuencia que debe generar es:

```
1000000000000000
0100000000000000
0010000000000000
0001000000000000
0000100000000000
...
0000000000000001
1000000000000000
0100000000000000
```

Además, este sistema debe responder a las señales generadas por los siguientes botones:

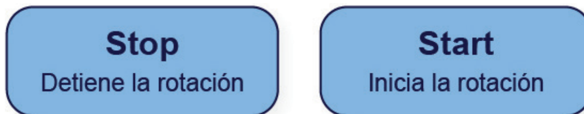


Figura 8.8

El sistema debe iniciar la rotación cuando ocurra un *start* y detenerse con la señal de *stop*. De iniciar con un *Start*, el sistema iniciará donde se había quedado. A continuación, se muestra la definición del puerto:

©Editorial Digital Tecnológico de Monterrey

```
entity rotador is
    Port ( Start : in std_logic;
              Stop : in std_logic;
         clk: in std_logic;
              salida : out std_logic_vector(15 downto 0));
end rotador;
```

Como *start* y *stop* son funciones ejercidas por botones, se requiere un *latch* para "memorizar" que ocurrió un pulso por cualquiera de estos botones. Si se conectaran estos botones en forma directa, al soltarlos se perdería su función. El diseño esquemático es:

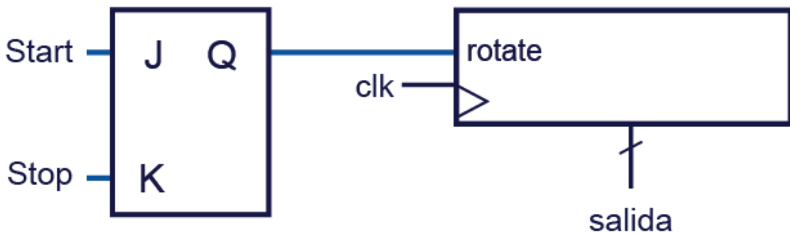


Figura 8.9 Diseño esquemático del círculo de rotación

Al registro que se usa se le fuerza una entrada inicial de 1000000000000000.

©Editorial Digital Tecnológico de Monterrey

```
entity rot is
    Port (start : in std_logic:='0';
           stop : in std_logic:='0';
      clk: in std_logic:='0';
           salida : out std_logic_vector(15 downto 0));
end rot;

architecture Behavioral of rot is
signal Q :std_logic:='0';
signal reg: std_logic_vector(15 downto
0):="1000000000000000";
begin
Q <= (start and not Q1) or (not stop and Q);
 salida<= reg;
process(clk)
begin
  if (clk='1') and clk'event and (Q='1')
  then reg<=reg(0)&reg(15 downto 1);
  end if;
end process;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

8.5 Secuencia maestro-esclavo

Rotación maestro-esclavo. Algunos circuitos que sirven para refrescar tableros o pantallas requieren rotar sus datos repetidamente. En este problema se definen dos registros A y B de 4 *bits* cada uno. Inician con 1000 respectivamente. El registro A rota su salida una posición a la derecha con cada transición positiva del reloj. Cada vez que A muestre 0001, el registro B rota una posición a la derecha.

En cada transición positiva del reloj las salidas de A y B son las siguientes:

A	B
1000	1000
0100	1000
0010	1000
0001	0001
1000	0100
0100	0100
0010	0100
0001	0100
1000	0010
....	....
0001	0001
1000	0001
....	....
0001	1000
1000	1000
....	

El esquemático lo constituyen dos registros que rotan, uno esclavo del otro:

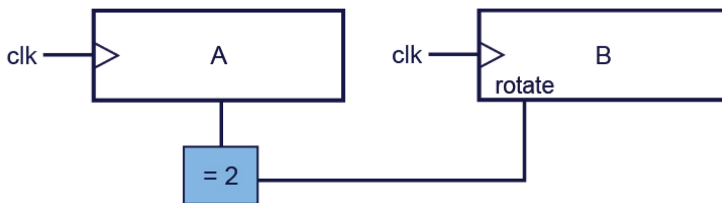


Figura 8.10 Diseño esquemático del circuito de rotación maestro-esclavo

©Editorial Digital Tecnológico de Monterrey

En VHDL este circuito se describe como se presenta a continuación (note el momento en el que ocurre la rotación en ambos registros):

```
entity registros is
    Port(Aout: out std_logic_vector(3 downto 0);
              Bout: out std_logic_vector(3 downto 0);
       CLK:in std_logic);
end registros;


architecture comportamiento of registros is

begin

     process(CLK)
          begin
          if(CLK = '1' and CLK'event) then
               A <= A(0) & A(3 downto 1);
               end if;
          end if;
     end process;

     process(CLK)
          begin
          if(CLK = '1' and CLK'event) then
if A = "0010" then
                         B <= B(0)& B(3 downto 1);
                         else null;
               end if;
          else null;
          end if;
     end process;
     Aout <= A;
     Bout <= B;
     end comportamiento;
```

©Editorial Digital Tecnológico de Monterrey

8.6 División de un número entero entre un número que es una potencia de 2

En cualquier sistema numérico posicional, como el decimal y el binario, una división aritmética entre un número que represente una potencia a la "n" de la base se realiza recorriendo el punto a la izquierda "n" posiciones. Si se busca un cociente entero sin punto, se desechan los "n" dígitos de más a la derecha. Por ejemplo:

$$\frac{101100101}{2^3} = \frac{101100101}{1000} = 101100$$

Se solicita diseñar un circuito que realice una división entera A / B.

Las entradas al circuito son:

A, número entero de 8 *bits*

B, número entero de 4 *bits*. B es una potencia de 2, no tiene que validarlo. B solo puede ser: 0001 o 0010 o 0100 o 1000.

Señal de reloj Clk y,

Señal de ***start***, que proviene de un *switch*, debe iniciar en 1. Mientras permanezca en 1 se realiza la carga de datos, al bajar a 0 comienza la división.

Las salidas del circuito son:

Z, número entero de 8 *bits* que representará el cociente de la división.

V, indica que el resultado Z es válido.

©Editorial Digital Tecnológico de Monterrey

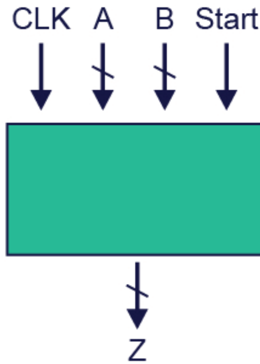


Figura 8.11 Puerto del circuito divisor

El procedimiento a seguir es el siguiente:

a) Cargar ambos circuitos operando en dos registros, rega y regb, cuando start=1.

b) Mientras el *bit* menos significativo de B sea 0, recorrer a la derecha ambos números.

c) El *bit* menos significativo de B es la salida V.

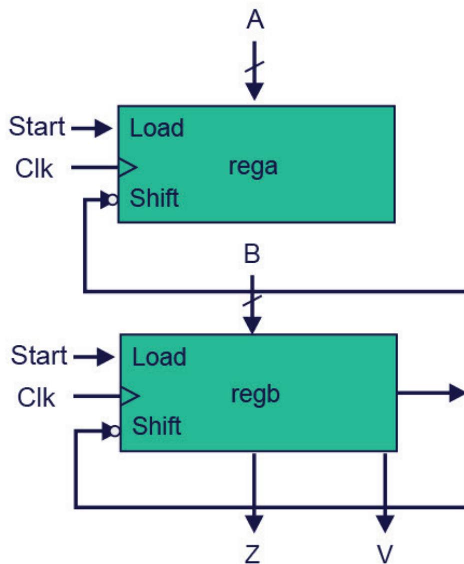


Figura 8.12 Diseño esquemático secuencial del circuito divisor

©Editorial Digital Tecnológico de Monterrey

```
entity div2n is
    Port(A: in std_logic_vector(7 downto 0);
         B: in std_logic_vector(3 downto 0);
         Clk: in std_logic;
         Start: in std_logic;
         Z: out std_logic_vector(7 downto 0);
         V: out std_logic);
end div2n;

architecture Behavioral of div2n is
signal rega, regb: std_logic_vector(7 downto 0);
begin
z <= rega;
process(clk, start)
begin
if start = '1' then rega <= A; regb <= B; --la entrada load se esta usando
                                          -- como "carga paralela asincrónica"
elsif (clk = '1') and clk'event and (regb(0) = '0')
 then
 regb <= '0'&regb(3 downto 1);
 rega <= '0'&rega(7 downto 1);
end if;
end process;
v <= '1' when b(0)='1' else '0';
end Behavioral;
```

Si el circuito se va a implementar y presenta algún problema entonces se recomienda cerrar el *if* con un "*else null*".

©Editorial Digital Tecnológico de Monterrey

Otra solución trivial (porque es difícil de generalizar para números de mayor longitud), combinacional, es la siguiente:

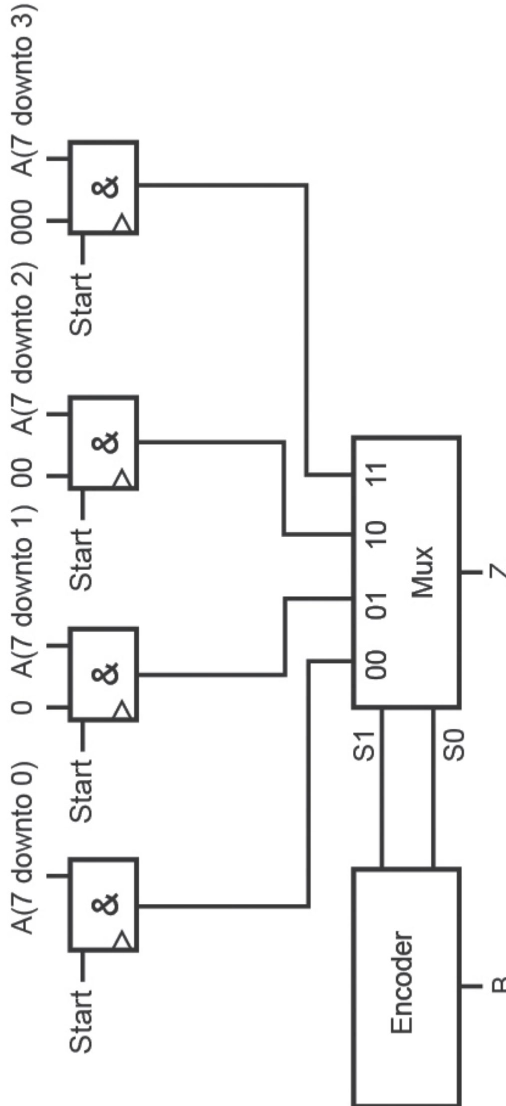


Figura 8.13 Diseño esquemático del circuito de rotación maestro-esclavo

©Editorial Digital Tecnológico de Monterrey

La descripción en VHDL puede resumir el *encoder* y el *multiplexer* en una misma estructura, como se muestra a continuación:

```
entity problema1 is
     port(A: in std_logic_vector(7 downto 0);
          B: in std_logic_vector(3 downto 0);
          start: in std_logic;
          Z: out std_logic_vector(7 downto 0));
end problema1;

architecture Behavioral of problema1 is

begin

process(start)
     begin
               if start = '1' then
                    if B = "0001" then
                       Z <= A;
                    elsif B = "0010" then
                          Z <= '0'&A(7 downto 1);
                    elsif B = "0100" then
                          Z <= "00"&A(7 downto 2);
                    elsif B = "1000" then
                          Z <= "000"&A(7 downto 3);
                    else
                         null;
                    end if;
     end process;
V <= not start;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

8.7 Entrada serial salida decodificada

Se solicita diseñar un circuito que recibe una secuencia de *bits* en forma serial sincrónica con una señal clk. Se requiere decodificar el número que se forme cada ocho *bits*, para esto se tiene un componente llamado DecodNum. Cada vez que se reunan ocho *bits*, a DecodNum se le dará una señal de *Enable* con duración de un pulso de reloj, indicando que el número a decodificar está completo. Las cadenas de ocho *bits* que se decodificarán ingresan a partir del *bit* menos significativo. La salida decodificada debe ser válida y estable durante ocho ciclos de reloj a partir del octavo ciclo. La secuencia de *bits* se recibe a partir de que reset haya cambiado de uno a cero, esto establece la sincronía. La señal de reloj es recibida del mismo dispositivo que genera la transmisión serial.

©Editorial Digital Tecnológico de Monterrey

En el diseño esquemático, el componente DecodNum solo se indica con un bloque:

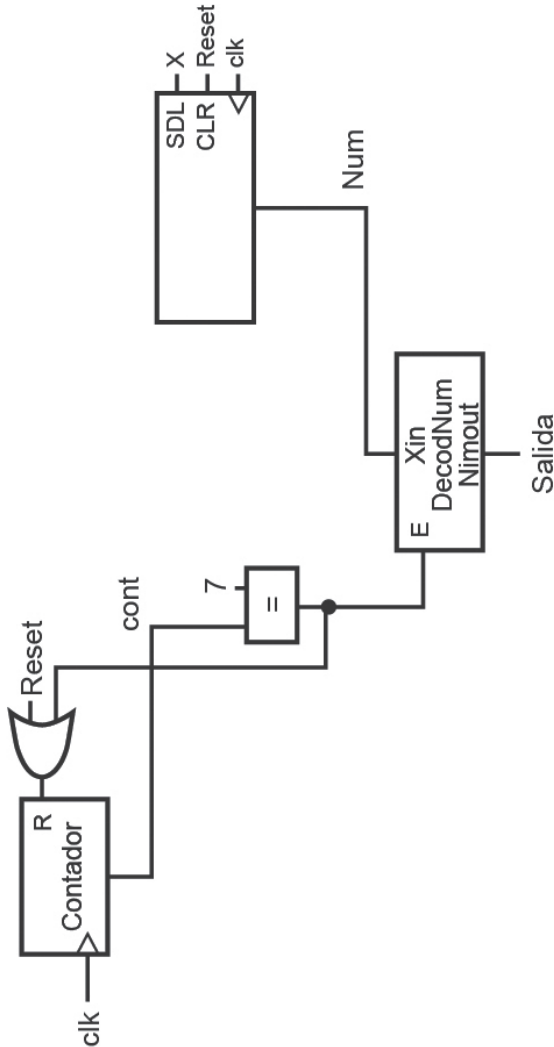


Figura 8.14 Diseño esquemático del decodificador de números

La codificación de esta arquitectura en VHDL es la siguiente (omitiendo el código de DecodNum):

©Editorial Digital Tecnológico de Monterrey

```
entity decodificadorserial is
    Port(Reset, x: in STD_LOGIC;
            clk: in STD_LOGIC;
            salida: out STD_LOGIC_VECTOR (7 downto 0));
end decodificadorserial;

architecture Behavioral of decodificadorserial is

component decodnum is
   Port(E: in std_logic;
        xin: in std_logic_vector (7 downto 0);
                Numout: out std_logic_vector (7 downto 0));
end component;

signal num: std_logic_vector(7 downto 0);
signal cont: std_logic_vector(3 downto 0);
signal enable: std_logic:='0';

begin

     process(clk, reset)
     begin
                if reset = '1' then cont<="0000";
                     num <= "00000000";
elsif clk = '1' and clk'event then
num <= num(6 downto 0) & x;
```

```
if (cont=7) then
                          cont <= "0000";
                     else
                          cont <= cont+1;
                     end if;
                end if;
     end process;
decodificacion: decodnum port map(enable,num,salida);
enable <= '1' when cont = 7 else '0';
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

8.8 Registro de arquitectura especial

Controlador de LEDs. Se le solicita diseñar un control para ocho LEDs. El control cuenta con 6 *switches*. La función que activa el *switch* es:

Switch	Función
Reset:	Cuando es puesto en 1 junto con alguno de los modos, los LEDs inician correctamente, de lo contrario seguirán una secuencia imprevista.
Modo1:	La secuencia se muestra en la gráfica.
Modo2:	La secuencia se muestra en la gráfica.
Modo3:	La secuencia se muestra en la gráfica.
Modo4:	La secuencia se muestra en la gráfica.

Tabla 8.1

El cambio en los LEDs ocurre con la frecuencia dada por un reloj que se conecta al circuito.

©Editorial Digital Tecnológico de Monterrey

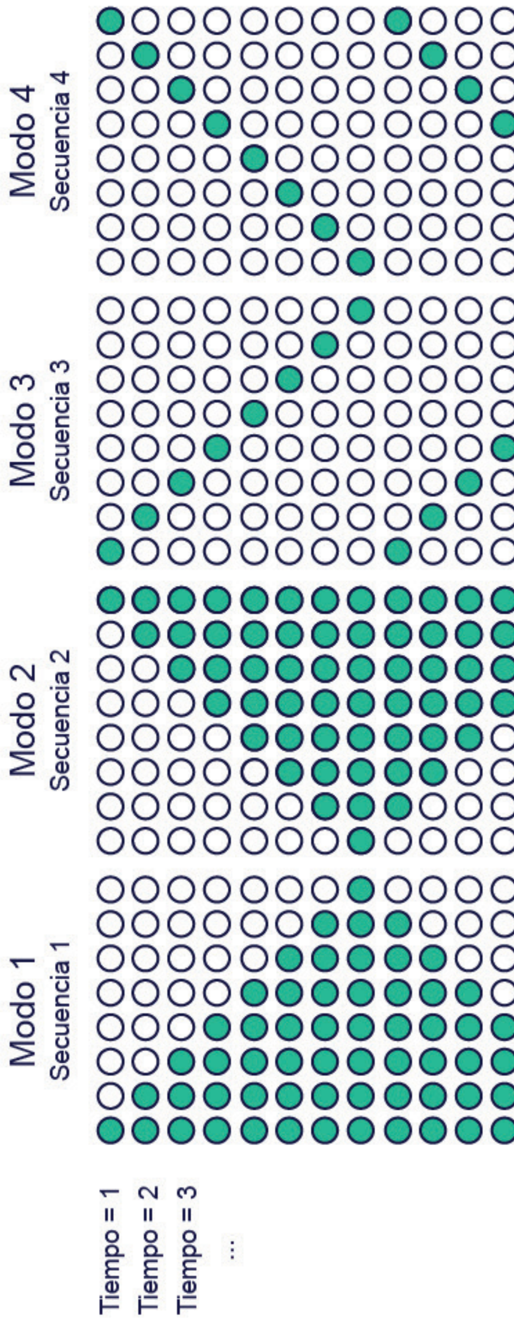


Figura 8.15

©Editorial Digital Tecnológico de Monterrey

El diseño esquemático puede ser muy general, no es necesario especificar la arquitectura interna del registro Reg, ya que la arquitectura interna puede ser descrita en VHDL a partir de su comportamiento.

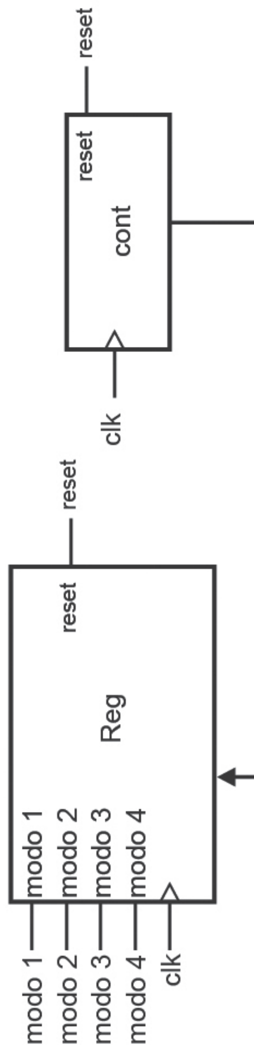


Figura 8.16 Diseño esquemático para seguir el patrón de iluminación

©Editorial Digital Tecnológico de Monterrey

El contador debe ser de cuatro *bits*. Al contar de 0 a 7 hace una función, de 8 a 15 otra y después del 15 continúa el 0.

```
entity control_leds is
  Port (clk, modo1,modo2,modo3,modo4,reset:in std_logic;
          Leds: out std_logic_vector(7 downto 0) );
end control_leds;

architecture Behavioral of control_leds is
signal reg: std_logic_vector(7 downto 0);
signal cont: std_logic_vector(3 downto 0);
begin
```

©Editorial Digital Tecnológico de Monterrey

```
process(clk, reset)
     begin
          if reset = '1' then
               cont <= "000";
               if (modo1 = '1') or (modo3='1')  then  reg<="10000000";
                     elsif (modo2 = '1') or (modo4 = '1')  then  reg <= "00000001";
               else reg <= "00000000";
               end if;
elsif clk='1' and clk'event then
               cont <= cont+1;
               if modo1 = '1' and (cont <= 7) then
               reg <= '1' & reg(7 downto 1);
               elsif modo1 = '1' and (cont > 7) then
reg<=reg(6 downto 0) & '0';
elsif modo2='1' and (cont <= 7) then
               reg <= reg(6 downto 0) & '1';
               elsif modo2='1' and (cont > 7) then
               reg <= '0' & reg(7 downto 1);
               elsif modo3='1' then
               reg <= '0' & reg(7 downto 1);
               elsif modo4 = '1' then
               reg <= reg(6 downto 0) & '0';
               else null;
               end if;
else null;
          end if;
     end process;
Leds <= reg;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

8.9 Bomba de gasolina

Se le solicita diseñar un circuito para un prototipo de bomba de gasolina. Para facilitar el diseño, supondremos que la bomba opera basándose en litros solamente, sin considerar el cobro de la gasolina.

La bomba funciona en dos modos.

Manual: en este modo el *switch* de modo debe permanecer en "off", el operador cuenta con un *botón* de *abre* y otro de *cierra* para accionar o cerrar la válvula que se encuentra en la manguera.

Automático: para operar en este modo es necesario poner en "on" el *switch* de *modo*. El operador cuenta con 8 *switches* para colocar un número binario que indica la cantidad de litros a servir. El operador debe oprimir el *botón* de *abre* para iniciar el servicio y la válvula debe cerrar automáticamente al llegar al número de litros que se indicó en los 8 *switches*.

La válvula requiere de una señal externa para abrirse o cerrarse. Hay una gran variedad de válvulas industriales de este tipo; esta es una normalmente cerrada. Con un 1 en su entrada se abre, con 0 se cierra.

El circuito que usted diseñará tiene dos propósitos:

- Operar la válvula.
- Mostrar en binario, en 8 Leds la cantidad de litros que se estén vertiendo.

El circuito recibe las siguientes conexiones a las entradas del circuito:

- **Botón *reset***, para poner en ceros las salidas.
- **Botón abre.**
- **Botón cierra.**

©Editorial Digital Tecnológico de Monterrey

- ***Switch* modo.**
- **8 *switches***, para colocar la cantidad de litros a servir (en binario).
- **Un reloj**, que tiene un período de 1 seg, y las siguientes conexiones a las salidas:
- **V**, que se conecta a la válvula.
- **8 LEDs**, correspondientes al número binario que indicará la cantidad de litros que se han "servido" a un automóvil.

Notas:

1. La gasolina sale de la manguera a la razón de 1 litro/seg.
2. La válvula debe abrirse al iniciarse el servicio y cerrarse hasta el final.
3. Siempre se vierten litros completos.

El puerto del circuito es:

```
Entity gas is
Port (abre, cierra, modo, clk: in std_logic;
        Cant_lit: in std_logic_vector (7 downto 0);
        Litros_serv: out std_logic_vector (7 downto 0)),
end gas;
```

Se requiere un *latch* para "memorizar" el pulso abre. Un registro contador puede llevar la cuenta de los litros servidos, que puede ser iniciada a ceros por una función de *reset* asincrónico. Para cerrar la válvula hay dos condiciones, en el modo manual se cierra con el botón cierra, en el modo automático se cierra cuando la cuenta coincida (salida I) con la cantidad de litros a verter colocados en

©Editorial Digital Tecnológico de Monterrey

switches. La salida V coincide con la señal que habilita la cuenta ascendente en el contador. El diseño esquemático resultante es:

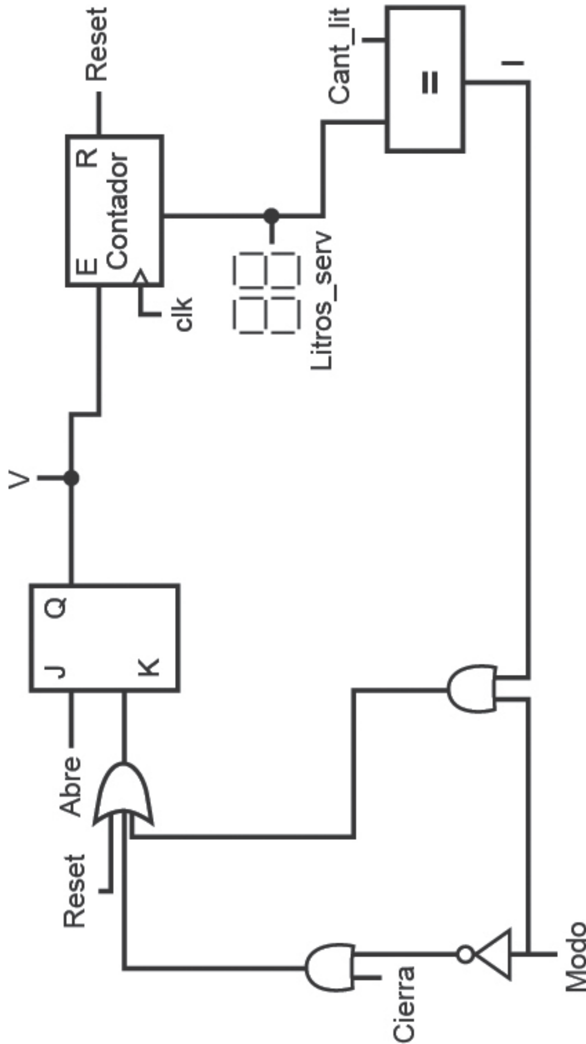


Figura 8.17 Diseño esquemático del controlador de la bomba de gasolina

Se aclara que la salida se presenta en LEDs, aunque en el diseño esquemático se muestren *display*s de siete segmentos por ser una salida más representativa de una bomba real.

©Editorial Digital Tecnológico de Monterrey

Codificación en VHDL:

```
entity gasolina is
     port( abre,cierra,modo,clk,reset: in std_logic;
                Cant_lit: in std_logic_vector(7 downto 0);
                Litros_serv: out std_logic_vector(7 downto 0);
                V:out std_logic);
end gasolina;

architecture Behavioral of gasolina is

signal cont: std_logic_vector(7 downto 0);
signal Q,I,K: std_logic;

begin
Q <= (Abre and not Q) or (not K and Q);
K <= Reset or (Cierra and not Modo) or (Modo and I);
I <= '1' when (cont = Cant_lit) else '0';

     process(clk,Q,reset)
          begin
               if reset = '1' then
                    cont <= "00000000";
               elsif clk = '1' and clk'event and (Q = '1') then
                    cont <= cont + 1;
               end if;
     end process;

V <= Q;
Litros_serv <= cont;

end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

8.10 Juego de basketball para niños

El dueño de una cadena de negocios de juegos infantiles que funcionan con *tokens* (monedas que se insertan en un dispositivo electromecánico para pagar por el juego) le solicita que diseñe un circuito para sus juegos de *basketball*. El juego ya tiene alambrados sus dispositivos de entrada y salida, solo le falta el circuito controlador. Se tiene un botón de *start* (St) y tiene un sensor en la canasta que detecta que la pelota haya sido insertada (SP), esta señal tiene una duración de más de un segundo. El diseño del juego incluye un dispositivo electromecánico que se encarga de proveer al jugador de pelotas para luego recircularlas. Al oprimir el botón de *start*, el jugador dispone de 1 minuto para jugar. En este tiempo se le mostrará en un *display* de 7 segmentos cuántas pelotas insertó en la canasta (aunque para este ejercicio se mostrará en binario), cada canasta registra un punto.

Para simplificar el problema supondremos que, en un minuto, a lo más, se alcanzarán a anotar 9 puntos. Cuando se cumple el minuto se envía una señal hacia una compuerta mecánica que sujeta las pelotas para que el jugador ya no pueda seguir jugando, y aunque se llegara a insertar una pelota ya no se aumentarían los puntos. No se diseñará la validación del *token* que se inserte (o la llave, o la tarjeta). Se supondrá que cuando se oprime *start* ya está validada la forma de acceso al juego. La señal de *reset* debe poner en ceros al contador de puntos y de tiempo, así como enviar la señal al actuador mecánico que suelta las pelotas (P). Se cuenta con un reloj que tiene una frecuencia de 1 Hz. El minuto se contará en segundos (Seg) y se mostrará en binario para simplificar el problema. Los puntos (P) también se muestran en binario. Así mismo, hay que controlar un sujetador de canasta (C) en 1 abre la canasta para que puedan insertar, en 0 la cierra para que termine el juego.

La interfaz del circuito es la siguiente:

- Entradas: CLK, sensor de pelotas anotadas (SP), botón de *start* (St), *reset* (Res).
- Salidas: tiempo (Seg), puntos (P) Sujetador de canasta (C).

©Editorial Digital Tecnológico de Monterrey

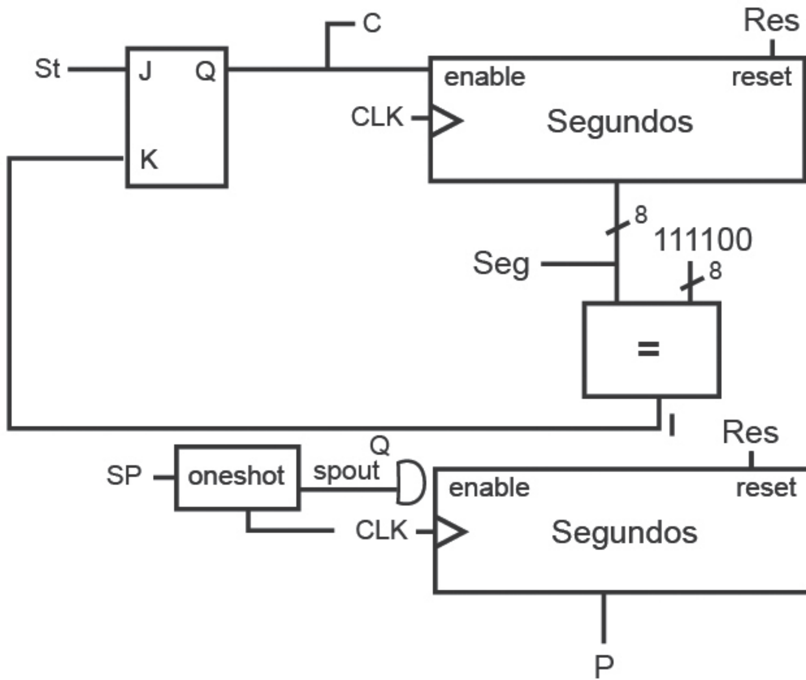


Figura 8.18 Diseño esquemático del controlador del juego de basketball

Codificación en VHDL:

```
entity playland is
    Port( St: in STD_LOGIC;
          Res: in STD_LOGIC;
          SP: in STD_LOGIC;
          Clk: in STD_LOGIC;
          C: out STD_LOGIC;
```

©Editorial Digital Tecnológico de Monterrey

```
P: out STD_LOGIC_VECTOR (3 downto 0);
           Seg: out  STD_LOGIC_VECTOR (5 downto 0));
end playland;

architecture Behavioral of playland is
component oneshot is
   port(clk, din: in std_logic; qout: out std_logic);
end component;
signal Q, I, spout: std_logic;
signal puntos: std_logic_vector(3 downto 0);
signal segundos: std_logic_vector(5 downto 0);
begin
Q<= (St and not Q) or (not I and Q);
I<= '1' when segundos = "111100" else '0';
sensor: oneshot port map (Clk, SP, spout);
--contador de tiempo
process(Clk, Res, Q)
begin
  if Res = '1' then segundos <= "000000";
  elsif Clk = '1' and Clk'event and (Q = '1')
     then segundos <= segundos + 1;
  end if;
end process;
process(Clk, spout, Q,Res)
begin
  if Res = '1' then puntos <= "0000";
  elsif Clk = '1' and Clk'event and (spout =' 1') and (Q = '1')
     then puntos <= puntos + 1;
  end if;
end process;
C <= Q;
P <= Puntos;
Seg <= segundos;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

8.11 Diseño de una alarma

Se le solicita diseñar un sistema para hacer sonar una alarma. Se tienen tres *switches*; si se selecciona el primero la alarma sonará 3 segundos, si se selecciona el segundo la alarma sonará 5 segundos, y si se selecciona el tercero la alarma sonará 8 segundos. Diseñe un circuito al cual ingresen los tres *switches*, una señal de reloj con período de 1 segundo, y que como salida se genere una señal que se utilizará como *Enable* para el dispositivo que producirá el sonido de la alarma. Este *Enable* deberá estar en 1 los segundos que indiquen el *switch* seleccionado. También habrá un botón de start que causará que inicie la operación del circuito, haciendo que el *Enable* suba a 1 por el tiempo indicado.

El puerto del circuito es:

```
entity problema5 is
     port(sw1,sw2,sw3,clk,start: in std_logic;
               enable: out std_logic);
end problema5;
```

Un diseño esquemático puede ser:

©Editorial Digital Tecnológico de Monterrey

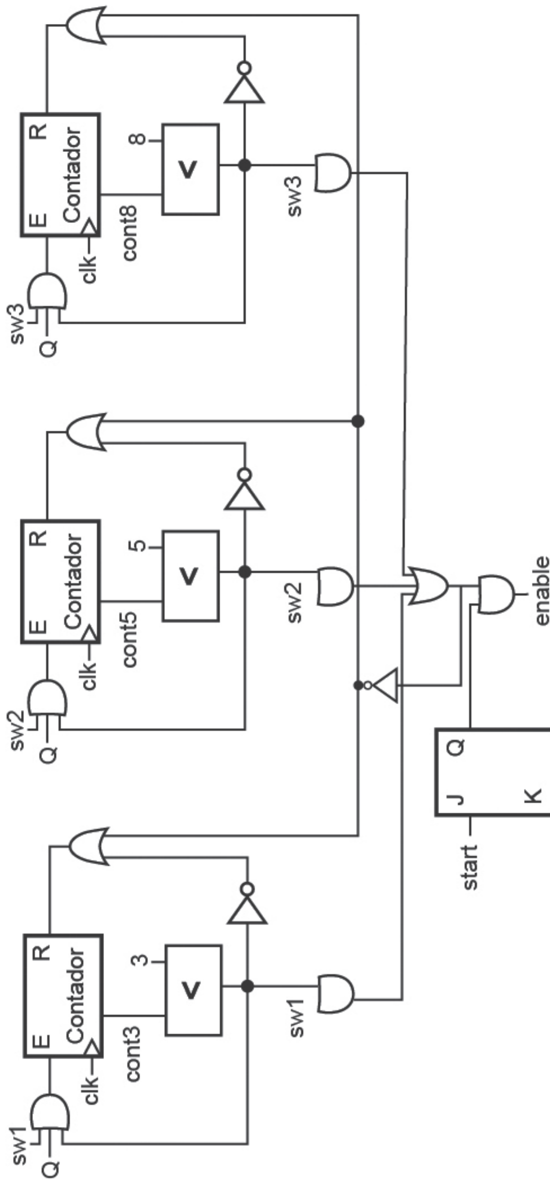


Figura 8.19 Diseño esquemático del controlador de la alarma

©Editorial Digital Tecnológico de Monterrey

La arquitectura en VHDL correspondiente a este diseño es la siguiente (este diseño hace que el circuito funcione recurrentemente, no tiene reset ni stop. Otra forma sería llevar una señal a la entrada K):

```
architecture Behavioral of problema5 is

signal cont3,cont5,cont8: std_logic_vector(3 downto 0): = "0000";
signal q:std_logic: = '0';
begin
q <= (start and not q) or q;

    process(sw1,sw2,sw3,clk,start)
        begin
        if q = '1' then
            if clk = '1' and clk'event then

                if sw1 = '1' and cont3 < 3 then
                    enable <= '1';
                    cont3 <= cont3 + 1;
                elsif (sw1 = '0' and sw2 = '0' and sw3 = '0') or cont3 = 3 then
                    enable <= '0';
                    cont3 <= "0000";
                end if;
                if sw2 = '1' and cont5 < 5 then
                    enable <= '1';
                    cont5 <= cont5 + 1;
                elsif (sw1 = '0' and sw2 = '0' and sw3 = '0') or cont5 = 5 then
                    enable <= '0';
                    cont5 <= "0000";
                end if;
```

©Editorial Digital Tecnológico de Monterrey

```
                    if sw3 = '1' and cont8 < 8 then
                        enable <= '1';
                        cont8<=cont8 + 1;
                    elsif (sw1 = '0' and sw2 = '0' and sw3 = '0') or cont8 = 8 then
                        enable <= '0';
                        cont8 <= "0000";
                    end if;

                end if;
            end if;
        end process;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

8.12 Horno de microondas

Se le solicita diseñar el circuito para controlar un horno de microondas. Al circuito se le conectarán los siguientes periféricos:

Entradas:

Botones:

Start: Inicia la operación de las microondas, siempre y cuando la puerta esté cerrada.
Stop: Detiene la operación de las microondas sin borrar el tiempo transcurrido.
Reset: Pone en ceros el tiempo transcurrido.
Minuto: Agrega 60 segundos al tiempo de calentamiento. Asegure que solo se agregan 60 por cada vez que se pulsa, aunque se quede oprimido mucho tiempo. No detiene la operación.
Tiempo: Al pulsarlo se registra el tiempo que se haya colocado en los *switches*.

Switches:

S: Ocho *switches* para colocar el tiempo de operación en segundos.

Sensor: ***Puerta:*** En 1 indica que la puerta está abierta, 0 de otro modo.

Señal de reloj: ***Clk:*** Señal de reloj que tiene una frecuencia de 1Hz.

©Editorial Digital Tecnológico de Monterrey

Salidas:

Micro: En 1 acciona el dispositivo que genera las microondas. Termina la operación, es decir, se pone en 0 cuando se oprime *Stop* o cuando se abre la puerta o cuando el tiempo transcurrido llega a cero. El circuito debe generar esta señal.

TT: Es el tiempo transcurrido, se muestra en 8 leds, en binaro en orden descendente.

Para este problema el circuito esquemático basado en registros, latches, flip flops y compuertas es el siguiente:

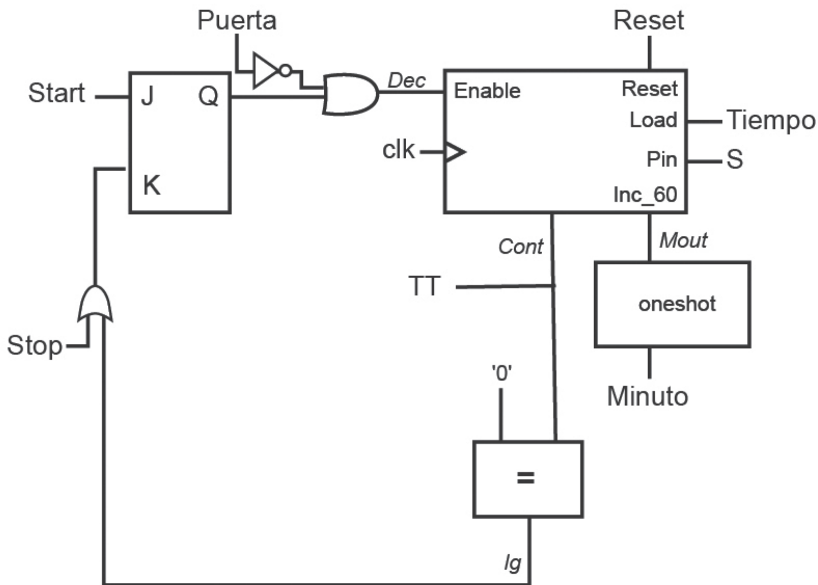


Figura 8.20 Diseño esquemático del controlador de microondas

©Editorial Digital Tecnológico de Monterrey

En VHDL la arquitectura de este circuito es la siguiente:

```
entity microondas is
    Port(Start: in  STD_LOGIC;
         Stop: in  STD_LOGIC;
         Reset: in  STD_LOGIC;
         Minuto: in  STD_LOGIC;
         Tiempo: in  STD_LOGIC;
Puerta: in  STD_LOGIC;
Clk: in  STD_LOGIC;
           S: in  STD_LOGIC_vector(7 downto 0);
Micro: out  STD_LOGIC
           TT: out  STD_LOGIC);
end display;

architecture comportamiento of display is

     component oneshot is
          Port(clk: in  STD_LOGIC;
               D: in  STD_LOGIC;
               Q: out  STD_LOGIC);
     end component;

     signal Q, K , DEC, Mout, Ig:  STD_LOGIC;
     signal CONT: STD_LOGIC_VECTOR(7 downto 0);
begin

     k <= Stop or Ig;
     Q <= Start and not Q or (not k and Q);
     Dec <= not Puerta and Q;
     Ig <= ‘1’ when cont = “00000000” else ‘0’;
     Os: oneshot port map(clk, Minuto, Mout);
```

©Editorial Digital Tecnológico de Monterrey

```
process(clk, reset, tiempo,Mout,Dec)
begin
     if reset = '1' then cont <= "00000000";
elsif tiempo = '1' then cont <= S;
elsif clk = '1' and clk'event then
     if Mout = '1' then cont <= cont + 60;
     elsif Dec = '1' then cont <= cont - 1;
     else null;
end if;
          else null;
     end if;
     end process;
Micro <= Dec;
TT <= cont;
end comportamiento;
```

©Editorial Digital Tecnológico de Monterrey

8.13 Captura de botones

Se le solicita diseñar el circuito que controlará la sección de un sistema digital que se encarga de la captura de un número BCD. Para escribir los números se cuenta con diez botones (*push buttons*). Un botón simboliza el 0, otro el 1, otro el 2 y así sucesivamente. Cada vez que se oprima un botón, el dígito BCD correspondiente se almacenará en los cuatro *bits* menos significativos de un registro, el resto de los dígitos BCD previamente almacenados se recorrerán hacia la izquierda. La salida del circuito es la salida del registro. El registro es de 80 *bits*, así que después de capturar 20 dígitos se pierde el primero que se haya escrito. Considere también a la entrada un botón de *reset* para borrar el contenido del registro.

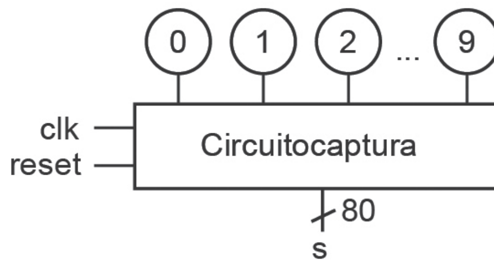


Figura 8.21 Botones

El puerto es el siguiente:

```
entity Circuitocaptura is
     port(botones: in std_logic_vector (0 to 9);
              clk, reset: in std_logic;
              s: out std_logic_vector(79 downto 0));
end Circuitocaptura;
```

©Editorial Digital Tecnológico de Monterrey

El diseño esquemático, con algunos puntos suspensivos para no saturar el dibujo, es el siguiente:

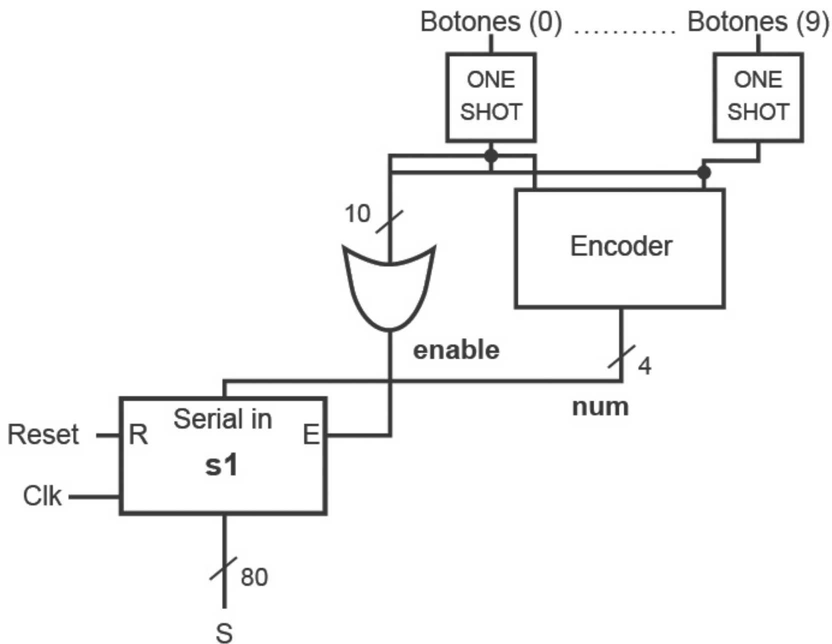


Figura 8.22 Diseño esquemático del circuito para capturar números BCD seleccionados por botones

La codificación de la arquitectura, con algunos puntos suspensivos para evitar incluir todo el código que sea semejante, es la siguiente:

©Editorial Digital Tecnológico de Monterrey

```
architecture behavioral of Circuitocaptura is
signal enable, b0,b1,b2,b3,b4,b5,b6,b7,b8,b9:std_logic;
signal s1: std_logic_vector(79 downto 0);
signal num: std_logic_vector(3 downto 0);
 begin

     botones00: oneshot port map(clk,botones(0),b0);
                    .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
     botones09: oneshot port map(clk,botones(9),b9);
    registro_serial: process(clk,reset,s1)
                begin
                   if reset = '1' then
                         s1 <= (others => '0'); --con esta opción se inicializa a ceros
                  elsif clk = '1' and clk'event and (enable = '1') then
                        s1 <= s1(75 downto 0) & num;
                  end if;
           end process;
    num <=  "0000" when (b0 = '1') else  --éste es el encoder
                   "0001" when (b1 = '1') else
                           .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
                  "1001" when (b9 = '1');
   s <= s1;
   enable <= '1' when (b0 = '1') or (b1 = '1') or . . . . . . . .( b9 = '1') else '0';
end behavioral;
```

Responde la siguiente actividad.

Revisa si es posible reducir la cantidad de one shots a uno solo.

©Editorial Digital Tecnológico de Monterrey

8.14 Marcador de futbol

Diseñe en VHDL un circuito que genere las señales necesarias para que un tablero registre el marcador de un juego de fútbol. Los equipos se distinguirán como casa y visitante.

Las entradas al circuito son las señales generadas por tres botones:

- golc: que se usa para registrar un gol marcado por el equipo de casa.
- golv: que se usa para registrar un gol marcado por el equipo visitante.
- *reset:* borra a ceros todas las salidas del marcador.

Las salidas del circuito son:

- dec_cout: 4 *bits* que representa las decenas, en BCD, de los goles del equipo de casa.
- unid_cout: 4 *bits* que representa las unidades, en BCD, de los del equipo de casa.
- dec_vout: 4 *bits* que representa las decenas, en BCD, de los goles del equipo visitante.
- unid_vout: 4 *bits* que representa las unidades, en BCD, de los goles del equipo visitante.

A continuación, se muestra la definición del puerto:

©Editorial Digital Tecnológico de Monterrey

```
entity tablerofutbol is
    Port ( gol_c : in std_logic;  gol_v : in std_logic; reset : in
std_logic;clk
           dec_cout, unid_cout : out std_logic_vector(3 downto 0);
           dec_vout, unid_vout : out std_logic_vector(3 downto 0));
end tablerofutbol;

architecture Behavioral of tablerofutbol is

begin
```

©Editorial Digital Tecnológico de Monterrey

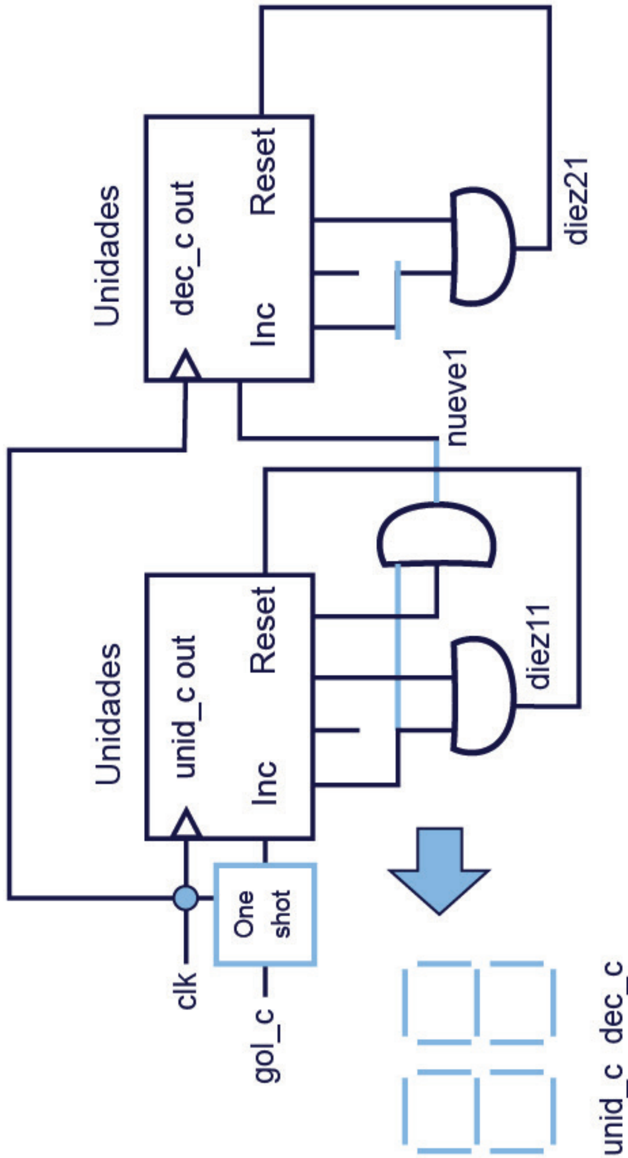


Figura 8.23 Diseño esquemático del circuito para tablero de futbol (1)

©Editorial Digital Tecnológico de Monterrey

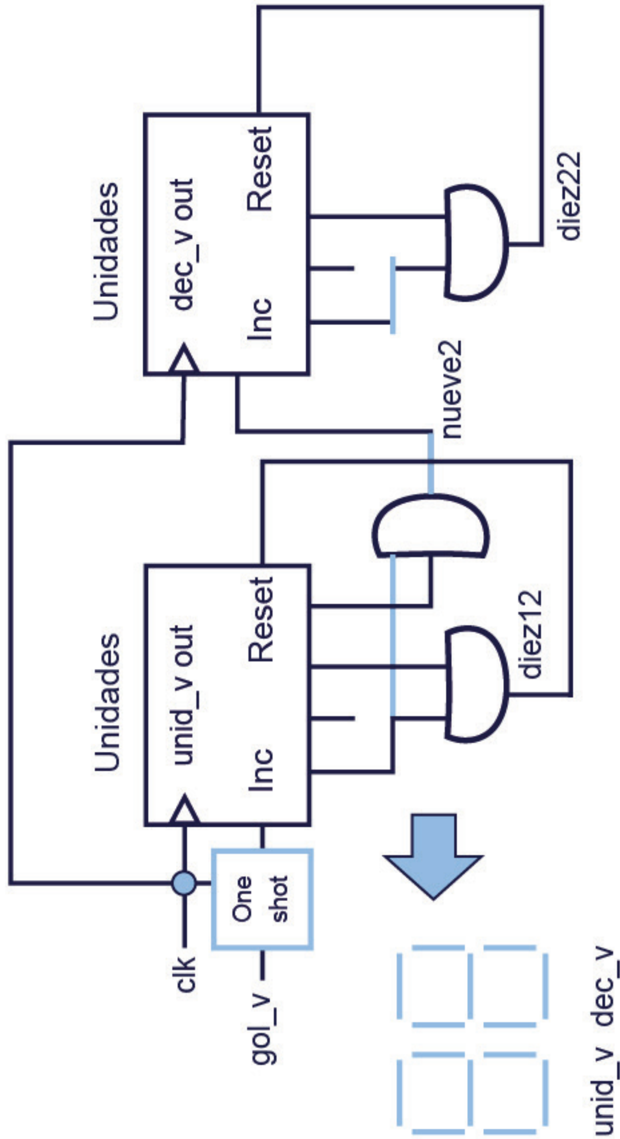


Figura 8.24 Diseño esquemático del circuito para tablero de futbol (2)

©Editorial Digital Tecnológico de Monterrey

Código:

```
entity futbol is
    Port( unid_c, dec_c, unid_v, dec_v: out std_logic_vector(6
downto 0);
             gol_ c,gol_v: in std_logic;
             clk: in std_logic);
end futbol;

architecture Behavioral of futbol is
component oneshot is
   port( clk, din:in std_logic; qout:out std_logic);
end component;
```

©Editorial Digital Tecnológico de Monterrey

```
component contador
Port(clk, inc, reset : in std_logic;
     s: out std_logic_vector(3 downto 0));
end component;
component conviertedisplay is
   port(bin:in std_logic_vector(3 downto 0); bout: out std_logicstd_logic_vector(6 downto 0));
end component;

signal diez11, diez12, diez21, diez22, nueve1, nueve2: std_logic;
signal unid_cout,dec_cout, unid_vout,dec_vout:std_logic_vector(3 downto 0);
begin
os1: oneshot port map (clk, gol_c, gol_cout);
os2: oneshot port map (clk, gol_v, gol_vout);

e1:  contador port map (clk, gol_cout, diez11,  unid_cout);
e2:  contador port map (clk, nueve1, diez21, dec_cout);
e1:  contador port map (clk, gol_vout, diez12,  unid_vout);
e2:  contador port map (clk, nueve2, diez22, dec_vout);

diez11 <= unid_cout (3) and unid_cout (1);
diez12 <= unid_vout (3) and unid_vout (1);
diez21 <= dec_cout (3) and dec_cout (1);
diez22 <= dec_vout (3) and dec_vout (1);

unid_c: conviertedisplay port map(unid_cout, unid_c);
unid_v: conviertedisplay port map(unid_vout, unid_c);
dec_c: conviertedisplay port map(dec_cout, dec_c);
dec_v: conviertedisplay port map(dec_vout, dec_v);

end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

La descripción del one shot es la siguiente:

```
entity one_shot is
    Port(Din, clk: in  STD_LOGIC;
         Qout: out  STD_LOGIC);
end one_shot;

architecture Behavioral of one_shot is
signal Q1, Q2, Q3: std_logic;
begin
process(clk)
begin
 if clk = '1' and clk'event then
            Q1 <= Din;
      Q2 <= Q1;
      Q3 <= Q2;
 end if;
end process;
Qout <= Q1 and Q2 and not Q3;
end Behavioral;

 entity contador is
    Port(clk, inc, reset : in std_logic;
              s: out std_logic_vector(3 downto 0));
end contador;

architecture Behavioral of contador is
signal saux: std_logic_vector(3 downto 0): = ”0000”;
                           --la inicialización es válida tanto
                           --para la simulación como para la síntesis,
                           --pero si en el sistema de desarrollo que esté
                           --usando no fuera válida entonces se requeriría
                           --un pulso de reset para que se asegure
                           --que el contador esté en ceros
```

©Editorial Digital Tecnológico de Monterrey

```
begin
process(clk, inc, reset)
begin
if reset='1' then s<="0000";
   elsif clk='1'and clk'event and (inc='1') then saux<= saux+1;
end if;
end process;
s<=saux;
end Behavioral;
entity conviertedisplay is
   port(binin:in std_logic; bcdout:out std_logic);
end component;

architecture comportamiento of conviertedisplay is
begin
bcdout<=   "1111110" when binin="0000"
          else  "0110000" when binin="0001"
          else  "1101101" when binin="0011"
          else  "1111001" when binin="0011"
          else  "0110011" when binin="0100"
          else  "1011001" when binin="0101"
          else  "1011111" when binin="0110"
          else  "1110000" when binin="0111"
          else  "1111111" when binin="1000"
          else  "1111011" when binin="1001"
          else null;
end comportamiento;
```

©Editorial Digital Tecnológico de Monterrey

8.15 Calculadora

El siguiente es un problema que ejemplifica una calculadora. Suponga que dispone de pocos periféricos (ocho *switches*, botones, *displays*). El funcionamiento del circuito sería el siguiente. Da clic en los botones de la calculadora.

a) Formar un número de uno a tres dígitos, para esto se debe elegir un dígito de cero a siete utilizando ocho *switches* (aceptando un dígito octal, pero si dispone de 10 *switches* para implementar este circuito, puede formar un número decimal). Cuando se oprima un botón que se use como <enter>, el dígito se desplegará en un *display* de siete segmentos (posición cero).

Puede elegir un nuevo dígito con los *switches* y oprimir <enter> nuevamente. El dígito que ya estaba iluminado se pasa a la posición 1 y el nuevo dígito va a la posición 0. Así puede elegir un tercer dígito, luego oprimir <enter>, y los dos dígitos ya iluminados avanzan una posición mientras que el nuevo dígito se ilumina en la posición 0. Si se oprime de nuevo <enter>, el primero de los dígitos desaparecerá (debido al tamaño del registro en el que está almacenando los datos, lo puede extender si desea manejar números mayores).

b) Luego se espera que se oprima un botón de suma <+> o de resta <−>.

c) Enseguida se espera que se forme un nuevo número de uno a tres dígitos.

d) A continuación, al oprimir igual < = >, se iluminará en los *displays* el resultado de la suma o de la resta entre ambos números, pero solo en tres dígitos (para simplificar). La suma

©Editorial Digital Tecnológico de Monterrey

es decimal. El número se forma con dígitos menores o iguales a 8 porque solo dispone de ocho *switches*, pero la idea es que sea un número decimal, no octal.

e) Es posible proseguir, regresando al paso a, es decir, capturando un nuevo número. El lector definirá el resto de la funcionalidad, ya que hay diferentes opciones, así como también definirá su propia calculadora.

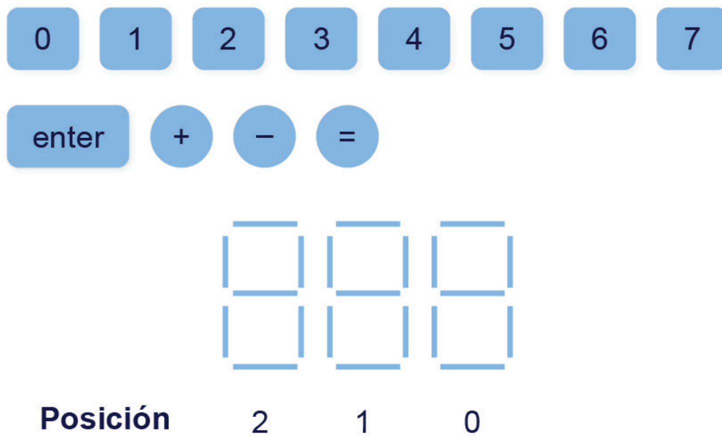


Figura 8.25 Diagrama de los elementos de los que dispone su calculadora

Para este problema se requiere comprender la manera de convertir una selección de un *switch* (que bien podría ser un pulso de botón) en un número. Asimismo, entenderá la diferencia entre un número binario a uno BCD (la mayoría de los humanos funcionamos con el sistema decimal, debido probablemente a que tenemos diez dedos), así que la comunicación con un sistema digital debe ocurrir con algo que facilite la notación en dígitos (decimales).

©Editorial Digital Tecnológico de Monterrey

Es posible plantear una solución como la siguiente:

1) Ingresar los dígitos (pasando los *switches* por un *encoder*) a un registro A, recorriendo los que ya fueron capturados. La carga al registro A se habilita con un pulso *one shot* de la tecla enter. Se necesita el one shot porque si se habilita la carga por más de un período se harían más corrimientos de los necesarios.

2) Cuando se oprima suma o resta, se transfiere el contenido del registro A al registro B y se pone a ceros A.

3) Se captura un nuevo número, dígito a dígito, con la lógica descrita en 1.

4) Con la tecla de igual se carga el resultado de la suma o de la resta, según sea el caso. (Suma o resta quedaron grabadas en un *latch*).

5) Ir al paso 2, para seguir sumando o restando un tercer (o enésimo) número.

En el problema no se planteó un botón de *reset* para iniciar una nueva operación, por eso no se encuentra en el diseño.

©Editorial Digital Tecnológico de Monterrey

El esquemático que corresponde a esta solución es el siguiente:

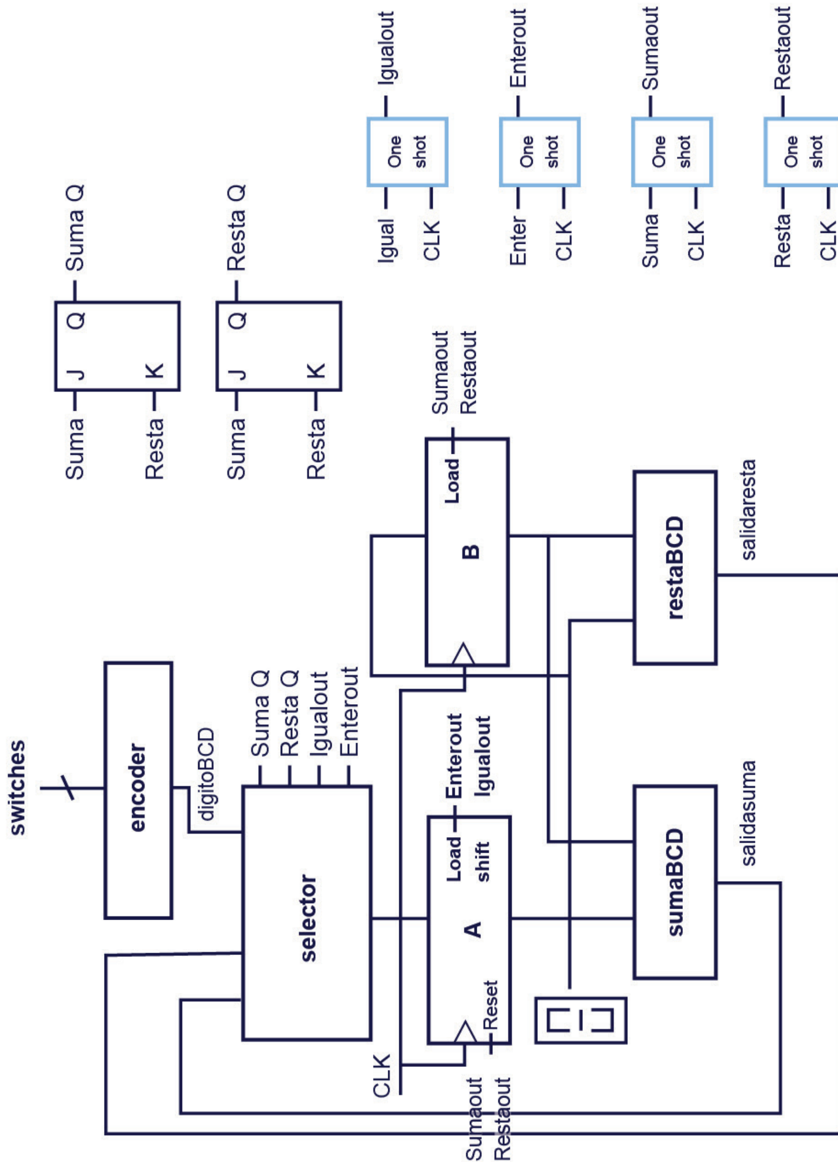


Figura 8.26 Diseño esquemático del circuito de la calculadora

©Editorial Digital Tecnológico de Monterrey

A este diseño le corresponde la siguiente codificación en VHDL. Como ejercicio, identifique la sección de código que corresponde a cada parte del diseño esquemático.

```
entity calculadora is
    Port(clk, enter, suma, resta, igual: in std_logic;
                   switches: in STD_LOGIC_VECTOR (7 downto 0);
           display2, display1, display0: out STD_LOGIC_VECTOR (7 downto 0));
end calculadora;

architecture Behavioral of calculadora is

component oneshot is
   port(clk, din:in std_logic; qout: out std_logic);
end component;
component sumabcd is
   port(operando1, operando2: in std_logic_vector(11 downto 0);
           resultado: outstd_logic_vector(11 downto 0));
end component;
component restabcd is
   port(operando1, operando2: in std_logic_vector(11 downto 0);
           resultado: outstd_logic_vector(11 downto 0));
end component;
signal digitoBCD: std_logic_vector(3 downto 0);
signal igualout, enterout, sumaout, restaout, sumaQ, restaQ: std_logic;
signal resultadosuma, resultadoresta, A, B: STD_LOGIC_VECTOR (11 downto 0);
begin

--one shots
 boton_igual: oneshot(clk,igual,igualout);
 boton_enter: oneshot(clk,enter,enterout);
 boton_suma: oneshot(clk,suma,sumaout);
 boton_resta: oneshot(clk,resta,restaout);
```

©Editorial Digital Tecnológico de Monterrey

```
--botones de suma y resta
sumaQ <= (suma and not sumaQ) or (not resta and sumaQ);
 restaQ <= (resta and not restaQ) or (not suma and restaQ);

--decodificador
 digitoBCD <=        "0000" when switch(7) = '0'
                          else "0001" when switch(6) = '1'
                     else "0010" when switch(5) = '1'
                     else "0011" when switch(4) = '1'
                     else "0100" when switch(3) = '1'
                     else "0101" when switch(2) = '1'
                     else "0110" when switch(1) = '1'
                     else "0111" when switch(0) = '1'
                     else "1111";
```

©Editorial Digital Tecnológico de Monterrey

```
RegistroA: process(clk, enterout, igualout, sumaout, restaout)
      begin
            if (clk='1') and clk'event  then
                  if (sumaout = '1') or (restaout = '1') then A <= "000000000000";
                  elsif ((enterout = '1') or (igualout = '1'))
                        then
                        if enterout = '1' then A <= A(7 downto 0)& digitoBCD;
                              elsif igualout = '1' then
                                    if sumaQ = '1' then A <= salidasuma;
                                    elsif restaQ = '1' then A <= salidaresta;
                                           else null;
                                    end if;
                                    else null;
                                    end if;
                               else null;
                              end if;
                        end process;

RegistroB: process (clk)
            begin
            if clk = '1' and clk'event and ((sumaout = '1') or (restaout = '1'))
                  then B <= A;
                  else null;
            end if;
              end process;
 suma: sumabcd port map (A, B, salidasuma);
 resta: restabcd port map (A, B, salidaresta);
Display2: conviertedisplay port map(A(11 downto 8, display2);
Display1: conviertedisplay port map(A(7 downto 4, display1);
Display0: conviertedisplay port map(A(3 downto 0, display0);
end Behavioral;
```

En la solución no se presenta el manejo del signo de la respuesta para el caso en que el resultado de la resta sea negativo.

Los problemas que se resolvieron a lo largo de este capítulo muestran cómo la interconexión de *flip flops*, *latches*, *one shots*, registros y lógica combinacional es un arte. En el siguiente capítulo se mostrará una herramienta de diseño de circuitos secuenciales bastante poderosa, con la cual se facilitará muchísimo el diseño de este tipo de circuitos.

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 8

Los siguientes problemas están propuestos para resolverse en el laboratorio, por lo que no se presentarán las respuestas, sino que tendrá que probar sus diseños en la práctica.

1. Diseñar un contador que cuente de 00 a 99.

Se deberán utilizar dos contadores sincrónicos de 4 *bits* que cuenten hasta diez, cada uno funcionando de la siguiente manera:

- Todos los cambios de estado ocurren en la transición positiva del reloj.
- Cuando CLR=0 el contador es borrado a ceros, sin importar los valores de las otras entradas. (Cuando se conecten ambos contadores, con esta señal (CLR) se borran a ceros ambos contadores).
- Si CLR = 1 cada contador se incrementa de 1 en 1 hasta llegar a 9, cuando cada contador llega a 9 prende un *carry out* (CO) y a continuación se reinicia en 0.

El puerto de estos contadores (ya de ambos conectados) es:

```
entity CONTADORES is
     Port(CLR: in std_logic;
          CLK: in std_logic;
          CO1, CO0: out std_logic;
          A0, B0: out std_logic_vector(3 downto 0));
end CONTADORES;
```

©Editorial Digital Tecnológico de Monterrey

2. Se le solicita diseñar el circuito que controle al dispositivo que emite calor en un horno. Se pretende hacer un diseño similar al de un horno de microondas. La temperatura se proporciona con una perilla que no consideraremos en el diseño del circuito.

El horno cuenta con 10 switches para colocar el tiempo de cocimiento, suponga que el tiempo se proporcionará en binario.

Llame C al registro que lleve la cuenta regresiva del tiempo de cocimiento. El valor de C se observará en leds en el horno, en binario. Suponga que, al encender eléctricamente al horno, C inicia en ceros.

Se contaría con los siguientes botones:

Time: Carga en C el número que se haya colocado en los switches, este botón solo tiene acción si el horno no está en operación.

Start: inicia la operación del horno, así como la cuenta regresiva de tiempo.

Stop: detiene la operación del horno sin borrar la cuenta de tiempo.

Diez: incrementa en 10 seg. al registro de tiempo (registro C), este botón solo tiene acción si el horno no está en operación.

A la salida que debe generar su circuito se le llamará:

Horno: se debe generar un 1 en horno luego de que se oprima el botón de *start*. Horno debe permanecer en 1 el número de segundos que tenga el contador C o hasta que se oprima *stop*. C deberá de decrementar segundo a segundo hasta llegar a cero o hasta se oprima *stop*.

Suponga que cuenta con un reloj externo con períodos de 1 seg.

©Editorial Digital Tecnológico de Monterrey

a) Diseñe un circuito esquemático basado en registro(s), *flip flops* y otros componentes, sin utilizar máquina de estados. El registro puede tener la funcionalidad que usted requiera.

b) Describa en VHDL el circuito completo, que incluya las señales que vienen de los botones, los *switches*, C y la salida horno.

3. Una empresa de publicidad quiere saber cuántos automóviles pasan por cierto crucero durante un período de tiempo, para efecto de colocación de anuncios. Se le solicita diseñar un circuito al cual ingresará la señal de un sensor que alimentará un 1 al circuito cada vez que pase un vehículo. El tiempo que dura el pulso que envía el sensor es variable, depende de la longitud del vehículo. La calle es de un carril y de un solo sentido.

Como parte del circuito se deben proporcionar doce *switches* para indicar, en binario, la cantidad de segundos durante la cual se realizará la cuenta de automóviles.

Debe haber un botón de *start* que inicie tanto la cuenta de automóviles como de tiempo. Cuando la cantidad de segundos termine, también termina el conteo de vehículos. Hay un botón de *reset* para borrar el conteo.

La salida del circuito será en binario de 12 *bits* para la cuenta de carros.

a) Diseñe un circuito esquemático

b) Diseñe en **VHDL** la arquitectura del circuito. Utilice el siguiente puerto:

©Editorial Digital Tecnológico de Monterrey

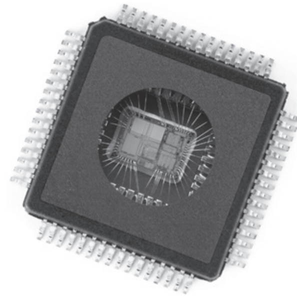
```
entity cuentacarros is
     Port(switches: in std_logic_vector (11 downto 0);
          clk, Start, sensor, reset: in std_logic;
          S: out std_logico_vector (11 downto 0));
end cuentacarros;
```

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 8

El diseño de circuitos secuenciales basados en registros, señales de reloj y *one shots* es un arte, ya que no hay una técnica que lo respalde. Por este motivo, en este capítulo se mostraron ejemplos con aplicaciones muy variadas, cubriendo diferentes aspectos del diseño basado en registros.

En el próximo capítulo se estudiará una técnica para diseño de circuitos secuenciales que representa una alternativa al diseño basado en registros.



©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 9. Diseño de autómatas

Diseño de autómatas

Diseño de contadores

- Contador de dos bits
- Contador de tres bits
- Diseño de contador con entrada externa

Diseño de máquinas de estados

- Bit de paridad impar
- Diseño de una máquina de estados
- Ejemplo de diseño de una máquina de estados
- Opciones de notación en el diseño de diagramas de estados
- Alternativas de codificación de una Máquina de Moore
- Ejemplos de alternativas de codificación de una Máquina de Moore

En el capítulo anterior se construyeron circuitos secuenciales uniendo "piezas" como en un rompecabezas. En este capítulo se utiliza una herramienta de construcción: secuencias basadas en estados. Los estados representan las salidas esperadas por el circuito en alguna etapa de la secuencia. Se llega a un estado "j" cuando se presentan un conjunto de salidas (estado "i") y un conjunto de entradas Xij. Los estados de un circuito se diseñan en **diagramas de estados**. Un diagrama de estados resume el comportamiento automático de un circuito secuencial. Un circuito con un comportamiento automático recibe el nombre de **autómata** o **máquina de estados**. Un circuito de este tipo realiza una serie de pasos repetitivos, siempre respondiendo de la misma manera ante una secuencia de estímulos en las entradas.

©Editorial Digital Tecnológico de Monterrey

El diseño autómatas por diagramas de estados constituye la herramienta más poderosa en el desarrollo de circuitos secuenciales. El diseñar un circuito secuencial interconectando registros es un arte; diseñar un circuito secuencial a través de un diagrama de estados requiere el dominio de una técnica.

Al definir un autómata, la lógica que se requiere para habilitar las funciones que realizan los registros como son las cargas, o las sumas, o los corrimientos, se diseña siguiendo un método.

Los estados se construyen con *flip flops* independientes, o bien visualizándolos como registros. La ventaja de este método consiste en que, en lugar de proponer circuitos para dirigir la secuencia de operaciones, se construye la lógica para hacer que el circuito "cambie de estado", con la ventaja de que los pasos que se siguen para el diseño de los circuitos de cambio de estado siempre son los mismos.

Los autómatas más simples son los contadores. A continuación, se resuelven dos problemas en los que se revisa el método a seguir para el diseño de contadores.

©Editorial Digital Tecnológico de Monterrey

9.1 Diseño de contadores

Se llamará estado del circuito a la combinación de valores que presenten las salidas (ya sea las salidas del puerto, es decir, las salidas finales del circuito, o bien las de algún contador auxiliar). La restricción que tiene el diseño de un contador con el método que se estudiará en esta sección es que no existan estados repetidos.

9.1.1 Contador de dos bits

Se describe el método para el diseño de un contador, paso a paso, con el siguiente ejemplo:

Diseñe un contador de dos *bits*, que cuente de 00 a 11 y que luego regrese a 00, utilizando *flip flops* D y una señal de reloj con la frecuencia que se desea para realizar el conteo.

Paso 1

Dibujar utilizando círculos y flechas la secuencia de salidas esperadas por el circuito. En cada círculo se incluye un estado del circuito.

Los diferentes estados de la secuencia se conectan por flechas. El cambio de estado ocurre con cada transición de la señal de reloj.

Como en este problema se diseña un contador de dos *bits*, los estados requieren dos salidas Z1 y Z0:

Figura 9.1 Un estado en un diagrama de estados

©Editorial Digital Tecnológico de Monterrey

A continuación, se traza la secuencia esperada de las salidas planteadas en los círculos por medio de flechas.

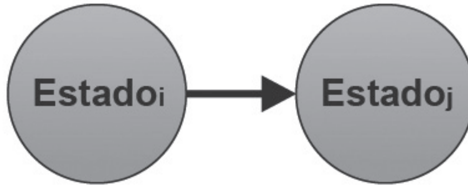


Figura 9.2 Una transición en un diagrama de estados

El estado de inicio debe distinguirse. Una posibilidad es indicar con una flecha, alguna señal que fuerce al circuito a dirigirse a este estado.

El diagrama de estados resultante para este contador es:

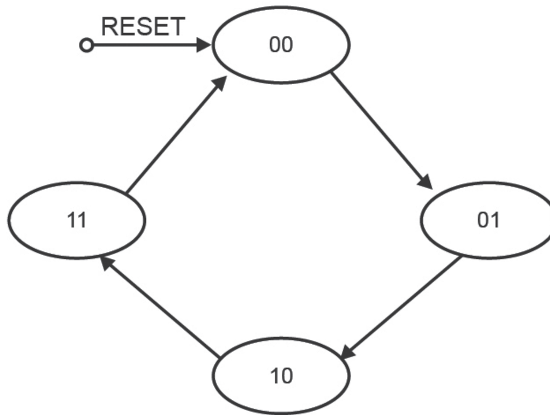


Figura 9.3 Diagrama de estados con reset

©Editorial Digital Tecnológico de Monterrey

Paso 2

Asociar la salida de un *flip flop* a cada *bit* de la salida del circuito. Por ejemplo, si la salida está dada por las señales Z1, Z0, o bien salida (1) y salida (0), estas señales pueden ser asociadas a Q1, Q0, las salidas de dos *flip flops* de cualquier tipo.

Paso 3

Construir una tabla de verdad en la que se indique para cada estado cuál es su siguiente estado (básese en la secuencia del autómata, de 00 a 01).

Q1	Q0	Q1+	Q0+
0	0	50	1
0	1	1	0
1	0	1	1
1	1	0	0

Tabla 9.1

Paso 4

Elegir con qué tipo de *flip flop* se construirá el circuito. Para este efecto solo se diseñarán autómatas con *flip flops* D, debido a los siguientes dos motivos: a) por ser el tipo de *flip flop* contenido en los FPGAs (y así la implementación del circuito es más eficiente) y, b) por representar la opción más sencilla de diseño. Se recordará que un *flip flop* D almacena (durante la transición del reloj) lo que se presente en la entrada, así que se conectará en la entrada el circuito que se requiera para que cambie al siguiente estado, de acuerdo a la tabla de verdad del problema.

©Editorial Digital Tecnológico de Monterrey

En un *flip flop* D, para provocar que **Q** cambie a **Q+** se requiere que la entrada D tenga el valor que corresponda (recuerde que la salida sigue a la entrada en el ffD):

Q	Q+	D=Q+
0	0	0
0	1	1
1	0	0
1	1	1

Por lo que se observa, basta conectar Q+ para que el circuito siga la secuencia deseada.

Para este problema, la secuencia es:

Q1	Q0	D1 = Q1+	D0 = Q0+
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

La otra alternativa es que el conjunto de *flip flops* se visualice como un registro, al que llamaremos State, que internamente está formado por *flip flops*.

Paso 5

En caso de haber elegido *flip flops* independientes, se debe obtener la función lógica que hay que conectar a la entrada de cada *flip flop* para obtener el siguiente estado. Puede describirse en su forma canónica o simplificada.

©Editorial Digital Tecnológico de Monterrey

En este ejemplo:

$$D_1 = Q_1+ = Q_1' \; Q_0 + Q_1 \; Q_0' = Q_1 \oplus Q_0$$
$$D_0 = Q_0+ = Q_1' \; Q_0' + Q_1 \; Q_0' = Q_0'$$

Si se decide visualizar el conjunto de *flip flops* como un registro, la lógica para obtener el siguiente estado puede quedarse pendiente y detallarse en VHDL como se mostrará en el paso 7.

Paso 6

Circuito resultante

Con *flip flops*:

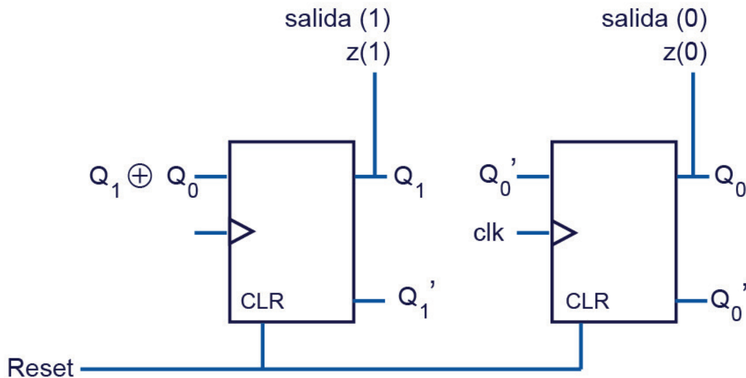


Figura 9.4 Contador de dos *bits* con *flip flops* D

©Editorial Digital Tecnológico de Monterrey

Con registro:

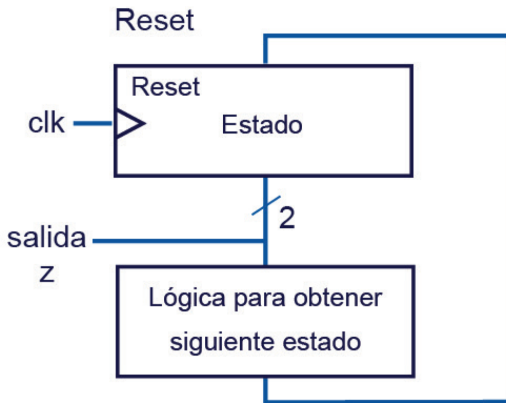


Figura 9.5 Contador de dos *bits* con *flip flops* D

Paso 7

Para la construcción del circuito es posible proceder a su cableo en protoboard o bien a su descripción en VHDL.

A continuación, la descripción de la arquitectura en VHDL.

Con *flip flops*:

```
Entity contador is
Port (clk, Reset:in std_logic;
      Z:std_logic_vector(1 downto 0));
End contador;

Architecture flipflops of contador is
Signal Q1, Q0:std_logic;

Process(clk)
Begin
If  Reset='1' then  Q1<='0';
                 Q0<='0';
elsif clk='1' and clk'event then
Q1<=Q1 xor Q0;
Q0<= not Q0;
Else null;
End if;
End process;
Z<=Q1&Q0;
End flip flops;
```

©Editorial Digital Tecnológico de Monterrey

Con registro, para lo cual no se requiere simplificar las funciones, sino que, a partir del comportamiento de la secuencia, extraída desde el diseño de los círculos y las flechas, es posible modelar el circuito como se muestra a continuación:

```
Entity contador is
Port (clk, Reset:in std_logic;
     Z:std_logic_vector(1 downto 0));
End contador;

Architecture flipflops of contador is
Signal Estado:std_logic_vector(1 downto 0));

Process(clk)
Begin
If  Reset=’1’ then  Estado <=”00”;
elsif clk=’1’ and clk’event then
Case Estado is
When “00” => Estado <=”01”;  --este case representa la lógica combinacional necesaria
When “01” => Estado <=”10”; --para calcular el siguiente estado
When “10” => Estado <=”11”;
When “11” => Estado <=”00”;
When others  => null;
End case;
Else null;
End if;
End process;
Z<=Estado;
End registro;
```

©Editorial Digital Tecnológico de Monterrey

Por supuesto, existe otra forma simple de definir esta secuencia, que es la que se utilizó en el capítulo 6. A continuación, se transcribe el esquemático que se utilizó:

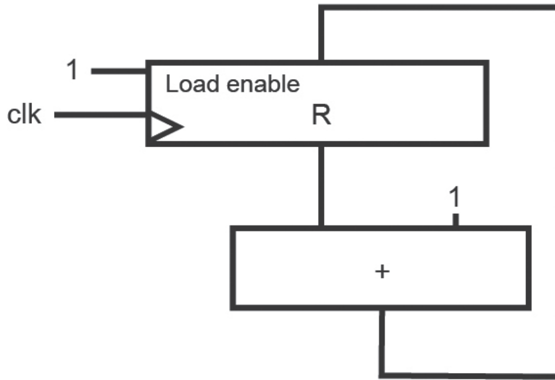


Figura 9.6 Diagrama esquemático de registro con carga paralela conectado a sumador

```
Entity contador is
Port (clk, Reset:in std_logic;
      Z:std_logic_vector(1 downto 0));
End contador;

Architecture registro of contador is
Signal Estado:std_logic_vector(1 downto 0));
```

```
Process(clk)
Begin
If  Reset='1' then  Estado <="00";
elsif clk='1' and clk'event then
Estado<=Estado+1;
Else null;
End if;
End Process;
Z<=Estado;
End registro;
```

©Editorial Digital Tecnológico de Monterrey

Como el registro estado es de dos *bits*, al llegar a la cuenta máxima de 3 y sumar 1 en vez de llegar a 4 (que requiere 3 *bits*, 100) regresará a 0 (ya que el *bit* más significativo de 4 no se registra).

9.1.2 Contador de tres bits

A continuación, se presenta otro problema que consiste en el diseño de un contador de tres *bits* con una cuenta no consecutiva. Para llegar al circuito se seguirán los pasos propuestos en el problema anterior.

Diseñe un contador que genere la siguiente secuencia: 000, 011, 100, 110, 000, 011… Suponga que cuenta con una entrada externa de reloj con la frecuencia que desea para generar las salidas.

El diseño gráfico del autómata es el siguiente:

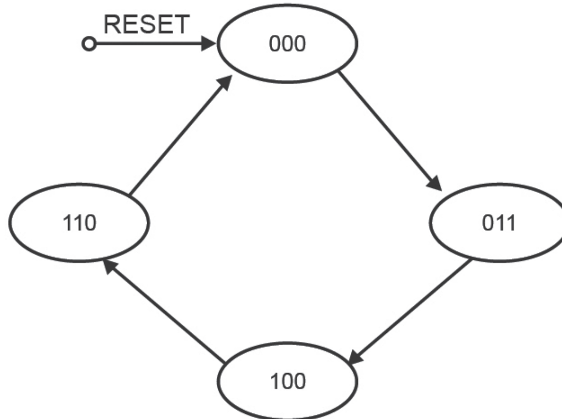


Figura 9.7 Contador de tres *bits* con cuatro combinaciones de salida

Se tienen tres salidas: Z2, Z1 y Z0 que se pueden conseguir con las salidas de tres *flip flops* D, Q2 Q1 Q0. La secuencia esperada para las salidas es:

©Editorial Digital Tecnológico de Monterrey

Q2	Q1	Q0	Q2+	Q1+	Q0+
0	0	0	0	1	1
0	0	1	X	X	X
0	1	0	X	X	X
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	X	X	X
1	1	0	0	0	0
1	1	1	X	X	X

Tabla 9.2

Las salidas X son condiciones que no importan, ya que el circuito no pasa por esos estados.

Para construir el circuito con *flip flops*, se extraen las funciones lógicas de la tabla de verdad.

$$D_2 = Q_2+ = Q_1 Q_0 + Q_2 Q_1'$$
$$D_1 = Q_1+ = Q_1'$$
$$D_0 = Q_0+ = Q_2' Q_1'$$

©Editorial Digital Tecnológico de Monterrey

El circuito resultante es:

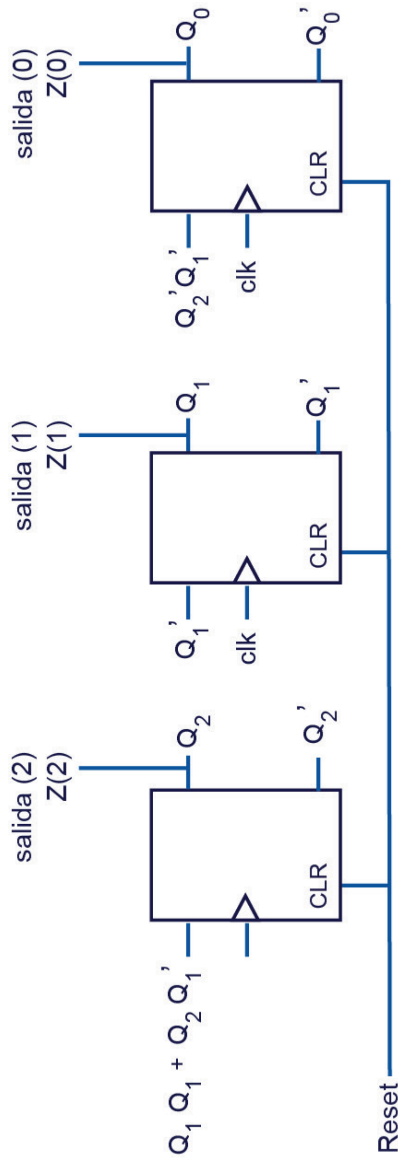


Figura 9.8 Circuito de contador de tres *bits* diseñado con *flip flops* D

©Editorial Digital Tecnológico de Monterrey

Con la siguiente descripción en VHDL:

```
Entity contador is
Port (clk, Reset:in std_logic;
      Z:std_logic_vector(2 downto 0));
End contador;

Architecture flipflops of contador is
Signal Q1, Q0:std_logic;

Process(clk)
Begin
If  Reset='1' then  Q1<='0';
                    Q0<='0';
elsif clk='1' and clk'event then
Q2<= Q1 and Q0   or  (Q2 and not Q1);
Q1<=  not Q1;
Q0<= not Q2 and not Q1;
Else null;
End if;
End process;
Z<=Q2&Q1&Q0;
End flip flops;
```

Con registro:

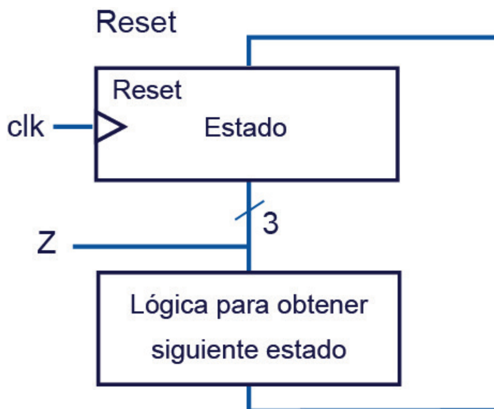


Figura 9.9 Diseño esquemático general del contador basado en registro y combinaciones de salida

©Editorial Digital Tecnológico de Monterrey

```
Entity contador is
Port (clk, Reset:in std_logic;
      Z:std_logic_vector(1 downto 0));
End contador;

Architecture registro of contador is
Signal Estado:std_logic_vector(1 downto 0));

Process(clk)
Begin
If  Reset='1' then  Estado <="000";
elsif clk='1' and clk'event then
Case Estado is
When "000" => Estado <="011";
--este case representa la lógica combinacional
When "011" => Estado <="100";
--para calcular el siguiente estado
When "100" => Estado <="110";
When "110" => Estado <="000";
When others  => null;
End case;
Else null;
End if;
End process;
Z<=Estado;
End registro;
```

9.1.3 Diseño de contador con entrada externa

En este problema se plantea diseñar un contador de dos *bits* que cuenta ascendentemente, siempre y cuando un *switch* X esté en 1, y contará de manera descendente si está en 0. En cualquier momento que X cambie de posición, el contador cambia el sentido del conteo a partir de su último estado.

La entrada se combina con cada estado para generar el próximo estado. La(s) entrada(s) se coloca(n) en las flechas indicando que, al ocurrir una transición de reloj, dependiendo del valor de la(s) entrada(s) el circuito se dirigirá hacia uno u otro estado.

©Editorial Digital Tecnológico de Monterrey

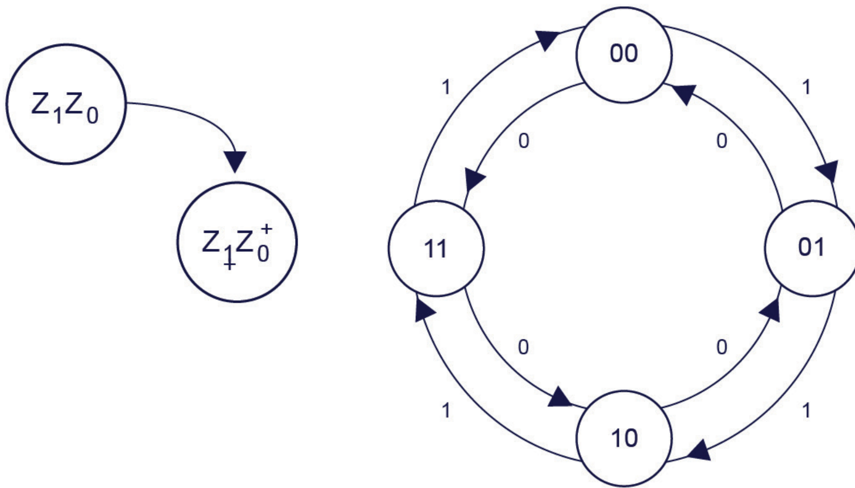


Figura 9.10 Diagrama de estados de contador con entrada externa

Como son dos *bits* de salida, se pueden utilizar dos *flip flops* con salidas Q1 y Q0.

Q1	Q0	X	Q1+	Q0+
0	0	0	1	1
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1
1	1	0	1	0
1	1	1	0	0

Tabla 9.3

©Editorial Digital Tecnológico de Monterrey

Si se extraen las funciones, usted puede comprobar que las funciones resultantes son:

$$D1 = X'Q1'Q0' + X'Q1\,Q0 + X\,Q1'Q0 + X\,Q1\,Q0' = (X \oplus Q1 \oplus Q0)' \ldots\ D0 = Q0'$$

Con el siguiente circuito:

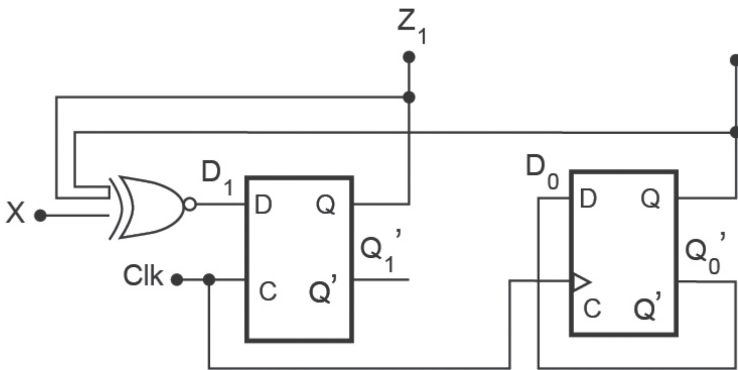


Figura 9.11 Circuito contador basado en flip flops con entrada externa

La codificación con *flip flops* queda muy evidente, así que se codificará únicamente por registro:

```
Entity contador is
Port (clk, X, Reset:in std_logic;
      Z:std_logic_vector(1 downto 0));
End contador;

Architecture flipflops of contador is
Signal Estado:std_logic_vector(1 downto 0));
```

©Editorial Digital Tecnológico de Monterrey

```
Process(clk)
Begin
if clk='1' and clk'event then
Case Estado is
When "00" => if X='0' then Estado <="11";
                      Else Estado <="01";
             End if;
When "01" => if X='0' then Estado <="00";
                      Else Estado <="10";
             End if;
When "10" => if X='0' then Estado <="01";
                      Else Estado <="11";
             End if;
When "11" => if X='0' then Estado <="10";
                      Else Estado <="00";
             End if;
When others  => null;
End case;
Else null;
End if;
End process;
Z<=Estado;
End registro;

if X='0' then Estado <="11";
                      Else Estado <="01";
               End if;
```

©Editorial Digital Tecnológico de Monterrey

9.2 Diseño de máquinas de estados

Al inicio del capítulo se mencionó que hay dos tipos de autómatas: los contadores y las **máquinas de estados**. En los contadores, el estado del circuito corresponde a sus salidas. Hay una gran variedad de problemas en los que las salidas de un autómata se repiten en la secuencia, o también puede ocurrir que el número de estados requerido en el diseño sobrepasa a 2s, donde s es el número de salidas. En estos casos se requiere un diseño distinto al de un contador, es decir, el diseño de un autómata o máquina de estados. Un autómata es un tipo de circuito y también es una herramienta muy poderosa de diseño de sistemas secuenciales. Los contadores también pueden diseñarse con esta herramienta.

Hay dos tipos básicos de máquinas de estados, que se conocen por los apellidos de sus autores: **máquina de Mealy** y máquina de Moore. La diferencia depende de las entradas de las funciones que emiten las salidas:

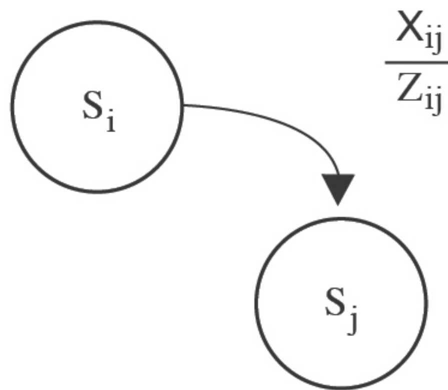


Figura 9.12 Máquina de Mealy

©Editorial Digital Tecnológico de Monterrey

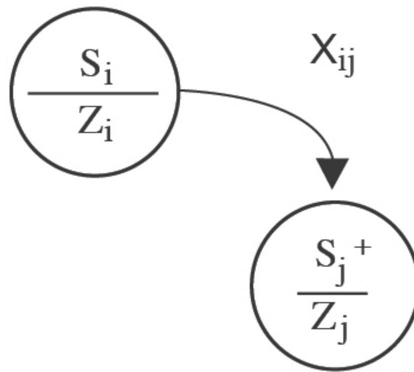


Figura 9.13 Máquina de Moore

La nomenclatura que se utiliza es:

S de *State* identifica un estado, en este texto no se usa E de "Estado" porque E podría confundirse con E de entrada.

X representa la(s) entrada(s) al circuito.

Z representa la(s) salida(s) del circuito.

En una máquina de Mealy las salidas Z se generan por un circuito combinacional cuyas entradas son Si y Xij.

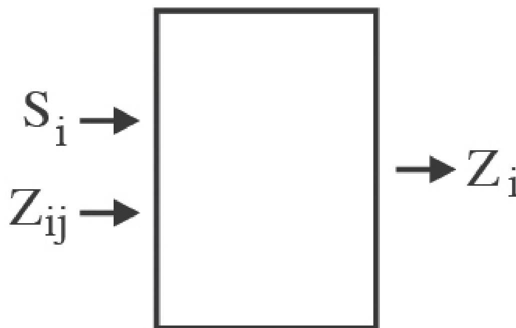


Figura 9.14 Salidas en una máquina de Mealy

©Editorial Digital Tecnológico de Monterrey

En una máquina de Moore la salida solo depende del estado, así que se generan a través de un circuito combinacional cuya entrada solo es Si.

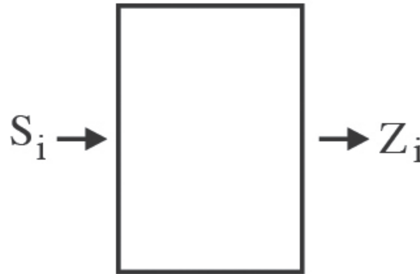


Figura 9.15 Salidas en una máquina de Moore

El funcionamiento es prácticamente el mismo, con la pequeña diferencia que en la máquina de Mealy la salida está sincronizada con la frecuencia y momento en que cambian las entradas, y en la máquina de Moore la salida está sincronizada con la frecuencia y momento en que cambian los estados.

La frecuencia del cambio de las entradas y de los estados debe ser la misma, solo pueden o no estar desfasados. Si no lo están, entonces ambas máquinas tienen exactamente el mismo funcionamiento. A medida que el lector gane práctica elegirá fácilmente uno u otro tipo de máquina; funcionalmente son equivalentes porque las salidas se producen con la misma frecuencia, solo con diferente fase. En resumen, en la máquina de Mealy la salida tiene la fase de la entrada, en la de Moore la del reloj.

El diseño de ambos tipos de máquinas se muestra con un ejemplo sencillo.

El problema viene en la siguiente sección.

©Editorial Digital Tecnológico de Monterrey

9.2.1 Bit de paridad impar

A un circuito ingresa una cadena serial de *bits*, uno a la vez, llamado X, con la frecuencia de una señal clk que también ingresa al circuito. El circuito debe generar un *bit* de salida Z que es un *bit* de paridad impar. Si la cantidad de unos que ha ingresado es impar el circuito genera un no, cero de otra manera.

Hay dos diseños de máquina para resolver este problema:

Por una máquina de Mealy, en la que ingresa un *bit* y se genera un *bit* X/Y:

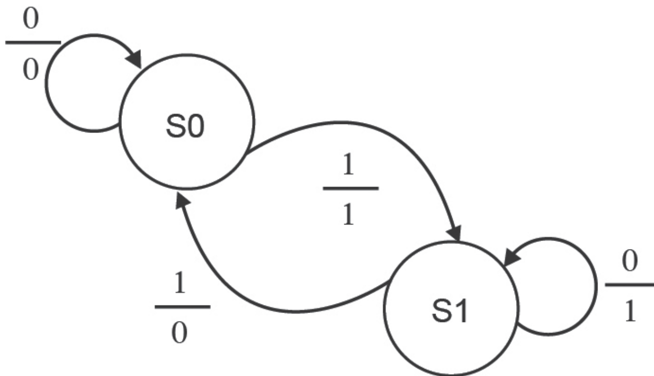


Figura 9.16 Máquina de Mealy para producir *bit* de paridad impar externa

El estado inicial es el estado 0 (S0). Si ocurre un 1 la salida es uno sin importar la cantidad de ceros que le sucedan; si ocurre otro 1 la salida es cero al convertirse la cuenta a par; si ocurre otro 1 vuelve a ser impar y la salida es uno.

Otra solución equivalente, esta vez con una **máquina de Moore**, es en la que la salida se produce en cada estado y con la entrada X hay cambio de estado en cada transición de reloj.

©Editorial Digital Tecnológico de Monterrey

El diseño es el siguiente:

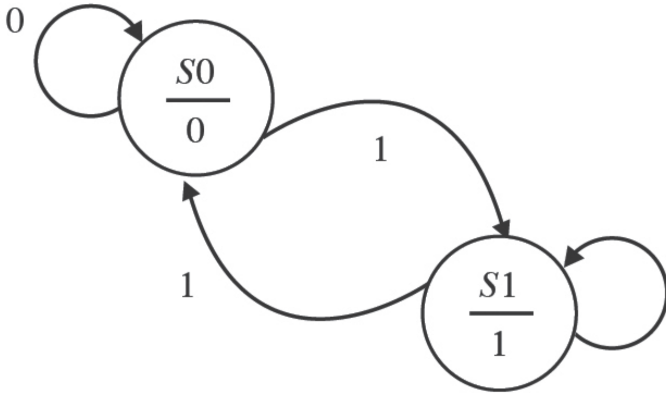


Figura 9.17 Máquina de Moore para producir *bit* de paridad impar

En general, la diferencia entre los dos tipos de máquinas es el momento en que es valida la salida, que a continuación se presenta gráficamente:

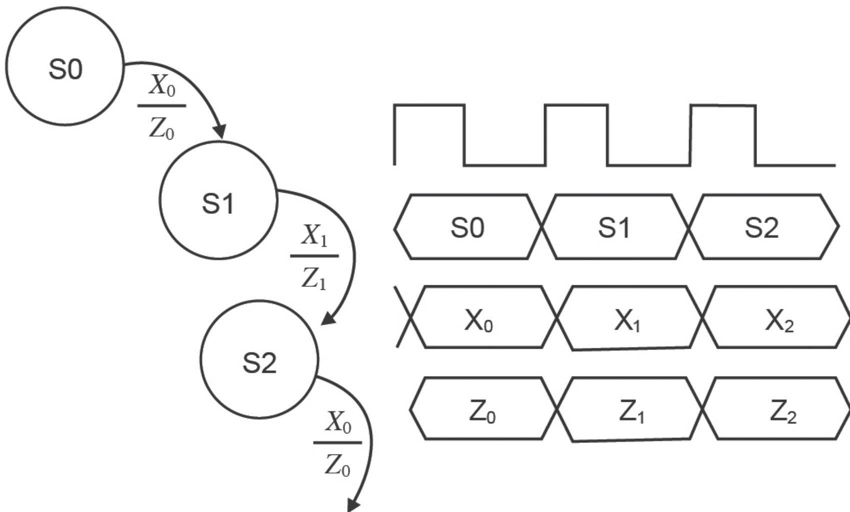


Figura 9.18 Diagrama de tiempos para máquina de Mealy

©Editorial Digital Tecnológico de Monterrey

En general, para una secuencia de entradas x0, x1, x2 … las salidas correspondientes son z0, z1, z2, que ocurren prácticamente sincronizadas con las entradas, solo con el retraso de un circuito combinacional.

En una máquina de Moore las salidas ocurren sincronizadas con los estados, solo con el retraso de un circuito combinacional, como se muestra en la siguiente gráfica:

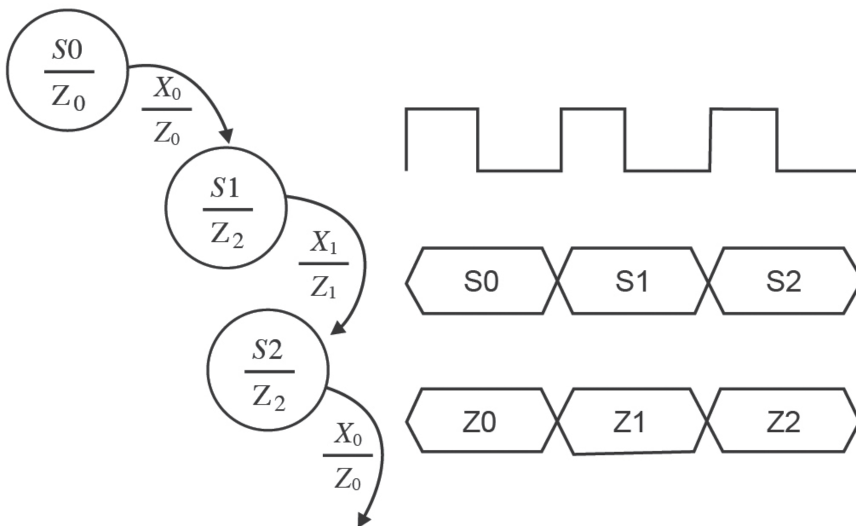


Figura 9.19 Diagrama de tiempos para máquina de Moore

A continuación, se presenta un conjunto de problemas en cuya solución se han utilizado distintos estilos, con el fin de que finalmente el lector adopte el suyo. La única forma de dominar la técnica del diseño de autómatas es a través de la solución de problemas. Eso es lo que se propone en este capítulo, presentar una variedad de problemas para que el lector resuelva y compare su solución.

©Editorial Digital Tecnológico de Monterrey

9.2.2 Diseño de una máquina de estados

Los elementos básicos de diseño de una máquina de estados se presentan con el siguiente problema:

Complementos a 2

La comunicación serial entre dispositivos es un recurso muy útil porque, en vez de requerir múltiples canales de comunicación, solo se requiere uno, lo cual economiza la comunicación (aunque la haga más lenta).

El siguiente problema requiere diseñar un circuito que reciba una secuencia de *bits* que ingresan de manera serial, y generar también una secuencia de unos y ceros. La secuencia consiste en una cadena serial de números de 4 *bits* que ingresan a partir de su *bit* menos significativo (no hay división alguna entre un número y otro, los números no tienen ningún formato) y genere el complemento a 2 de cada número de 4 *bits* también en forma serial, a partir de su *bit* menos significativo. Cada *bit* del complemento debe ser válido un delta de tiempo después de la entrada. Por ejemplo (separando por espacios los números cada cuatro *bits*, solo para efectos de identificar los números):

El circuito asegura su correcta operación cuando una señal de reset sube a 1 después de haber estado en 0.

El puerto del circuito es:

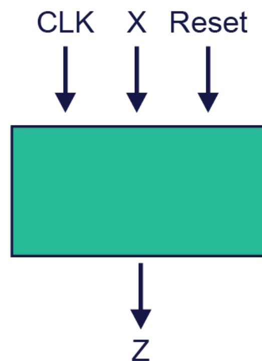


Figura 9.20 Puerto del circuito generador de complementos a 2

©Editorial Digital Tecnológico de Monterrey

Para este circuito solo hay dos posibles salidas: 0 o 1, y por la naturaleza del problema se requieren más de dos estados.

Una máquina de estados que resuelve este problema es una máquina de Mealy con el siguiente diseño:

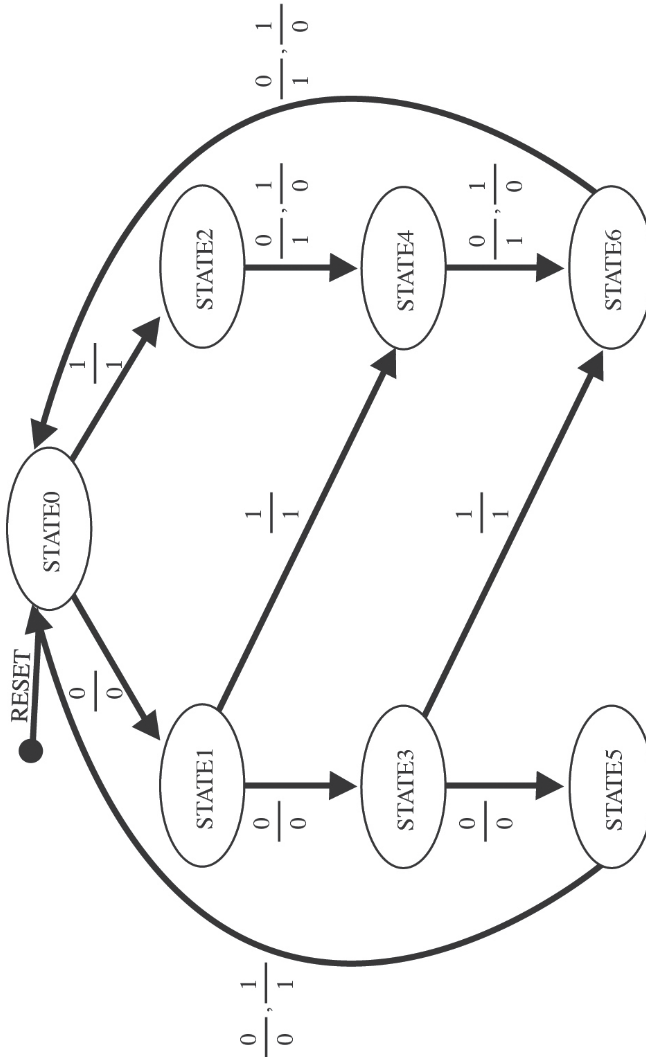


Figura 9.21 Máquina de Mealy para generador de complementos a 2

©Editorial Digital Tecnológico de Monterrey

La explicación es la siguiente: mientras ingresen ceros, las salidas son ceros, al ingresar el primer uno la salida es uno y las salidas subsecuentes se invierten. Cuando ya hayan ocurrido cuatro *bits* se regresa al estado inicial. Al diseñar una máquina de estados se debe considerar que se conecta un estado con un estado nuevo cuando el camino posterior sea nuevo, y que se llega a un estado ya existente cuando de ahí en adelante el comportamiento del circuito es igual al de las otras ramas conectadas a este, es decir, cuando la historia de los estados anteriores sea la misma. En este diagrama debe considerarse que en todos los ciclos de reloj hay un *bit* de entrada, por lo que no debe diseñarse ninguna flecha para conectar entradas que no tenga una entrada presente.

Para la implementación de este circuito basado en estados se requerirá un conjunto de n *flip flops* con los que pueda asociar hasta 2n estados, donde el número de estados debe estar entre 2n−1 y 2n.

Entonces, como en este ejemplo son siete estados, 4 < 7 < 8, 23−1 < 7 < 23 con 3 *flip flops* se pueden representar hasta 8 estados. En base en lo anterior, para este problema podemos utilizar tres *flip flops* con salidas Q2, Q1 y Q0.

Habrá que asociar a cada estado, arbitrariamente, una combinación de las salidas de los *flip flops*. Arbitrariamente porque puede ser cualquiera, con que sea una combinación distinta para cada estado. Y lo que va a pasar es que cuando se produzca esa combinación, y de acuerdo a la entrada, se va a producir una salida.

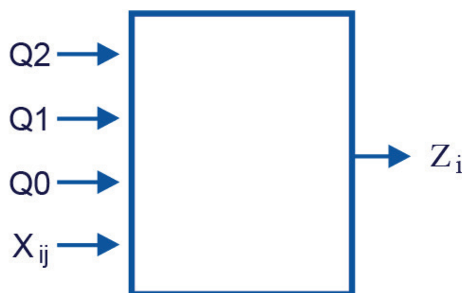


Figura 9.22 Diagrama de bloque de circuito combinacional para producir salida

©Editorial Digital Tecnológico de Monterrey

Y también se va a producir la combinación requerida para el siguiente estado:

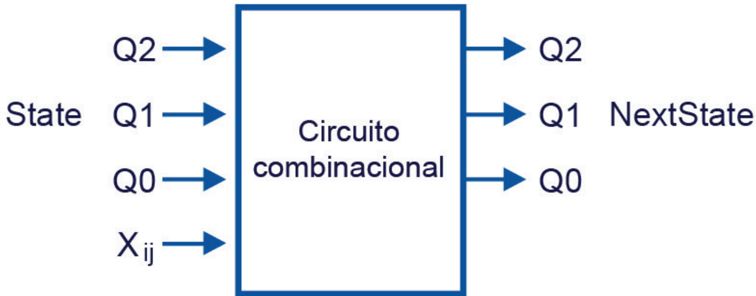


Figura 9.23 Diagrama de bloque de circuito combinacional para producir el siguiente estado

La codificación más directa y natural es asociar la combinación en binario correspondiente al número de estado. Esto conduce a:

State	Q2	Q1	Q0
S0	0	0	0
S1	0	0	1
S2	0	1	0
S3	0	1	1
S4	1	0	0
S5	1	0	1
S6	1	1	0

Tabla 9.4

©Editorial Digital Tecnológico de Monterrey

Con esto se puede diseñar la tabla de verdad para entender a fondo el funcionamiento del circuito.

State	Q2	Q1	Q0	X	Q2+	Q1+	Q0+	Z	NextState+
S0	0	0	0	0	0	0	1	0	S1
S1	0	0	0	1	0	1	0	1	S2
	0	0	1	0	0	1	1	0	S3
	0	0	1	1	1	0	0	1	S4
S2	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**S4**
S3	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**S4**
	0	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**S5**
	0	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**S6**
S4	1	0	0	0	1	1	0	1	S6
S5	1	0	0	1	1	1	0	0	S6
	1	0	1	0	0	0	0	0	S0
	1	0	1	1	0	0	0	1	S0
S6	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**S0**
X	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**S0**
	1	**1**	**1**	**0**	**X**	**X**	**X**	**X**	**X**
	1	**1**	**1**	**1**	**X**	**X**	**X**	**X**	**X**

Tabla 9.5

©Editorial Digital Tecnológico de Monterrey

El diagrama esquemático del circuito es el siguiente:

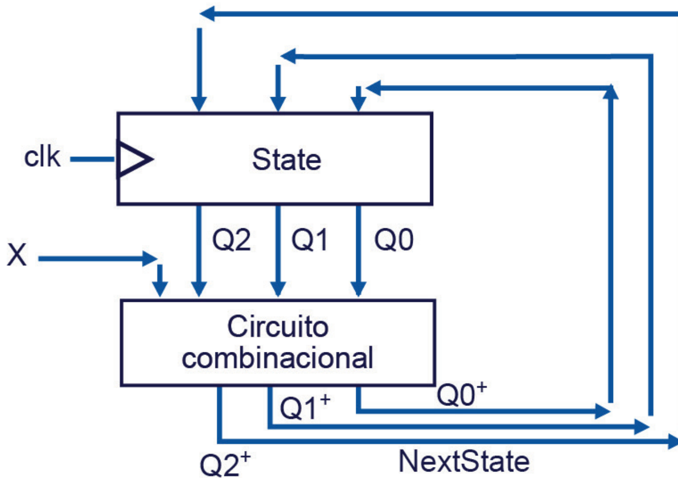


Figura 9.24 Diagrama de bloques general para producir el siguiente estado

State se puede construir con tres *flip flops* independientes, o bien, con un registro de tres *bits*.

Si se construye con tres *flip flops* habrá que extraer las tres funciones Q2+, Q1+, Q0+ de la tabla de verdad, ya sea en forma canónica o simplificada.

Funciones Originales

$D_2 = Q_2'Q_1'Q_0X + Q_2'Q_1Q_0'X' + Q_2'Q_1Q_0'X + Q_2'Q_1Q_0X' + Q_2'Q_1Q_0X + Q_2Q_1'Q_0'X' + Q_2Q_1'Q_0'X$

$D_1 = Q_2'Q_1'Q_0'X + Q_2'Q_1'Q_0X' + Q_2'Q_1Q_0X + Q_2Q_1'Q_0'X' + Q_2Q_1'Q_0'X$

$D_0 = Q_2'Q_1'Q_0'X' + Q_2'Q_1'Q_0X' + Q_2'Q_1Q_0X'$

$Z = Q_2'Q_1'Q_0'X + Q_2'Q_1'Q_0X + Q_2'Q_1Q_0'X' + Q_2'Q_1Q_0X + Q_2Q_1'Q_0'X' + Q_2Q_1'Q_0X + Q_2Q_1Q_0'X'$

©Editorial Digital Tecnológico de Monterrey

Funciones simplificadas

$D_2 = Q_2'Q_1 + Q_2'Q_0X + Q_2Q_1'Q_0'$

$D_1 = Q_1Q_0X + Q_1'Q_0'X + Q_2Q_1'Q_0' + Q_2'Q_1'Q_0X'$

$D_0 = Q_2'Q_1'X' + Q_2'Q_0X'$

$Z = Q_0X + Q_2'Q_1'X + Q_2Q_0'X' + Q_1Q_0'X'$

©Editorial Digital Tecnológico de Monterrey

Una vez extraídas las funciones de la tabla de verdad, se llega al siguiente circuito:

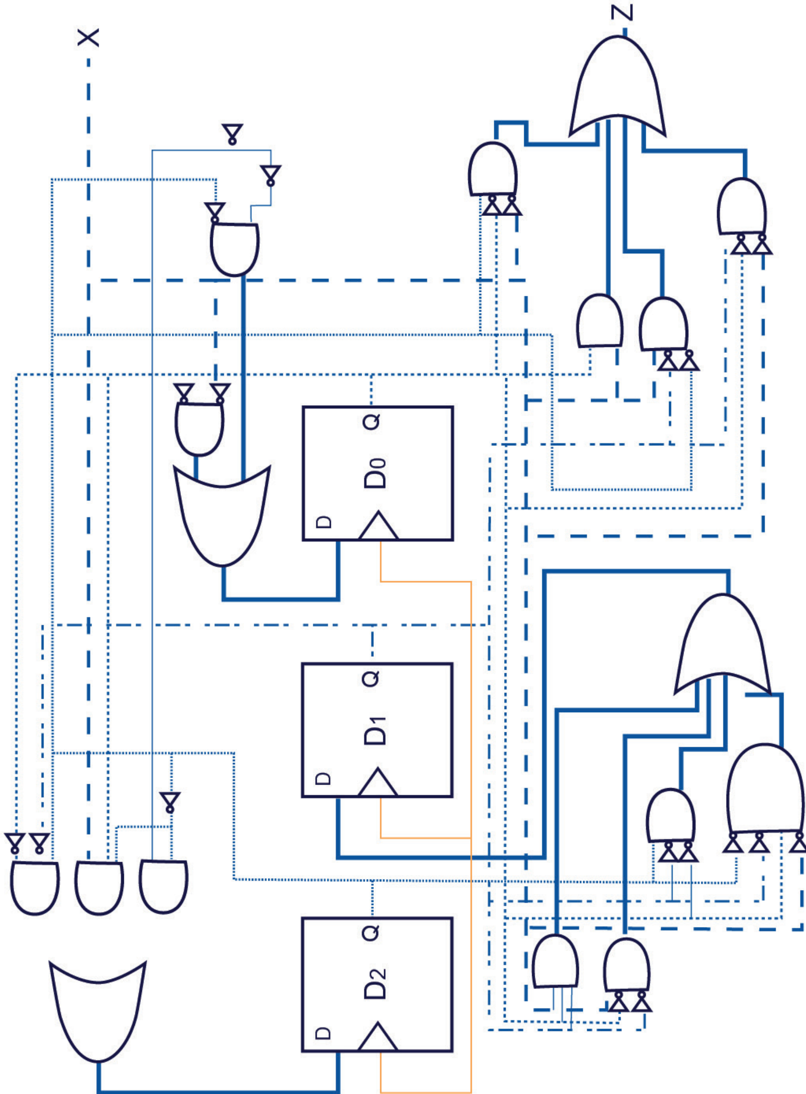


Figura 9.25 Diagrama de circuito generador de complementos a 2 basado en *flip flops*

©Editorial Digital Tecnológico de Monterrey

Si el circuito se construye utilizando el circuito State y se describe en VHDL el circuito combinacional, puede quedar descrito de la siguiente forma (observando el comportamiento de la máquina del diagrama de estados, ni siquiera es necesario construir la tabla de verdad):

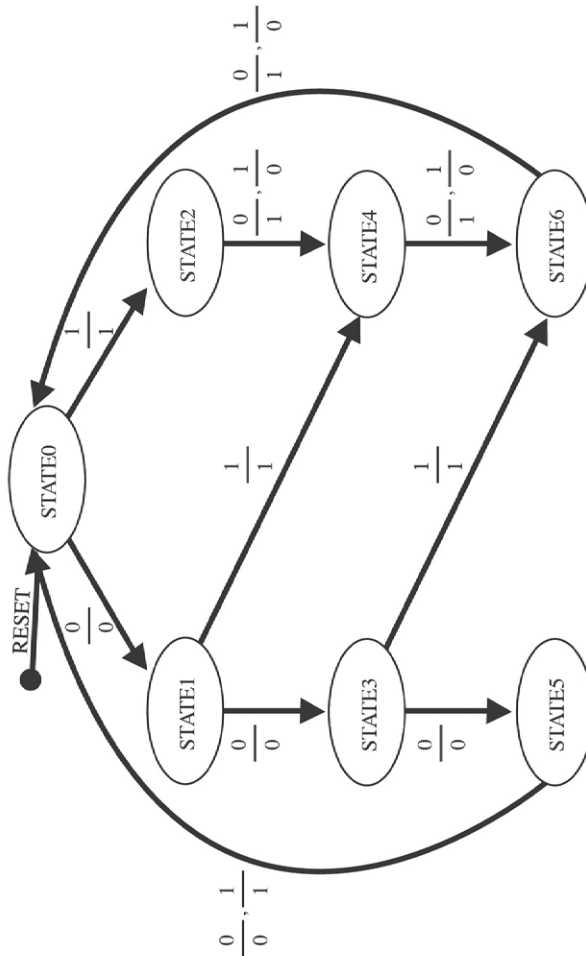


Figura 9.26 Máquina de Mealy para generador de complementos a 2

©Editorial Digital Tecnológico de Monterrey

Observando nuevamente el diseño de esta máquina, no resulta sencillo convertirlo a una máquina de Moore, pues habría que añadirle más estados, ya que de un estado a otro hay ramas con diferentes salidas. Tal es el caso del estado 2 al 4, además de que no habría manera de determinar la salida del estado 0.

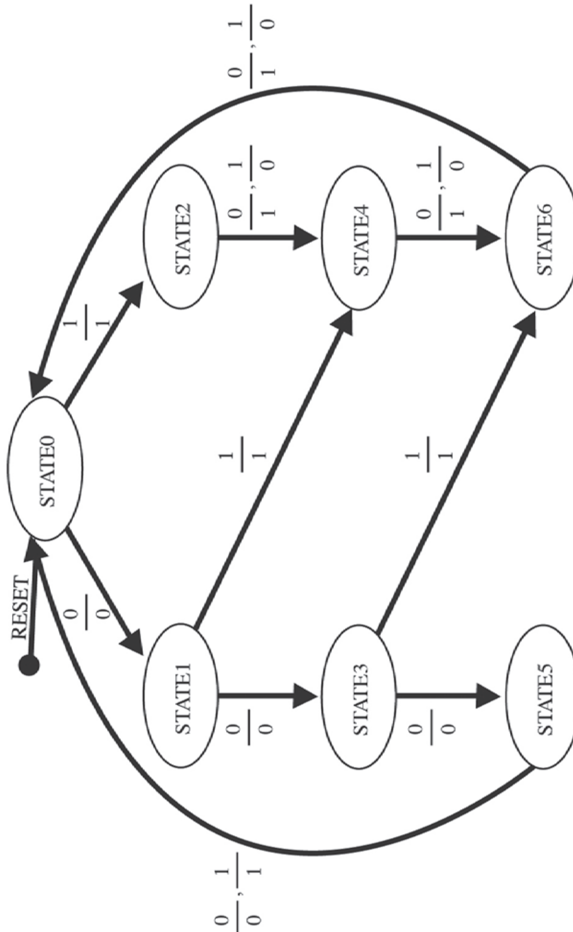


Figura 9.27 Máquina de Mealy para generador de complementos a 2

©Editorial Digital Tecnológico de Monterrey

Código:

```
entity comp2 is
  Port ( Reset, X : in std_logic;
      Z : out std_logic;
      clk  : in std_logic);
end comp2;

architecture Behavioral of comp2 is
signal State, Nextstate: integer := 0;
begin
process(State,X) -Circuito combinacional
begin
case State is
when 0 =>
  if X='0' then Z<='0'; Nextstate<=1;
  elsif X='1' then Z<='1'; Nextstate<=2;
  else null;
  end if;
when 1 =>
  if X='0' then Z<='0'; Nextstate<=3;
  elsif X='1' then Z<='1'; Nextstate<=4; else null;
  end if;
  when 2 =>
if X='0' then Z<='1'; Nextstate<=4;
  elsif X='1' then Z<='0'; Nextstate<=4;
  else null;
  end if;
when 3 =>
  if X='0' then Z<='0'; Nextstate<=5;
  elsif X='1' then Z<='1'; Nextstate<=6;
  else null;
  end if;
```

©Editorial Digital Tecnológico de Monterrey

```
when 4 =>
  if X='0' then Z<='1'; Nextstate<=6;
  elsif X='1' then Z<='0'; Nextstate<=6;
  else null;
  end if;
when 5 =>
  if X='0' then Z<='0'; Nextstate<=0;
  elsif X='1' then Z<='1'; Nextstate<=0;
  else null;
  end if;
when 6 =>
  if X='0' then Z<='1'; Nextstate<=0;
  elsif X='1' then Z<='0'; Nextstate<=0;
  else null;
  end if;
when others => null; -- should not occur
end case;
end process;
process(CLK , start) -- State Register
begin
if Reset='1' then State<=0;
elsif (clk ='1')and clk 'event then -- rising edge of clock
State <= Nextstate;
end if;
end process;
end Behavioral;
```

Más adelante se plantea otro problema en el que sí es posible diseñar tanto una máquina de Mealy, como una máquina de Moore.

A continuación, se presenta a manera de ejercicio de repaso, un problema similar.

9.2.3 Ejemplo de diseño de una máquina de estados

Código exceso 3

A un circuito ingresa un *bit* y una señal de reloj. El *bit* proviene de una cadena de números BCD transmitidos en serie. Cada número BCD ingresa a partir de su *bit* menos significativo. Lo que se propone para este circuito es que calcule los códigos exceso 3 de los números BCD de entrada también de manera serial, y que los presente en un *bit* de salida serial también a partir del *bit* menos significativo.

©Editorial Digital Tecnológico de Monterrey

El código exceso 3 no es más que la suma del número de entrada más tres, la dificultad de la lógica es calcular la suma del número que va ingresando serialmente. A continuación, se presenta un ejemplo del *bit* X que ingresa de acuerdo a una señal de reloj y del *bit* Z que se produce.

Se recalca que ingresan números BCD en serie a partir del *bit* menos significativo y se generan números BCD más 3 en serie a partir del *bit* menos significativo.

```
X      010011100110101010000000…
Z      101001011001000100101100…
```

Separando los *bits* de cuatro en cuatro, para que el ejemplo sea más claro, ocurre que:

```
X      0100 1110 0110 1010 1000 0000…
Z      1010 0101 1001 0001 0010 1100…
```

Ya que 2 + 3 = 5, 7 + 3 = 10, 6 + 3 = 9, 1 + 3 = 4, 0 + 3 = 3.

Un diseño apropiado lo constituye una máquina de Mealy. Se pueden diferenciar de un lado los estados en los que no ocurre un acarreo (“*carry*”) y del otro los estados en los que sí.

©Editorial Digital Tecnológico de Monterrey

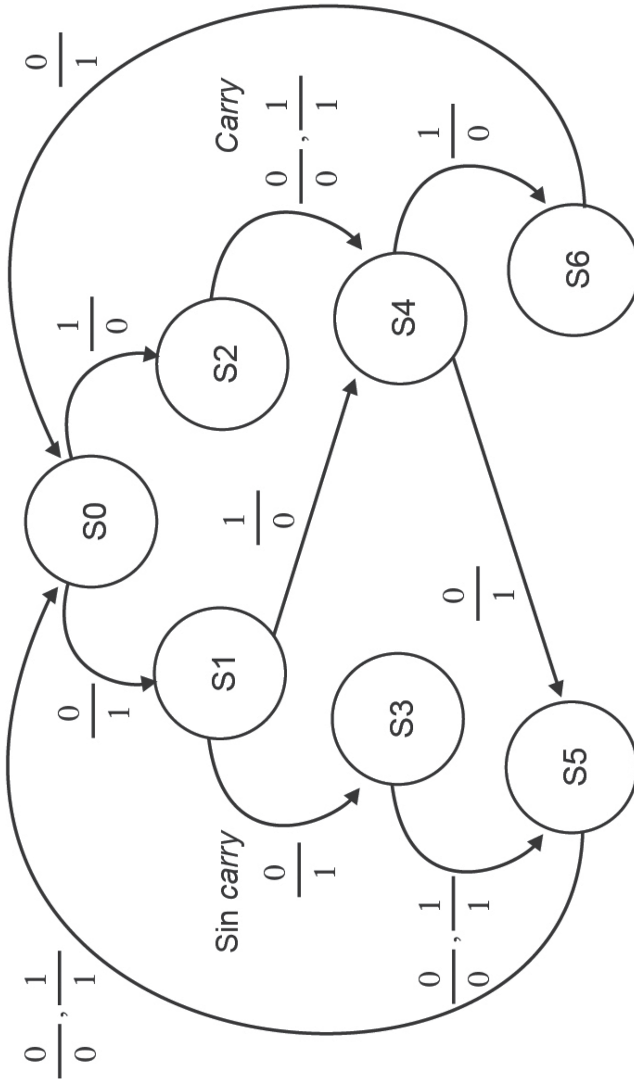


Figura 9.28 Diagrama de estados de máquina de Mealy que genera código exceso 3

©Editorial Digital Tecnológico de Monterrey

```
entity exc3behavioral1 is
  Port (x, clk , reset:in std_logic; z:out std_logic);
end exc3behavioral1;

architecture Behavioral of exc3behavioral1 is
signal State, Nextstate: integer;
begin
process(State,X) --Combinational Network
begin
case State is
when 0 =>  if X='0' then Z<='1'; Nextstate<=1;
           elsif X='1' then Z<='0'; Nextstate<=2;
           else null;
           end if;
when 1 =>  if X='0' then Z<='1'; Nextstate<=3;
           elsif X='1' then Z<='0'; Nextstate<=4;
           else null;
           end if;
when 2 =>  if X='0' then Z<='0'; Nextstate<=4;
           elsif X='1' then Z<='1'; Nextstate<=4;
           else null;
           end if;
when 3 =>  if X='0' then Z<='0'; Nextstate<=5;
           elsif X='1' then Z<='1'; Nextstate<=5;
           else null;
           end if;
when 4 =>  if X='0' then Z<='1'; Nextstate<=5;
           elsif X='1' then Z<='0'; Nextstate<=6;
           else null;
           end if;
```

©Editorial Digital Tecnológico de Monterrey

```
when 5 =   if X='0' then Z<='0'; Nextstate<=0;
           elsif X='1' then Z<='1'; Nextstate<=0;
           else null;
           end if;
when 6 =>  if X='0' then Z<='1'; Nextstate<=0;
           else null;
           end if;
when others => null; -- should not occur
end case;
end process;
process(CLK , reset) -- State Register
begin
if reset='1' then State<=0;
elsif CLK ='1' and Clk 'event then -- rising edge of clock
State <= Nextstate;
end if;
end process;
end Behavioral;
```

A continuación, se presenta el reporte del sintetizador para este diseño, que muestra que el diseño se reconoce como una máquina de estados.

©Editorial Digital Tecnológico de Monterrey

```
Synthesizing Unit <exc3behavioral1>.

    Related source file is C:/xilinx_webpack/bin/COMB/exc3behavioral1.vhd.
    Found finite state machine <FSM_0> for signal <state>.
    -----------------------------------------------------------------------
    | States              | 7                          |
    | Transitions         | 11                         |
    | Inputs              | 1                          |
    | Outputs        | 1                          |
    | Reset type          | asynchronous         |
    | Encoding       | automatic                  |
    | State register | D  flip-flops              |
    -----------------------------------------------------------------------
WARNING:Xst:737 - Found 1-bit latch for signal <z>.
    Summary:
     inferred   1 Finite State Machine(s).
     inferred   1 Latch(s).
Unit <exc3behavioral1> synthesized.

=========================================================
HDL Synthesis Report

Macro Statistics
# FSMs                     : 1 (finite state machine)
# Latches                  : 1
1-bit latch                : 1
```

Antes de proseguir, se esbozan un grupo de consideraciones para tomar en cuenta en el momento de diseñar un autómata. Estas consideraciones son solamente de notación, finalmente si se usa VHDL para traducir el autómata a un circuito, no es necesario ajustarse a un estilo de notación, así que el mejor es el que sea más claro.

©Editorial Digital Tecnológico de Monterrey

9.2.4 Opciones de notación en el diseño de diagramas de estados

Consideración necesaria: la suma lógica de ramas debe ser 1. Las entradas a un circuito (si las hay) son las que causan el cambio de un estado a otro. En cada estado deben considerarse todas las combinaciones de entradas. Esto se traduce a que la suma lógica de las ramas debe ser 1. A continuación, se presentan casos representativos.

Entradas

Una entrada. Al diseñar un autómata con solo una entrada, llamémosle X, las diferentes formas para expresar las condiciones lógicas de cambio de estado son las que se muestran en la imagen 9.28.

En el último caso de la figura, la rama que no se especifica se supone ser el complemento lógico del resto de las ramas.

©Editorial Digital Tecnológico de Monterrey

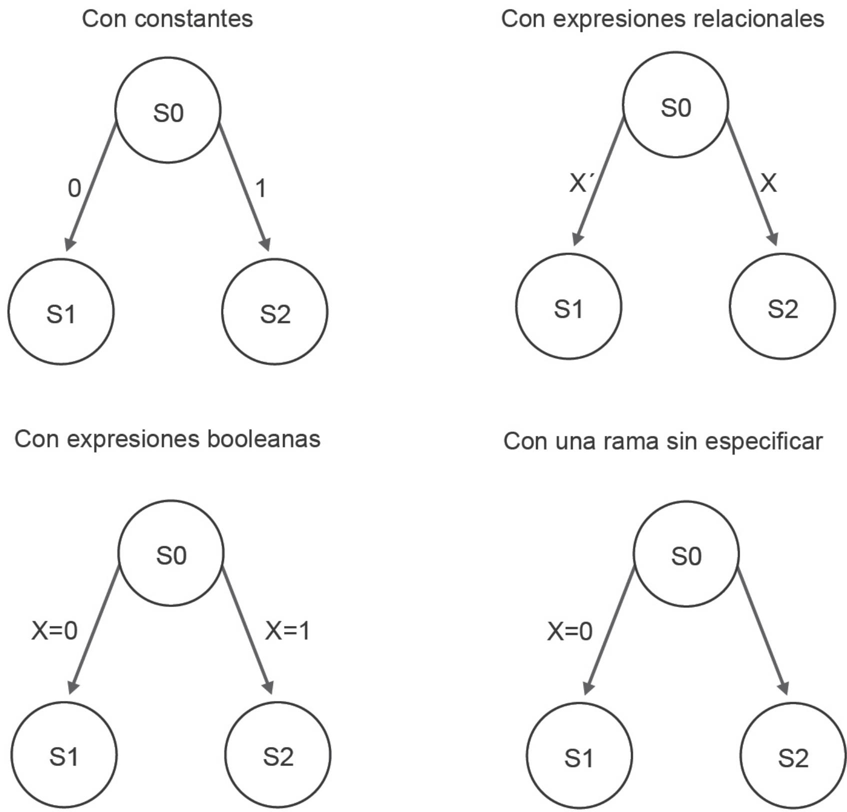


Figura 9.29 Alternativas de notación para la especificación de ramas con una entrada en un diagrama de estados

Dos o más entradas. En este caso hay que especificar previamente el orden de las entradas.

- Constantes

©Editorial Digital Tecnológico de Monterrey

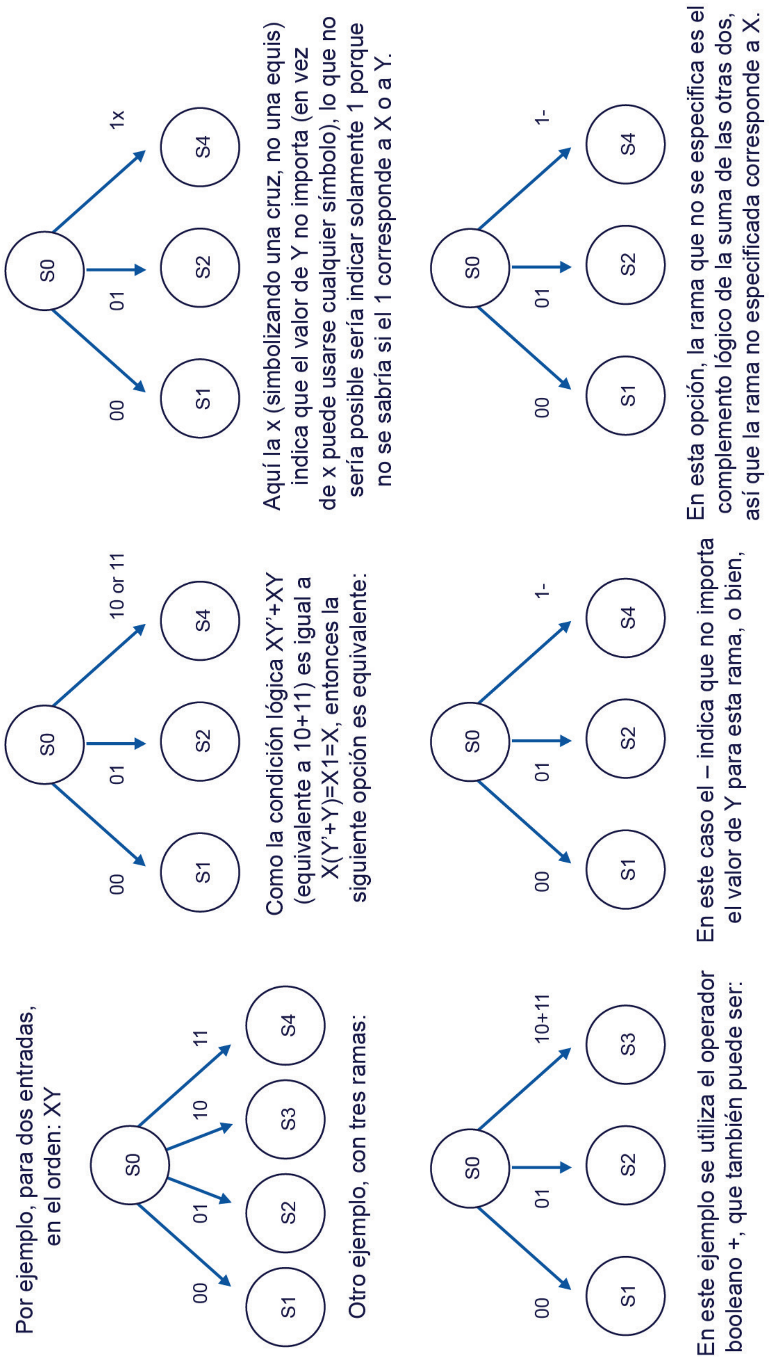


Figura 9.30 Especificación de ramas de dos entradas con expresiones relacionales

©Editorial Digital Tecnológico de Monterrey

• Expresiones relacionales

Aquí se presentan tres opciones equivalentes con pequeñas variaciones en la notación.

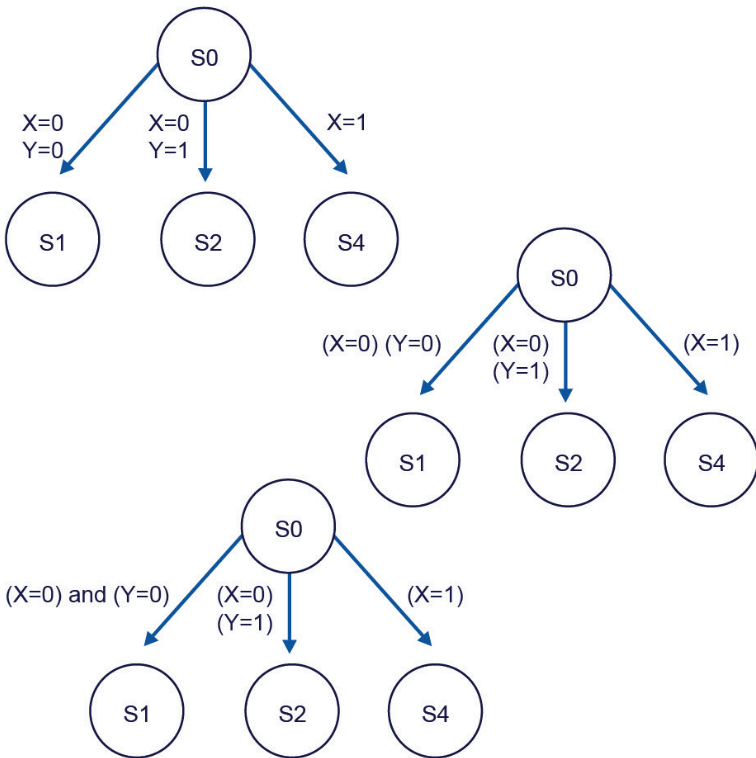
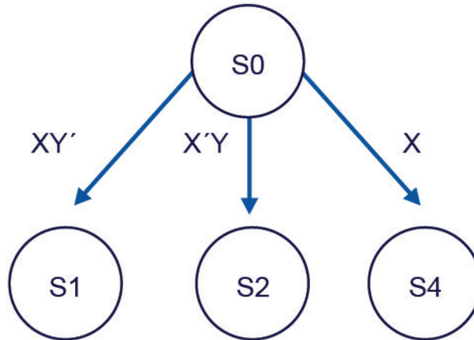


Figura 9.31 Especificación de ramas de dos entradas con expresiones relacionales

©Editorial Digital Tecnológico de Monterrey

- Con expresiones booleanas

El mismo ejemplo es:



O bien,

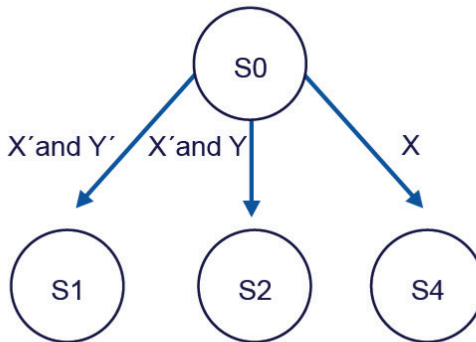


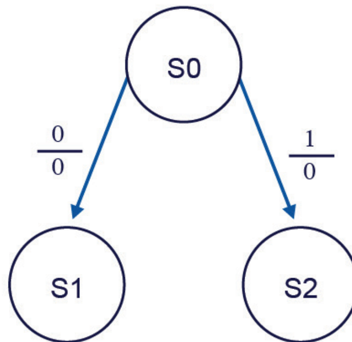
Figura 9.32 Especificación de ramas de dos entradas con expresiones booleanas

©Editorial Digital Tecnológico de Monterrey

Especificaciones de salida

Para especificar las salidas se mostrarán las opciones con una máquina de Mealy, mostrando así una variedad de estilos sin agotarlas.

• Con constantes



O bien,

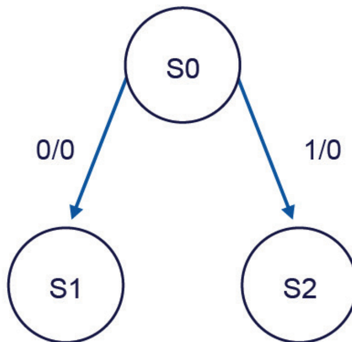
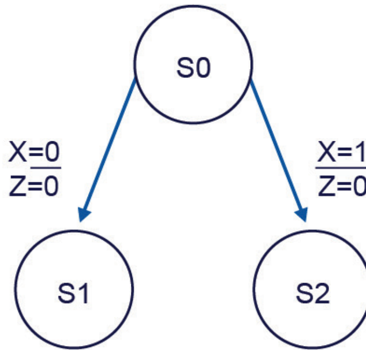


Figura 9.33 Especificación de salidas con constantes

©Editorial Digital Tecnológico de Monterrey

• Con expresión relacional



En este caso el operador "=" no significa lo mismoen la entrada que en la salida; en la entrada construye una expresión relacional y en la salida indica una asociación de un valor a la variable de salida. Por ejemplo, la primera rama se leería: si X es igual a 0 entonces a Z se le asigna 1. En este mismo contexto, otro ejemplo es:

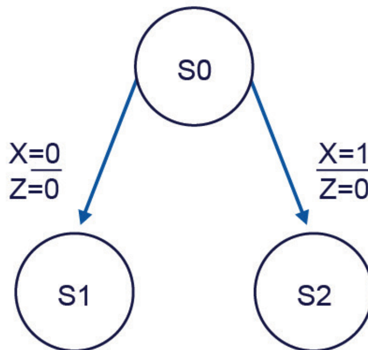


Figura 9.34 Especificación de salidas con expresiones relacionales

©Editorial Digital Tecnológico de Monterrey

• Con expresiones booleanas

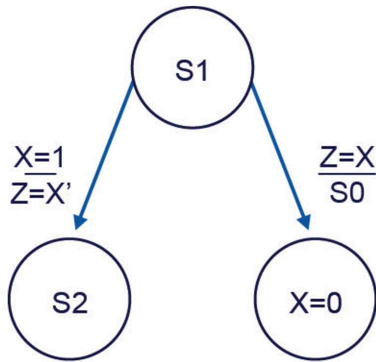
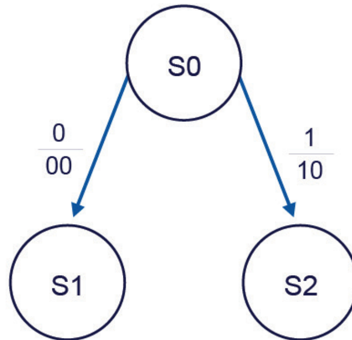


Figura 9.35 Especificación de salidas con expresiones booleanas

©Editorial Digital Tecnológico de Monterrey

Dos salidas. Con la misma tónica se utiliza la notación para varias variables de salida; si se usan constantes se debe indicar claramente su orden. A continuación, se incluirán un par de ejemplos.

Para las entradas y salidas: X/Z1Z2



O bien,

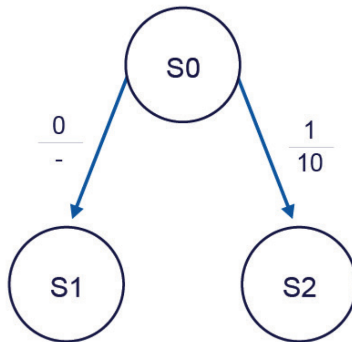


Figura 9.36 Especificación de dos salidas con constantes

©Editorial Digital Tecnológico de Monterrey

Generalmente con un “-“ se denota que todas las salidas son ceros.

Cuando se tienen varias salidas, también es común indicar las salidas que sean 1 y omitir los 0, por ejemplo:

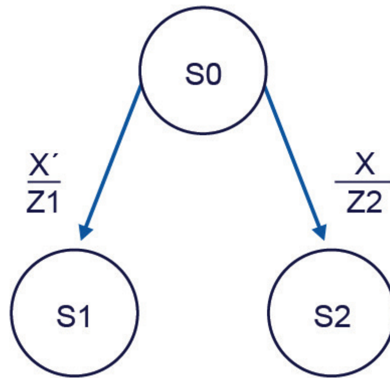


Figura 9.37 Especificación de salidas que sean uno

Aquí se asumiría que si X = 0 la salida Z1 es 1 y Z2 es 0; y si X = 1 la salida Z1 es 0 y Z2 es 1. Esta es una notación muy común cuando hay varias salidas. Esta opción se usa para simplificar el dibujo.

En algún ejemplo podría presentarse que se puede cambiar de un estado a otro por dos ramas.

©Editorial Digital Tecnológico de Monterrey

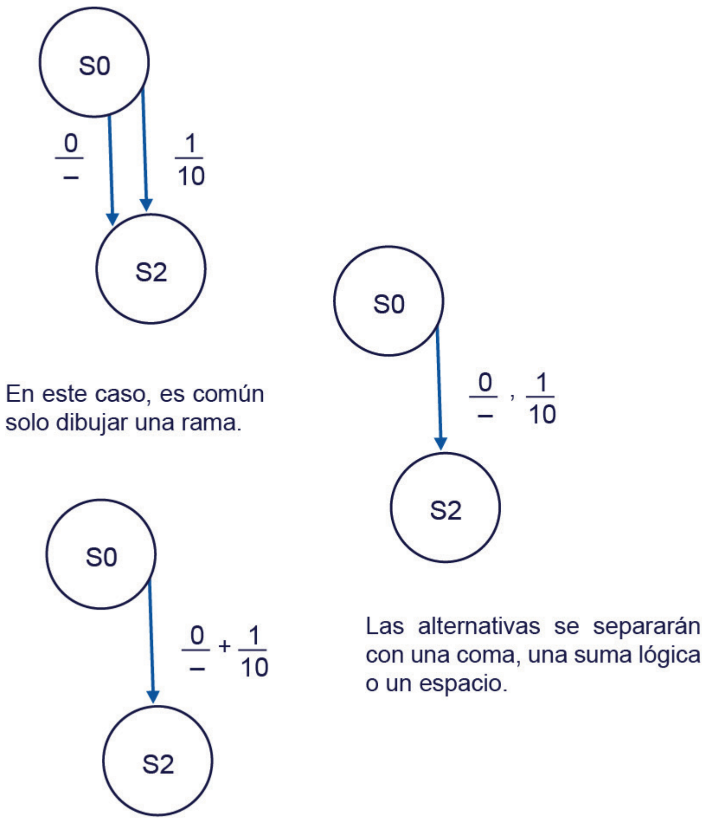


Figura 9.38 Conexión de dos estados con dos ramas

Cambio de estado directo. Si al cambiar de un estado a otro las entradas no importan, entonces los estados se conectarán con una flecha.

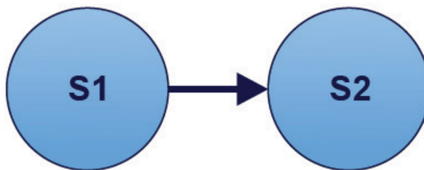


Figura 9.39 Cambio de estado sin entradas

©Editorial Digital Tecnológico de Monterrey

En el último caso de la **figura 9.39** se simboliza que no importan las entradas para que suceda un cambio de estado. Por supuesto, se puede haber diseño un híbrido en el que se experimenten diferentes notaciones para las entradas y las salidas (variables con constantes, etc.).

En los autómatas que resuelven los siguientes problemas se alternarán estilos para que, como se mencionó, el lector tome el que más se le facilite.

Con el siguiente ejemplo se presenta una variedad de las vistas para el diseño del diagrama de estados, así como diversas formas de codificación.

9.2.5 Alternativas de codificación de una máquina de Moore

Con el siguiente problema se mostrará en la práctica algunas opciones de diseño de diagramas de estados, además de su codificación en VHDL.

Juego de la adivinanza

Este juego cuenta con tres botones G1-G3 y tres LEDs L1-L3 que están localizados frente a su botón correspondiente (es decir, L1 está frente a G1, L2 frente a G2 y así sucesivamente). Adicionalmente hay otros dos indicadores luminosos: Acierto y Error.

El juego opera de la siguiente manera: cada uno de los tres leds L1-L3 se va encendiendo en secuencia, primero L1, luego L2, enseguida L3, de nuevo L1, etc., al encenderse el siguiente se apaga el anterior. El tiempo que permanece encendido el LED es de 0.25 seg. Si al estar prendido un LED, el usuario del juego oprime el botón correspondiente al LED encendido, se prende el LED de Acierto y se apagan los demás, quedándose así hasta que se deja de oprimir botón; en ese momento el dispositivo comienza nuevamente a mostrar la secuencia de leds, iniciando por L1. Por otra parte, si al estar encendido un LED, el usuario del juego oprime un botón que no corresponda al LED encendido, se prende el LED de Error y se apagan los demás, quedándose así

©Editorial Digital Tecnológico de Monterrey

hasta que se suelta al botón; en ese momento el dispositivo comienza nuevamente a mostrar la secuencia de LEDs, iniciando por L1.

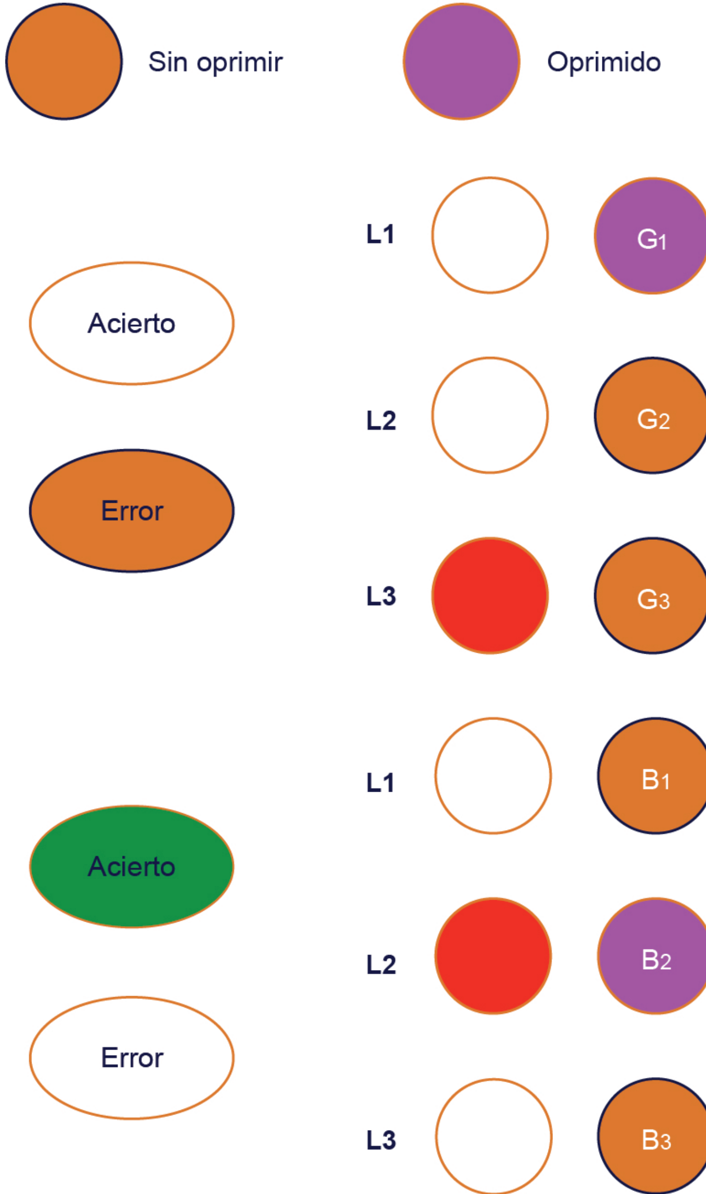


Figura 9.40

©Editorial Digital Tecnológico de Monterrey

Como solución se le sugiere diseñar una máquina de Moore.

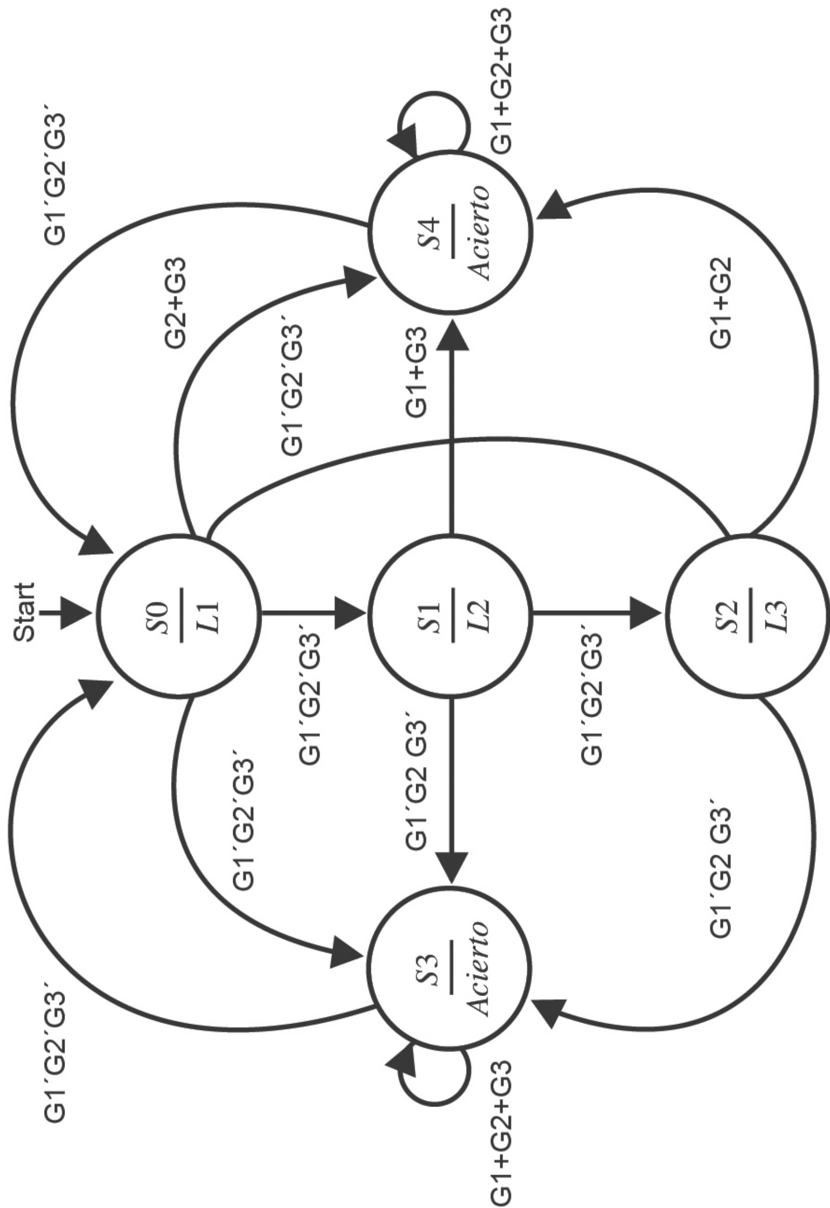


Figura 9.41 Máquina de Moore para juego de la adivinanza con tres entradas

©Editorial Digital Tecnológico de Monterrey

Explicación. Si se oprime el botón adecuado (solo un botón, los otros no) en la luz que le corresponde, llevará al estado de acierto.

Si no hay botones oprimidos la secuencia es L1, L2, L3, L1, y así sucesivamente.

Un botón inadecuado llevará a la luz de error. Estando en acierto o en error el circuito deberá aguardar hasta que estén todos los botones sin oprimir para regresar al estado de inicio.

Reduzca el juego a 2 LEDs, dos botones y los LEDs de acierto y error, y haga lo siguiente:

a) Diseñe el autómata.

©Editorial Digital Tecnológico de Monterrey

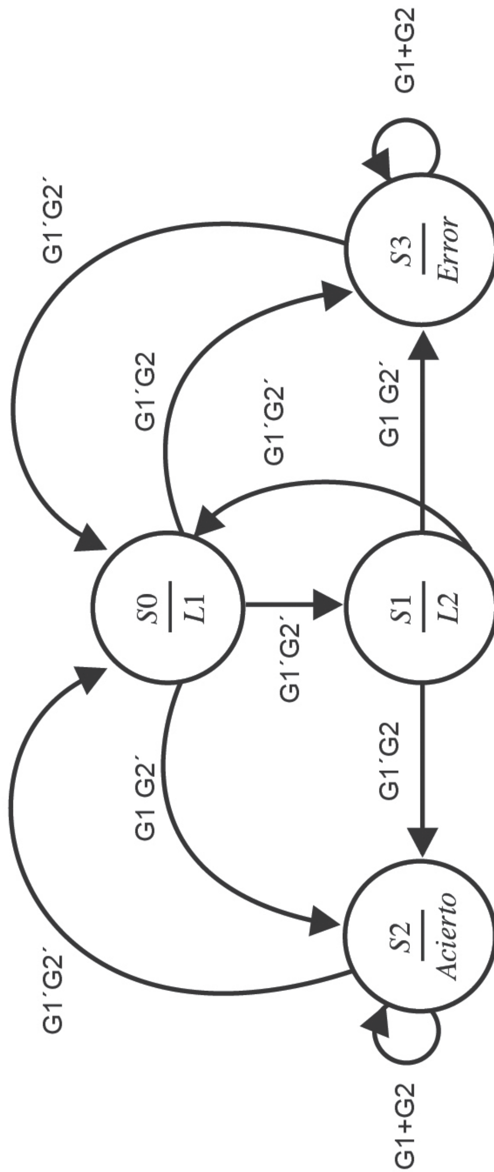


Figura 9.42 Máquina de Moore para juego de la adivinanza con dos entradas

©Editorial Digital Tecnológico de Monterrey

Enseguida se presenta el diseño manejando los estados en flip flops. La tabla de transiciones es la siguiente:

	Q1	Q0	G1	G2	Q1+	Q0+	
S0	0	0	0	0	0	1	S2
	0	0	0	1	1	1	S3
	0	0	1	0	1	0	S2
	0	0	1	1	1	1	S3
S1	**0**	**1**	**0**	**0**	**0**	**0**	**S0**
	0	**1**	**0**	**1**	**1**	**0**	**S2**
	0	**1**	**1**	**0**	**1**	**1**	**S3**
	0	**1**	**1**	**1**	**1**	**1**	**S3**
S2	1	0	0	0	0	0	S0
	1	0	0	1	1	0	S2
	1	0	1	0	1	0	S2
	1	0	1	1	1	0	S2
S3	**1**	**1**	**0**	**0**	**0**	**0**	**S0**
	1	**1**	**0**	**1**	**1**	**1**	**S3**
	1	**1**	**1**	**0**	**1**	**1**	**S3**
	1	**1**	**1**	**1**	**1**	**1**	**S3**

Tabla 9.6

©Editorial Digital Tecnológico de Monterrey

Al simplificar las dos funciones, las expresiones lógicas que resultan son:

```
Q1+ = G1 + G2
Q0+ = Q0G1 + Q1’Q0’G1’ + Q1Q0G2 + (Q1’Q0’G2  ó  Q1’G1G2)
```

Si se describe el circuito con dos *flip flops* D, el circuito queda descrito de la siguiente manera: codificación por *flip flops*.

```
entity juego_ec is
Port ( clk ,start : in std_logic;
       g1 : in std_logic;
       g2 : in std_logic;
       l1 : out std_logic;
       l2 : out std_logic;
       acierto : out std_logic;
       error : out std_logic);
end juego_ec;

architecture Behavioral of juego_ec is
signal State, Nextstate: std_logic_vector(1 downto 0);

begin

Nextstate(1)<=G1 or G2;
Nextstate(0)<=(Q0 AND G1) OR (NOT Q1 AND NOT Q0 AND NOT G1) OR
(Q1 AND Q0 AND G2) OR (NOT Q1 AND G1 AND G2);
```

```
process(CLK , start) -- State Register
begin
if start='1' then State<="00";
elsif CLK ='1' and CLK 'event then -- rising edge of clock
State <= Nextstate;
end if;
end process;

l1<='1' when state=”00” else '0';
l2<='1' when state=”01” else '0';
acierto<='1' when state=”10” else '0';
error<='1' when state=”11” else '0';
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Otra opción de codificación es incorporar la descripción combinacional dentro de la descripción de los *flip flops*:

```
entity juego_ec is
Port ( clk ,start : in std_logic;
       g1 : in std_logic;
       g2 : in std_logic;
       l1 : out std_logic;
       l2 : out std_logic;
       acierto : out std_logic;
       error : out std_logic);
end juego_ec;

architecture Behavioral of juego_ec is
signal State, Nextstate: std_logic_vector(1 downto 0);
```

```
begin
process(CLK , start) -- State Register
begin
if start='1' then State<="00";
elsif CLK ='1' and CLK 'event then -- rising edge of clock
State(1)<=G1 or G2;
State(0)<=(Q0 AND G1) OR (NOT Q1 AND NOT Q0 AND NOT G1) OR
(Q1 AND Q0 AND G2) OR (NOT Q1 AND G1 AND G2);
end if;
end process;

l1<='1' when state="00" else '0';
l2<='1' when state="01" else '0';
acierto<='1' when state="10" else '0';
error<='1' when state="11" else '0';
end Behavioral;
```

Se hará un breve paréntesis para ahondar sobre la máquina de Moore.

©Editorial Digital Tecnológico de Monterrey

El diseño general de una máquina de Moore es el siguiente:

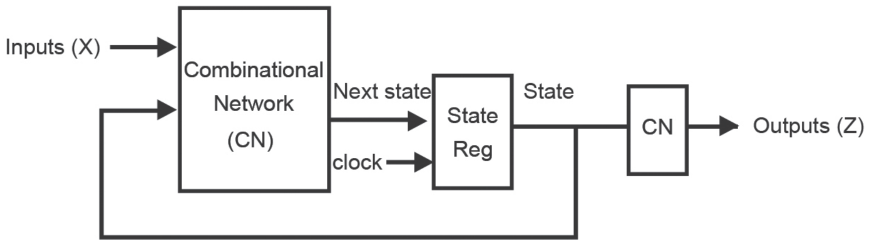


Figura 9.43 Diagrama esquemático general para una máquina de Moore

Hay diversas maneras de describir el circuito combinacional cuya función es calcular el siguiente estado. Resulta bastante cómodo recurrir a estructuras condicionales que permiten describir el comportamiento de la lógica combinacional.

En caso de utilizar estructuras condicionales, el diseño debe asegurar que el registro que contiene al estado actual (State Reg):

a) Se inicialice con el estado adecuado.

b) No se salga del conjunto de estados para los que está diseñado el problema.

Para asegurar una inicialización adecuada, es recomendable que la solución de este o cualquier problema, incluya una señal de *reset* o de *start*, que efectúe la inicialización correcta.

La conversión a VHDL de una máquina de Moore se ilustrará a partir del problema del juego de la adivinanza. A continuación, se muestran descripciones que permiten una buena síntesis del circuito de este juego.

Cabe señalar que la diferencia entre la solución en las ecuaciones es únicamente que los circuitos combinacionales son más simples, ya que en la solución por comportamiento los circuitos se encuentran en forma canónica.

©Editorial Digital Tecnológico de Monterrey

En la descripción por comportamiento basta observar el diagrama de estados, no es necesario recurrir a la tabla de verdad. A continuación, se vuelve a presentar.

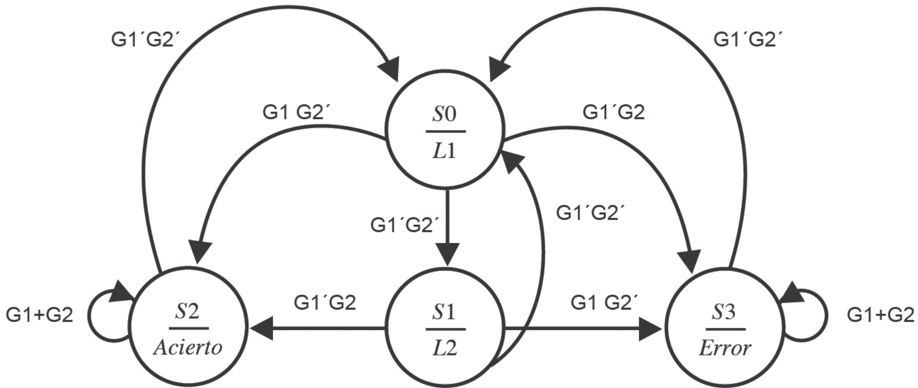


Figura 9.44 Máquina de Moore para juego de la adivinanza con dos entradas

Estados codificados con strings, con salida nextstate modelada con asignación selectiva. La siguiente opción muestra una alternativa para manejar los estados por sus "nombres" utilizando strings para lo anterior. El sistema de desarrollo provee una plantilla para utilizarse en este caso y no tener que recordar todos los detalles del lenguaje.

Lo que hay que adaptar en la plantilla son la cantidad de estados y con qué strings se reconocerán (aquí se dejó S0, S1, etc., que son los nombres que utiliza la plantilla, y las combinaciones en binario que se asociarán a cada estado). No necesariamente deben ir en secuencia, pero sí deben ser únicos, teniendo una combinación distinta para cada estado. Habiendo definido los nombres de estados como strings, ya es posible utilizar estos nombres directamente, es decir, sin comillas.

©Editorial Digital Tecnológico de Monterrey

```
entity juego_ec_string is
  Port ( clk , start : in std_logic;
         g1 : in std_logic;
         g2 : in std_logic;
         l1 : out std_logic;
         l2 : out std_logic;
         acierto : out std_logic;
         error : out std_logic);
end juego_ec_string;
architecture Behavioral of juego_ec_string is
type STATE_TYPE is (S0, S1, S2, S3);
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of STATE_TYPE: type is "00 01 10 11";
signal State, Nextstate: STATE_TYPE;
begin

Nextstate <= S0 when (((State=S1) or (State=S2)or (State=S3)) and (G1='0')
and (G2='0'))
```

```
else S1 when ((State=S0) and (G1='0') and (G2='0'))
         else S2 when (((State=S0) and (G1='1') and (G2='0')) or
           ((State=S1) and (G1='0') and (G2='1')) or
           ((State=S2) and  ((G1='1')or (G2='1'))))
         else S3;
end if;
end process;


process(CLK , start) -- State Register
begin
if start='1' then State<=S0;
elsif CLK ='1' and CLK 'event then -- rising edge of clock
State<=Nextstate;
Else null;
End if;
End process;

l1<='1' when state=S0 else '0';
l2<='1' when state=S1 else '0';
acierto<='1' when state=S2 else '0';
error<='1' when state=S3 else '0';
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Esta opción de codificación puede ser propensa a errores. Otra forma de proceder es la siguiente:

Estados codificados con strings, con salida *nextstate* modelada con estructura condicional *if*.

```
entity prueba5 is
  Port ( clk , start : in std_logic;
       g1 : in std_logic;
       g2 : in std_logic;
       l1 : out std_logic;
       l2 : out std_logic;
       acierto : out std_logic;  error : out std_logic);
end prueba5;

architecture Behavioral of prueba5 is
type STATE_TYPE is (S0, S1, S2, S3);
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of STATE_TYPE: type is "00 01 10 11";
signal State, Nextstate: STATE_TYPE;

begin
process(State,g1,g2) --Combinational Network
begin
case State is
when S0 =>
   if (g1='0') and (g2='0') then Nextstate<=S1;
    elsif (g1='1') and (g2='0')then Nextstate<=S2;
    elsif (g2='1')then Nextstate<=S3;
    else null;
   end if;
```

©Editorial Digital Tecnológico de Monterrey

```
when S1 =>
 if (g1='0') and (g2='0') then Nextstate<=S0;
   elsif (g1='0') and (g2='1')then Nextstate<=S2;
   else Nextstate<=S3;
  end if;
when S2 =>
 if (g1='0') and (g2='0') then Nextstate<=S0;
   else Nextstate<=S2;
  end if;
when S3 =>
 if (g1='0') and (g2='0') then Nextstate<=S0;
   else Nextstate<=S3;
  end if;
when others => Nextstate<=S0;
end case;
end process;

process(CLK , start) -- State Register
begin
if start='1' then State<=S0;  -- inicialización segura
   elsif CLK ='1' and CLK 'event then -- rising edge of clock
   State <= Nextstate;
   Else null;
end if;
end process;

l1<='1' when state=S0 else '0';  --cálculo combinacional de outputs
l2<='1' when state=S1 else '0';
acierto<='1' when state=S2 else '0';
error<='1' when state=S3 else '0';
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Estados codificados con strings, modelando la lógica combinacional dentro de la descripción del registro *state* con estructura *case*. Bajo esta alternativa es posible embeber en un mismo proceso la lógica combinacional con la modelación del registro state, presentando la siguiente codificación:

```
entity prueba5 is
  Port ( clk , start : in std_logic;
      g1 : in std_logic;
      g2 : in std_logic;
      l1 : out std_logic;
      l2 : out std_logic;
      acierto : out std_logic;  error : out std_logic);
end prueba5;


architecture Behavioral of prueba5 is
type STATE_TYPE is (S0, S1, S2, S3);
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of STATE_TYPE: type is "00 01 10 11";
signal State: STATE_TYPE;

begin
process(CLK , start, g1,g2) -- State Register
begin
if start='1' then State<=S0;  -- inicialización segura de la máquina
   elsif CLK ='1' and CLK 'event then -- rising edge of clock
case State is
```

©Editorial Digital Tecnológico de Monterrey

```
when S0 =>
 if (g1='0') and (g2='0') then State<=S1;
   elsif (g1='1') and (g2='0')then State<=S2;
   elsif (g2='1')then State<=S3;
   else null;
  end if;
when S1 =>
 if (g1='0') and (g2='0') then State<=S0;
   elsif (g1='0') and (g2='1')then State<=S2;
   else State<=S3;
  end if;
when S2 =>
if (g1='0') and (g2='0') then State<=S0;
  else State<=S2;
 end if;
when S3 =>
 if (g1='0') and (g2='0') then State<=S0;
   else State<=S3;
 end if;
when others => State<=S0;
end case;
else null;
end if;
end process;

l1<='1' when state=S0 else '0';  --cálculo combinacional de outputs
l2<='1' when state=S1 else '0';
acierto<='1' when state=S2 else '0';
error<='1' when state=S3 else '0';
end Behavioral;
```

Si al aprendiz de VHDL le parece demasiado sofisticada la opción anterior, pareciendo los artificios más parecidos a la programación que a la descripción de *hardware*, a continuación, se presentan alternativas más cercanas a la construcción más básica del circuito. Continúa siendo una descripción de alto nivel, pero más cercana al circuito en cuanto a la descripción del registro state.

©Editorial Digital Tecnológico de Monterrey

Definición de los estados en binario, con señal de "*start*" para inicialización:

```
entity prueba3 is
        Port (clk, start : in std_logic;
             g1 : in std_logic;
             g2 : in std_logic;
             l1 : out std_logic;
             l2 : out std_logic;
             acierto : out std_logic;
             error : out std_logic);
end prueba3;
architecture Behavioral of prueba3 is
signal State, Nextstate: std_logic_vector (1 downto 0);
begin
process(State,g1,g2) --Combinational Network
begin
case State is
when "00" =>
if (g1='0') and (g2='0') then Nextstate<="01";
        elsif (g1='1') and (g2='0') then Nextstate<="10";
        elsif (g2='1') then Nextstate<="11";
        else null;
end if;
when "01" =>
if (g1='0') and (g2='0') then Nextstate<="00";
        elsif (g1='0') and (g2='1') then Nextstate<="10";
        else Nextstate<="11";;
end if;
when "10" =>
if (g1='0') and (g2='0') then Nextstate<="00";
        else Nextstate<="10";
end if;
when "11" =>
if (g1='0') and (g2='0') then Nextstate<="00";
        else Nextstate<="11";
```

©Editorial Digital Tecnológico de Monterrey

```
end if;
 when others => Nextstate<="00";  --esta inicialización no es segura porque la opción
 ----when others no tiene una implementación correcta
 else null;
 end if;
 end process;
 process (CLK , start) -- State Register
 begin
 if start='1' then State<=S0;  -- inicialización segura
elsif CLK ='1' and CLK 'event then -- rising edge of clock
State <= Nextstate;
Else null;
 end if;
 end process;
 l1<='1' when state="00" else '0';  --cálculo combinacional de outputs
l2<='1' when state="01" else '0';
 acierto<='1' when state="10" else '0';
 error<='1' when state="11" else '0';
 end Behavioral;
```

Codificación de los estados en binario, con señal de "*start*" para inicialización, describiendo la lógica combinacional dentro de la modelación del registro state

```
entity prueba3 is
         Port (clk, start : in std_logic;
               g1  : in std_logic;
               g2 : in std_logic;
               l1  : out std_logic;
               l2 : out std_logic;
               acierto : out std_logic;
               error : out std_logic);
end prueba3;
```

©Editorial Digital Tecnológico de Monterrey

```
architecture Behavioral of prueba3 is
signal State: std_logic_vector (1 downto 0);
begin
process(CLK , start, g1,g2) – Descripción del State Register
begin
if start='1' then State<="00"   --inicialización del registro de estado con la señal de start
elsif CLK ='1' and CLK 'event then
case State is
when "00" =>
if (g1='0') and (g2='0') then State<="01";
        elsif (g1='1') and (g2='0') then State<="10";
        elsif (g2='1') then State<="11";
        else null;
end if;

when "01" =>
if (g1='0') and (g2='0') then State<="00";
        elsif (g1='0') and (g2='1') then State<="10";
        else State<="11";
end if;
when "10" =>
if (g1='0') and (g2='0') then State<="00";
        else State<="10";
end if;
when "11" =>
if (g1='0') and (g2='0') then State<="00";
        else State<="11";
end if;
when others => State<="00";
end case;
else null;
end if;
end process;

l1<='1' when state="00" else '0'; --cálculo combinacional de outputs
```

©Editorial Digital Tecnológico de Monterrey

```
l2<='1' when state="01" else '0';
acierto<='1' when state="10" else '0';
error<='1' when state="11" else '0';
end Behavioral;
```

Estados codificados en tipo entero, con señal de "*start*":

```
entity juego is
        Port (clk, start: in std_logic;
              g1 : in std_logic;
              g2 : in std_logic;
              l1 : out std_logic;
              l2 : out std_logic;
              acierto : out std_logic;
              error : out std_logic);
end juego;
architecture Behavioral of juego is
signal State, Nextstate: integer;
begin
process(State,g1,g2) --Combinational Network
begin
case State is
when 0 =>
if (g1='0') and (g2='0') then Nextstate<=1;
        elsif (g1='1') and (g2='0') then Nextstate<=2;
        elsif (g2='1') then Nextstate<=3;
        else null;
end if;
when 1 =>
if (g1='0') and (g2='0') then Nextstate<=0;
        elsif (g1='0') and (g2='1') then Nextstate<=2;
        else Nextstate<=3;
end if;
when 2 =>
if (g1='0') and (g2='0') then Nextstate<=0;
        else Nextstate<=2;
```

©Editorial Digital Tecnológico de Monterrey

```
end if;
 when 3 =>
if (g1='0') and (g2='0') then Nextstate<=0;
          else Nextstate<=3;
end if;
 when others => Nextstate<=0;
 end case;
 end process;

 process(CLK , start) -- State Register
 begin
 if start='1' then State<=0;
 elsif CLK ='1' and CLK 'event then -- rising edge of clock
 State <= Nextstate;
 end if;
 end process;

 l1<='1' when state=0 else '0';
 l2<='1' when state=1 else '0';
 acierto<='1' when state=2 else '0';
 error<='1' when state=3 else '0';
 end Behavioral;
```

Estados codificados en tipo entero, con señal de "*start*" y describiendo la lógica combinacional junto con la descripción del registro state

```
entity juego is
          Port (clk , start: in std_logic;
               g1 : in std_logic;
               g2 : in std_logic;
               l1 : out std_logic;
               l2 : out std_logic;
               acierto : out std_logic;
               error : out std_logic);
```

©Editorial Digital Tecnológico de Monterrey

```
end juego;

architecture Behavioral of juego is
signal State: integer;
begin
process (CLK , start, g1,g2) -- State Register
begin
if start='1' then State<=0;
elsif CLK ='1' and CLK 'event then
case State is
when 0 =>
if (g1='0') and (g2='0') then State<=1;
        elsif (g1='1') and (g2='0') then State<=2;
        elsif (g2='1') then State<=3;
        else null;
end if;
when 1 =>
if (g1='0') and (g2='0') then State<=0;
        elsif (g1='0') and (g2='1') then State<=2;
        else State<=3;
end if;
when 2 =>
if (g1='0') and (g2='0') then State<=0;
        else State<=2;
end if;
when 3 =>
if (g1='0') and (g2='0') then State<=0;
        else State<=3;
end if;
when others => State<=0;
end case;
else null;
end if;
end process;
```

©Editorial Digital Tecnológico de Monterrey

```
l1<='1' when state=0 else '0';
l2<='1' when state=1 else '0';
acierto<='1' when state=2 else '0';
error<='1' when state=3 else '0';
end Behavioral;
```

9.2.6 Ejemplos de alternativas de codificación de una máquina de Moore

Comparador de dos números seriales de longitud indefinida

En el siguiente problema se le solicita diseñar un circuito que compare las magnitudes de dos números X y Y que ingresan serialmente a partir de su *bit* más significativo, con una frecuencia determinada por una señal de reloj Clk que también es entrada del circuito. La longitud de los números está indeterminada (es decir, la cantidad de *bits* que los conforman), así que una vez que el número formado por la secuencia de *bits* X sea mayor que el número formado por la secuencia Y, la salida es 1. Conforme los *bits* X y Y ingresan al circuito la salida Z va indicando, con 1, si el número formado por los *bits* X es mayor que el número formado por los *bits* Y, 0 de otra manera. El circuito también debe contar con una señal de reset que indique que se alimentará una nueva secuencia de *bits* de X y Y.

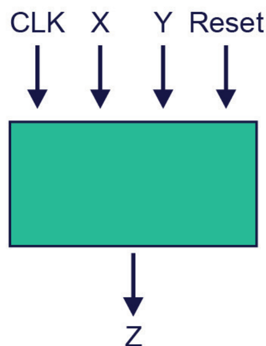


Figura 9.45 Puerto del circuito comparador de números seriales

©Editorial Digital Tecnológico de Monterrey

Si usted analiza la manera en que se detecta cuando un número es mayor que otro, el procedimiento consiste en comparar ambos números, *bit* a *bit*, de posición más significativa a menos, es decir, de izquierda a derecha; el primer *bit* que sea mayor o menor hace que un número sea mayor o menor.

Por ejemplo:

```
X     1000001000001 …
Y     1000000111111 …
Z     0000001111111 …
```

Mientras X = Y la salida es Z = 0. En el momento que X = 1 y Y = 0, Z = 1 y continúa en 1 sin importar las siguientes entradas X, Y.

Otro ejemplo:

```
X     1000000000001 …
Y     1000001000111 …
Z     0000000000000 …
```

Mientras X = Y la salida es Z = 0. En el momento que X = 0 y Y = 1, Z = 0 y continúa en 0 sin importar las siguientes entradas X, Y.

Último caso:

```
X     1000001000111 …
Y     1000001000111 …
Z     0000000000000 …
```

©Editorial Digital Tecnológico de Monterrey

Mientras X = Y la salida es Z = 0.

Dos propuestas de diseño con máquina de Mealy, una con expresiones booleanas y otra con expresiones relacionales. También sería posible un diseño con una mezcla de diferentes tipos de expresiones.

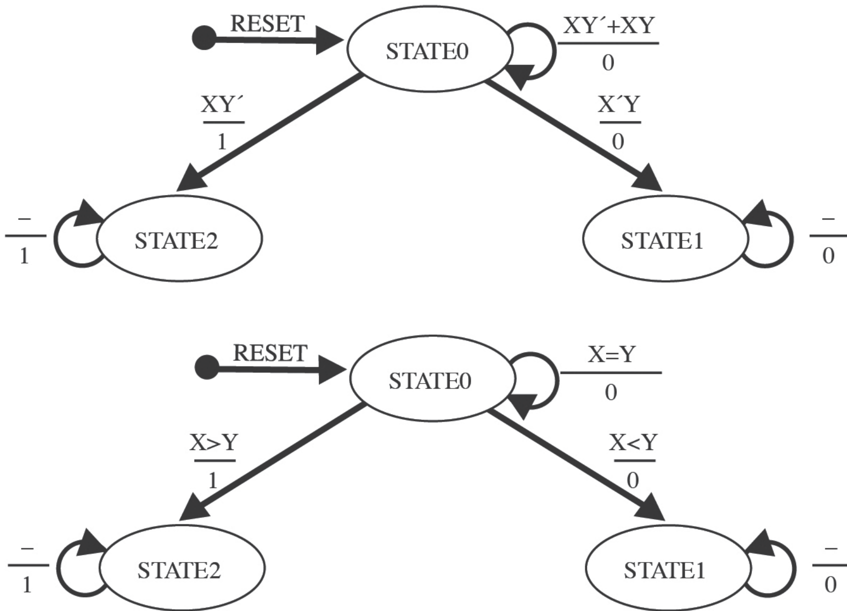


Figura 9.46

Ambos estilos de diseño son válidos y equivalentes. Aunque, si prefiere utilizar constantes, el diseño del diagrama es el siguiente:

©Editorial Digital Tecnológico de Monterrey

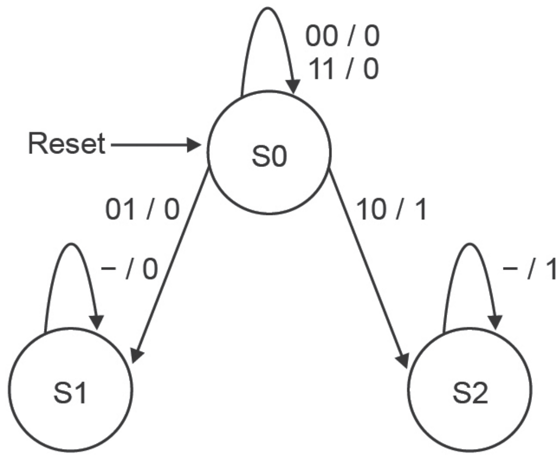


Figura 9.47 Máquina de Mealy para comparador de números seriales

La codificación para el primer estilo es el siguiente:

```
entity problema2 is
  port(x,y,reset,clk : in std_logic;
          z: out std_logic
          );
end problema2;

architecture Behavioral of problema2 is
signal state: integer:=0;

begin

     process(x,y, state)
          begin
                              case state is
                    when 0 => if x=y then
                                   z<='0';
                                   NextState <=0;
                                   elsif (x='1') and (y='0') then
                                   z<='1';
                                   NextState<=1;
                              elsif (x='0') and (y='1') then
                                   z<='0';
                                   NextState <=2;
                              else null;
                              end if;
                    when 1 =>  z<='1';
                              NextState <=1;
                    when 2 =>      z<='0';
                              NextState <=2;
                    when others => null;
               end case;
          end process;
```

©Editorial Digital Tecnológico de Monterrey

```
process(clk , reset)
begin
if reset='1' then state<='0';
elsif clk ='1' and clk 'event then
     state<=NextState;
else null;
end if;
end process;
end Behavioral;
```

Para este problema sí es adecuado una máquina de Moore, cuyo diseño es el siguiente (utilizando el estilo de expresiones relacionales):

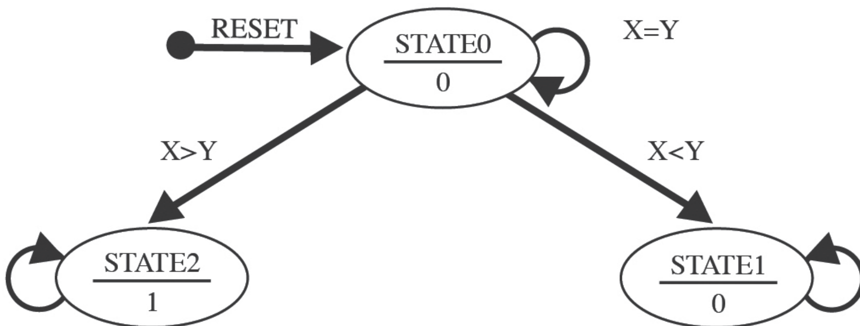


Figura 9.48 Máquina de Moore para comparador de números seriales

©Editorial Digital Tecnológico de Monterrey

Funcionalmente este diseño es equivalente.

```
entity problema2 is
  port(x,y,reset,clk : in std_logic;
          z: out std_logic
          );
end problema2;

architecture Behavioral of problema2 is
signal state: integer:=0;

begin

  process(x,y, state)
       begin
                                 case state is
                        when 0 => if x=y then
                                          NextState <=0;
                                          elsif x>y then
                                          NextState<=1;
                                     elsif x<y then
                                          NextState <=2;
                                     else null;
                                     end if;
                        when 1 =>    NextState <=1;
                        when 2 =>    NextState <=2;
                        when others => null;
                   end case;
       end process;
```

```
process(clk , reset)
begin
if reset=’1’ then state<=’0’;
elsif clk ='1' and clk 'event then
   state<=NextState;
else null;
end if;
end process;
z<=’1’ when state=2 else ‘0’;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Otra alternativa para describir este circuito es mezclar el diseño del circuito combinacional con la del registro state, es decir, modelar el registro state describiendo su funcionalidad en solo un proceso.

```
entity problema2 is
  port(x,y,reset,clk : in std_logic;
         z: out std_logic
         );
end problema2;

architecture Behavioral of problema2 is
signal state: integer:=0;

begin

  process(clk , reset)
  begin
  if reset='1' then state <='0';
```

```
elsif clk ='1' and clk 'event then
                case state is
                        when 0 => if x=y then
                                                state <=0;
                                          elsif (x='1') and (y='0') then
                                                 -- o bien x<y
                                                state<=1;
                                          elsif (x='0') and (y='1') then
                                                -- o bien x<y
                                                state <=2;
                                          else null;
                                          end if;
                        when 1 => state <=1;
                        when 2 =>      state <=2;
                        when others => null;
                end case;
      else null;
      end if;
      end process;
z<='1' when state=2 else '0';
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Encriptado I

Este problema propone el encriptado de una cadena serial de *bits*. La salida encriptada ocurre en forma serial sincronizada con un reloj externo.

El código de encriptación consiste en lo siguiente: la salida será igual a la entrada hasta que ocurra la secuencia 0110, el siguiente *bit* de esta secuencia deberá generarse negando la entrada, luego cuando ocurra la secuencia 111 la salida deberá ser de nuevo la entrada.

```
Para la entrada:     110011010101000111010110101 0...
La salida sería:     1100110010101110000101100101
```

©Editorial Digital Tecnológico de Monterrey

La máquina de Mealy que resuelve el problema es:

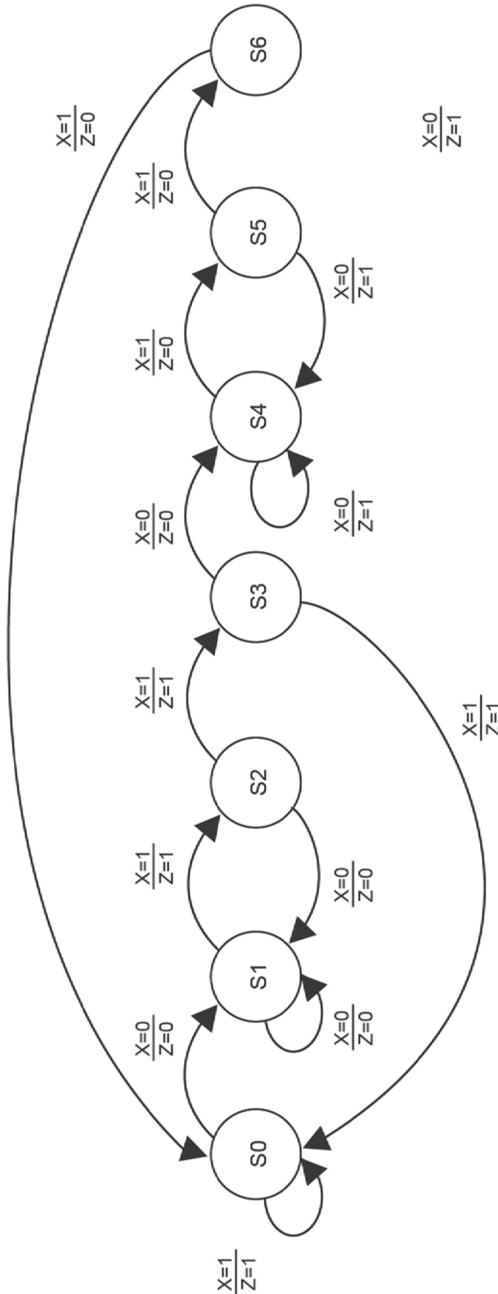


Figura 9.49 Máquina de Moore para encriptado de números seriales

©Editorial Digital Tecnológico de Monterrey

Y su descripción, retrasando la salida hasta la siguiente transición del reloj, es la siguiente:

```
entity automatas is
   port (x,clk : in std_logic;
                  z: out std_logic);
end automatas;

architecture Behavioral of automatas is

signal state: integer:=0;
begin

     process(clk ,x,state)
          begin
               if clk ='1' and clk 'event then
                        case state is
                        when 0 => if x='0' then
                                    state<=1;
                                    else
                                    state<=0;
                                    end if;
                                    z<= x;
                        when 1 => if  x='1' then
                                    state<=2;
                                    else
                                    state<=1;
                                    end if;
                                    z<= x;
```

©Editorial Digital Tecnológico de Monterrey

```
                    when 3 => if x='0' then
                                   state<=4;
                                       else
                                       state<=0;
                                       end if;
                                       z<=x;
                    when 4 => if x='1' then
                                       state<=5;
                                       else
                                       state<=4;
                                       end if;
                                       z<= not x;
                    when 5 => if x='1' then
                                       state<=6;
                                       else
                                       state<=4;
                                       end if;
                                       z<= not x;
                    when 6 => if x='1' then
                                       state<=0;
                                       else
                                       state<=4;
                                       end if;
                                       z<= not x;
                     when others => null;
                end case;
            end if;
        end process;
end Behavioral;
```

Tablero para juego de tenis

Cierto juego (que utiliza una puntuación similar a la del tenis) se juega con dos participantes. La secuencia de puntos que gana cada jugador al ganar una partida se muestra a continuación:

0, 15, 40 "gana"

Sin embargo, si empatan a 40, el siguiente punto es "ventaja" y hasta el siguiente es "gana", pero si empatan en "ventaja" a ambos se les regresa a 40, teniendo que anotar dos jugadas seguidas para ganar.

©Editorial Digital Tecnológico de Monterrey

Las entradas serían:

- Botones para registrar los puntos de cada jugador: J1 y J2
- Botón para borrar puntuación de ambos jugadores e indicadores: *reset*
- Reloj de 1 MHz: Clk

Una secuencia de puntos que podría ocurrir es:

Puntos de:	Jugador 1			Jugador 2		
	Puntos	Ventaja	Gana	Puntos	Ventaja	Gana
	0	0	0	0	0	0
J1	15	0	0	0	0	0
J1	40	0	0	0	0	0
J2	40	0	0	15	0	0
J2	40	0	0	40	0	0
J1	40	1	0	40	0	0
J2	40	0	0	40	0	0
J2	40	0	0	40	1	0
J2	40	0	0	40	1	0
J2	40	0	0	40	1	1

Tabla 9.7

Las salidas son:

- Puntuaciones de cada jugador (en binario, 5 *bits*): Puntos1 (P1) y Puntos2 (P2).
- Indicadores de ventaja, uno para cada jugador: Ventaja1(-Va1) y Ventaja2 (Va2).

©Editorial Digital Tecnológico de Monterrey

• Indicadores para indicar si un jugador gana el set: G1, G2.

Se diseñará el circuito que permita registrar los puntos en binario y que indique si un jugador gana el set.

Se requieren señales auxiliares porque se utilizan como entradas a comparadores, para la lógica combinacional requerida para el cambio de estado.

Por otra parte, como J1 y J2 son señales que provienen de botones, habrá que agregar un estado (como S1 y S2) para esperar a que el botón se suelte y la señal regrese a 0, sin afectar los puntajes.

El diseño resulta en una máquina de Mealy, porque el diseño en una máquina de Moore requeriría un mayor número de estados.

©Editorial Digital Tecnológico de Monterrey

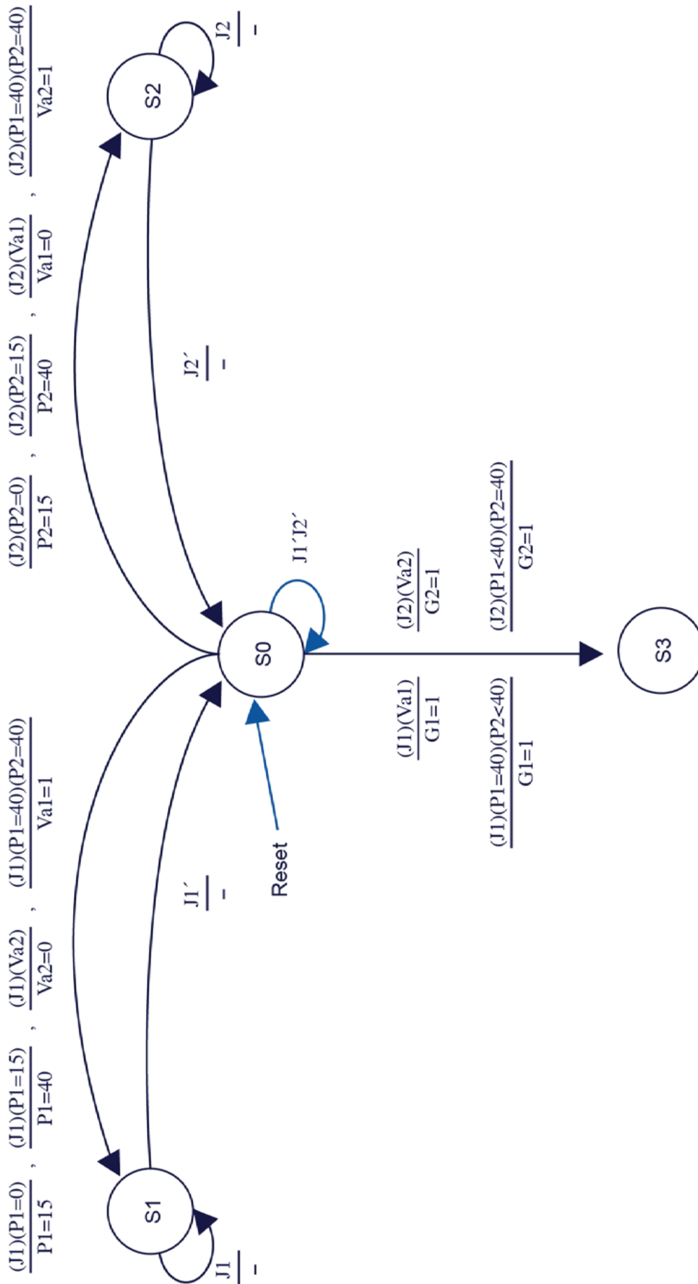


Figura 9.50 Máquina de Mealy para tablero de tenis

©Editorial Digital Tecnológico de Monterrey

Sin embargo, en este circuito no resulta crítico el momento en que se calculan las salidas, si es en el pulso de reloj del siguiente estado o si es justo cuando ocurre el punto, si el reloj tiene una frecuencia del orden de kilo o mega hertz es realmente imperceptible. Así que se eligió una codificación que combina los conceptos de máquina de Mealy y Moore. No resulta posible definirlo en el dibujo, pero sí en VHDL, resultando una codificación muy simple que se presenta a continuación.

```
entity problema5 is
  port ( J1,J2,Reset,clk : in std_logic;
              Puntos1,Puntos2: out std_logic_vector(5 downto 0);
              Ventaja1,Ventaja2,G1,G2: out std_logic);
end problema5;

architecture Behavioral of problema5 is
signal state: integer;
signal P1,P2: std_logic_vector(5 downto 0);
signal Va1,Va2;
begin
  process(clk ,state,J1,J2,reset)
       begin
             if reset ='1' then
                   P1<="000000";
                   P2<="000000";
                   Va1<='0';
                   Va2<='0';
                   G1<='0';
                   G2<='0';
                   state<=0;
             elsif clk ='1' and clk 'event then
                   case state is
                   when 0 =>if J1='1' then
                               if (Va1='1' or (P1=40 and P2<40)) then
                                    G1<='1';
                                    state<=3;
                               elsif P1=0 then
                                    P1<="001111";
                                    state<=1;
                               elsif P1=15 then
                                    P1<="101000";
                                    state<=1;
```

©Editorial Digital Tecnológico de Monterrey

```
                        elsif Va2='1' then
                                Va2<='0';
                                state<=1;
                        elsif P1=40 and P2=40 then
                                Va1<='1';
                                state<=1;
                        end if;
                  elsif J2='1' then
                        if (Va2='1' or (P1<40 and P2=40)) then
                                G2<='1';
                                state<=3;
                        elsif P2=0 then
                                P2<="001111";
                                state<=2;
                        elsif P2=15 then
                                P2<="101000";
                                state<=2;
                        elsif Va1='1' then
                                Va1<='0';
                                state<=2;
                        elsif P1=40 and P2=40 then
                                Va2<='1';
                                state<=2;
                        end if;
               end if;
when 1 =>if J1='1' then
                       state<=1;
                 else
                          state<=0;
                 end if;
```

```
when 2 =>if J2='1' then
                                state<=2;
                         else
                                state<=0;
                         end if;
          when others =>
          end case;
          end if;
  end process;
Puntos1<=P1;
Puntos2<=P2;
Ventaja1<=Va1;
Ventaja2<=Va2;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Sin embargo, si se desea describir la máquina "pura" de Mealy, la descripción se presenta a continuación:

```
entity problema5 is
     port ( J1,J2,Reset,clk : in std_logic;
                Puntos1,Puntos2: out std_logic_vector(5 downto 0);
                Ventaja1,Ventaja2,G1,G2: out std_logic);
end problema5;

architecture Behavioral of problema5 is
signal state, nextstate: integer;
signal P1,P2: std_logic_vector(5 downto 0);
signal Va1,Va2;
begin
```

```
process(state,J1,J2,reset)
  begin
       case state is
       when 0 =>if J1='1' then
                          if (Va1='1' or (P1=40 and P2<40)) then
                                 G1<='1';
                                 nextstate<=3;
                          elsif P1=0 then
                                 P1<="001111";
                                 nextstate<=1;
                          elsif P1=15 then
                                 P1<="101000";
                                 nextstate<=1;
     elsif Va2='1' then
                                 Va2<='0';
                                 nextstate<=1;
                          elsif P1=40 and P2=40 then
                                 Va1<='1';
                                 nextstate<=1;
                          end if;
                     elsif J2='1' then
                          if (Va2='1' or (P1<40 and P2=40)) then
                                 G2<='1';
                                 nextstate<=3;
                          elsif P2=0 then
                                P2<="001111";
                                nextstate<=2;
                          elsif P2=15 then
                                P2<="101000";
                                nextstate<=2;
                          elsif Va1='1' then
                                Va1<='0';
                                nextstate<=2;
```

©Editorial Digital Tecnológico de Monterrey

```
                        elsif P1=40 and P2=40 then
                                 Va2<='1';
                                 nextstate<=2;
                            end if;
                       end if;
     when 1 =>if J1='1' then
                            nextstate<=1;
                       else
                            nextstate<=0;
                       end if;
     when 2 =>if J2='1' then
                            nextstate<=2;
                       else
                            nextstate<=0;
                       end if;
    when others =>
    end case;
end if;
end process;
process(clk ,reset)
if reset ='1' then
                    state<=0;
             elsif clk ='1' and clk 'event then
state<=nextstate;
            else null;
end if;
end process;
Puntos1<=P1;
Puntos2<=P2;
Ventaja1<=Va1;
Ventaja2<=Va2;
end Behavioral;
```

Motor de pasos

Este problema está basado en un problema que se presenta en el libro “Sistemas digitales y electrónica digital, prácticas de laboratorio”, del M.C. Juan Ángel Garza.

Se desea diseñar un circuito para controlar un motor de pasos, específicamente de cuatro pasos. Un motor de pasos puede accionar dispositivos con gran precisión.

©Editorial Digital Tecnológico de Monterrey

Figura 9.51 Motor de pasos

Para la aplicación que se busca desarrollar, lo que el circuito deberá hacer es: al recibir una señal de *Start* generar por una salida Q1 un 1 durante 1mseg, a continuación, generar por una salida Q2 un 1 durante 1mseg, luego generar por la salida Q3 un 1 durante 1mseg, luego lo mismo por Q4, un 1 durante 1mseg, luego se repite el ciclo continuando por Q1.

Las entradas al circuito son:

- Clk 1M: una señal de reloj de 1MHz.
- *Start*: señal con la que se inicia la operación del controlador del motor.
- *Reset*: hace que el circuito genere ceros como salidas y de nuevo espere la señal de *start* para iniciar la secuencia.

En forma muy general el diseño esquemático del circuito es:

©Editorial Digital Tecnológico de Monterrey

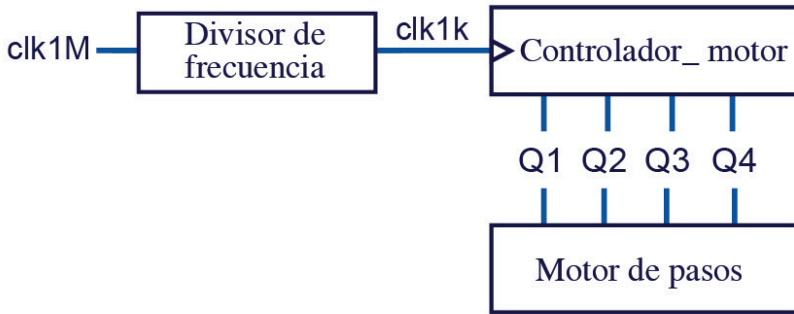


Figura 9.52 Diseño esquemático para controlador de motor de pasos

Como el reloj con el que se cuenta tiene una frecuencia de 1MHz, se requiere un divisor de frecuencias para lograr una frecuencia de 1KHz y así contar con un período de:

$$\text{periodo} = \frac{1 \text{ seg}}{1000 \text{ ciclos}} = \frac{10^{-3} \text{ seg}}{\text{ciclo}} = 1\text{mseg}$$

La máquina de estados se presenta en dos estilos, como máquina de Mealy:

©Editorial Digital Tecnológico de Monterrey

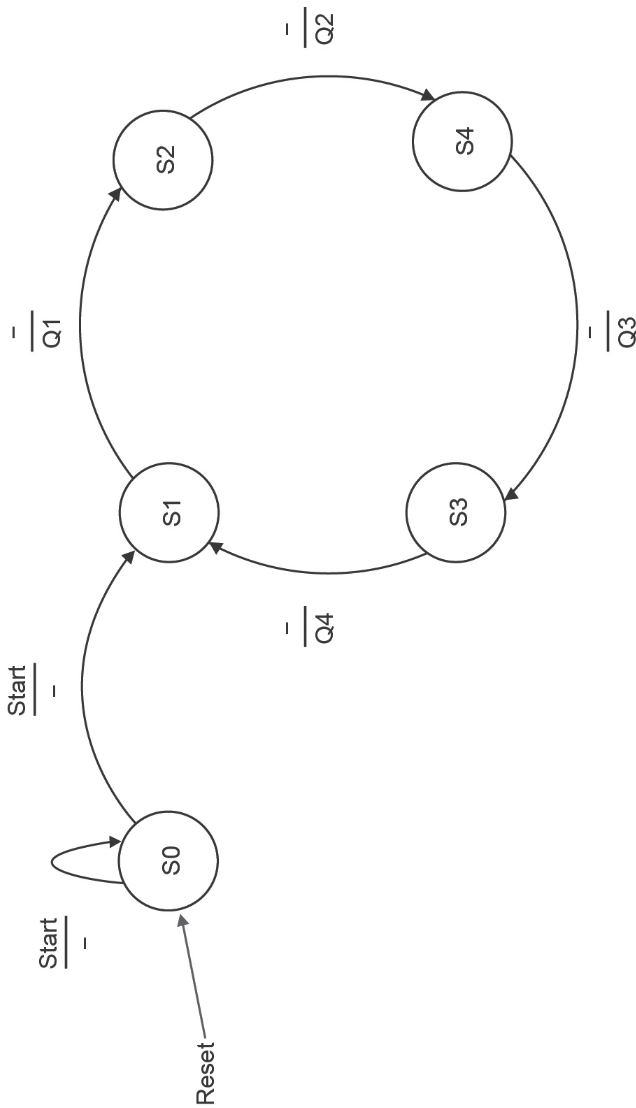


Figura 9.53

La codificación en VHDL que se presenta corresponde a la máquina de Moore porque las salidas se están actualizando con las transiciones del reloj. La descripción de las salidas se incluyó en el proceso que describe la lógica del siguiente estado.

©Editorial Digital Tecnológico de Monterrey

```
entity controlador_motor is
     port(clk 1M,start: in std_logic;
            Q1,Q2,Q3,Q4: out std_logic);
end controlador_motor;

architecture Behavioral of controlador_motor is
signal state,contclk : integer:=0;
signal clk 1K: std_logic;
begin

   process(clk 1M,clk 1K,contclk )
   begin
        if contclk =500 then
               clk 1K<= not clk 1K;
               contclk <=0;
        elsif clk 1M='1' and clk 1M'event then
               contclk <=contclk +1;
          end if;
     end process;

     process(clk 1K,state)
          begin
               if clk 1K='1' and clk 1K'event then
                     case state is
                            when 0=> if start='0' then
                                                 state<=0;
                                            else
                                                 state<=1;
                                            end if;
```

©Editorial Digital Tecnológico de Monterrey

```
                              when 1=>  Q1<= '1';
                                        Q2<='0';
                                        Q3<='0';
                                        Q4<='0';
                                        state<=2;
                              when 2=>  Q1<= '0';
                                        Q2<='1';
                                        Q3<='0';
                                        Q4<='0';
                                        state<=3;

                              when 3=>  Q1<= '0';
                                        Q2<='0';
                                        Q3<='1';
                                        Q4<='0';
                                        state<=4;

                              when 4=>  Q1<= '0';
                                        Q2<='0';
                                        Q3<='0';
                                        Q4<='1';
                                        state<=1;
                              when others=> null;
                              end case;
                      end if;
              end process;
end Behavioral;
```

Como se ha observado en estos ejemplos, una vez diseñado el diagrama de estados, la conversión a un circuito siempre se realiza siguiendo los mismos pasos. En los siguientes problemas se muestra solamente el diagrama de estados y se omite el resto de su solución, ya que en el diseño del diagrama reside la solución del problema.

Complementador con bit de stop

En este ejemplo se presenta una máquina de Mealy que calcula el complemento a 2 de un número de cualquier longitud que ingresa serialmente a partir del *bit* menos significativo y genera una respuesta también serial.

La máquina recibe dos entradas, el *bit* serial X y un *bit* de *stop*, que mientras es 0, indica que ha ingresado un *bit* que forma parte del número al cual se le está calculando su complemento, y cuando

©Editorial Digital Tecnológico de Monterrey

stop = 1, la máquina se dirige a un estado final de paro y ya no importa el valor del *bit* de X que ingrese, y tampoco importa la salida. Cuando *stop* cambia a 0 o permanece en 0, el cálculo del complemento vuelve a iniciar.

La solución, que presenta expresiones booleanas para las entradas y constantes para las salidas, es:

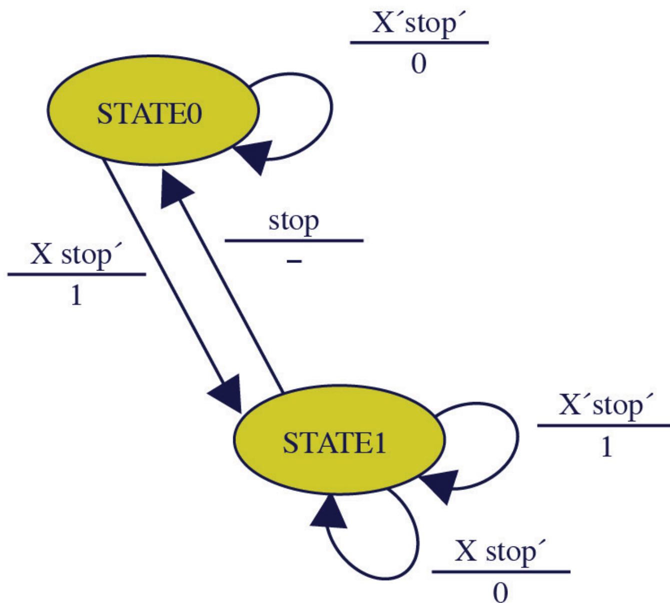


Figura 9.54

Encriptado II

A continuación, se explica cómo diseñar una máquina de estados que encripte la entrada serial que reciba. La máquina recibe un *bit* X y genera un *bit* Z. Al inicio la salida es la entrada negada Z = X', cuando se reconoce en la entrada la secuencia 101, después del segundo 1 la salida deja de ser la entrada complementada y se convierte en Z = X. A continuación, cuando se reconoce en la entrada la secuencia 110, la salida vuelve a ser la entrada complementada

©Editorial Digital Tecnológico de Monterrey

después del 0 y así continúa. Tómese en cuenta que el último 1 de la secuencia 101 puede representar el primer 1 de 110. Luego, si vuelve a ocurrir un 101 se deja de complementar la entrada para producir la salida (la secuencia 110 seguida de un 1 completa el string 101) y luego, después de 110 se complementa la entrada y así sucesivamente.

Ejemplo:

```
Entrada :       0011010011010100010110111
Salida :        1100100011000100010110011
```

El puerto del circuito es:

```
entity encripta is
    Port (x : in  STD_LOGIC;
        clk  : in  STD_LOGIC;
        z : out  STD_LOGIC);
end encripta;
```

En el diseño se utilizan constantes para entradas y salidas.

©Editorial Digital Tecnológico de Monterrey

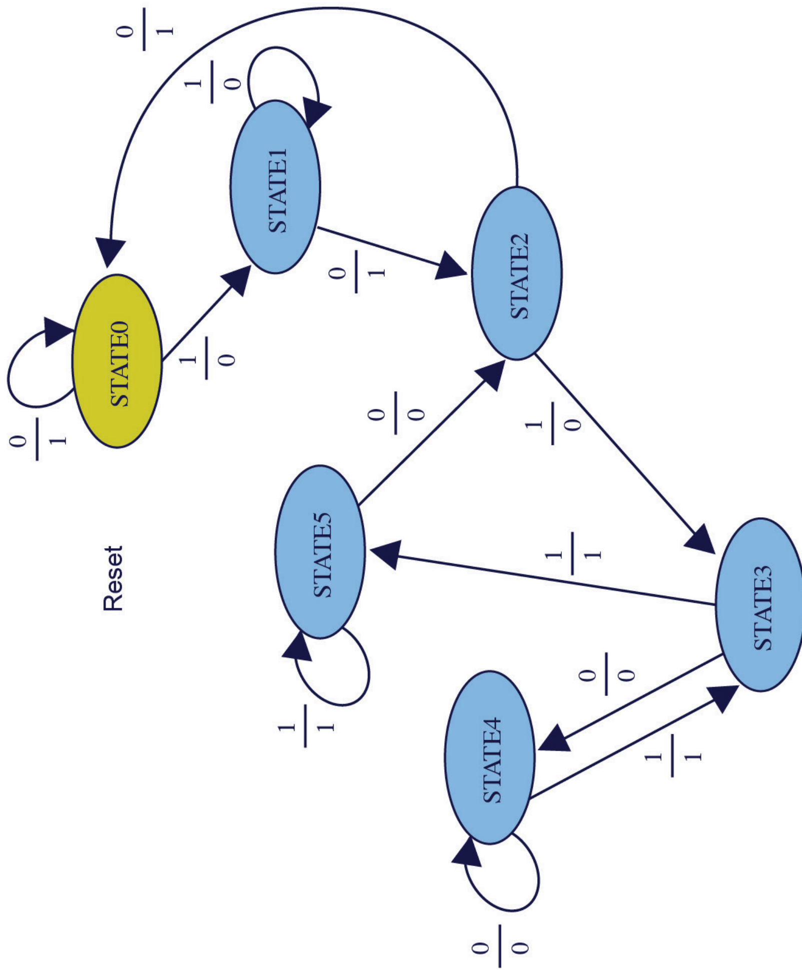


Figura 9.55 Máquina de Mealy para encriptador

Luces decorativas

En este problema se propone diseñar un circuito que controle una pequeña matriz de luces, configurada de la siguiente manera:

©Editorial Digital Tecnológico de Monterrey

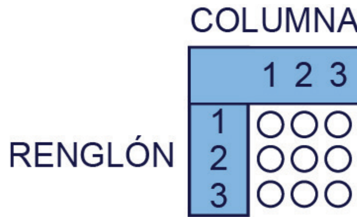


Figura 9.56 Matriz de luces

Se cuenta con dos botones: un botón “D” para encender las luces hacia la derecha por columnas completas. El otro botón sería “A”, para encender las luces hacia abajo por renglones completos. La siguiente figura lo ilustra:

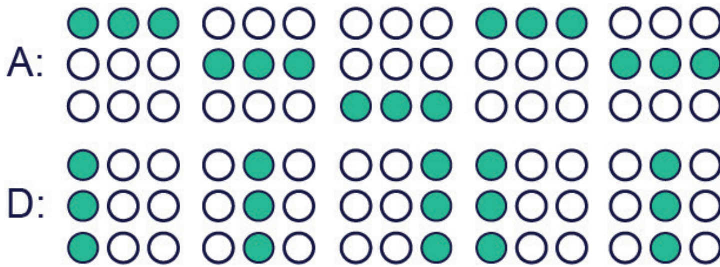


Figura 9.57 Patrón de iluminación para matriz de luces

Si al iluminar las luces hacia abajo se encuentra encendido el renglón i y se oprime el botón “D”, la secuencia cambiaría a iluminar la columna i+1 (considere que después de 3 sigue el 1 en carácter rotativo). Ahora, si se está iluminando la columna i y se oprime el botón “A”, la secuencia cambia a iluminar el renglón i+1 (de la misma manera, después de 3 sigue 1).

Los botones solo se tienen que pulsar sin tener que dejarlos oprimidos. Si al estar iluminando hacia abajo se vuelve a oprimir “A” se reinicia la secuencia volviendo a encender el primer renglón. Si esto sucede con la iluminación hacia la derecha se reinicia con la columna 1.

Una opción de configuración consiste en definir un vector de nueve posiciones, en el que la posición 1 represente el renglón 1, columna 1; la posición 2 sea el renglón 1, columna 2; y la posición 9 corresponda al renglón 3, columna 3.

©Editorial Digital Tecnológico de Monterrey

Un diseño por máquina de Moore es:

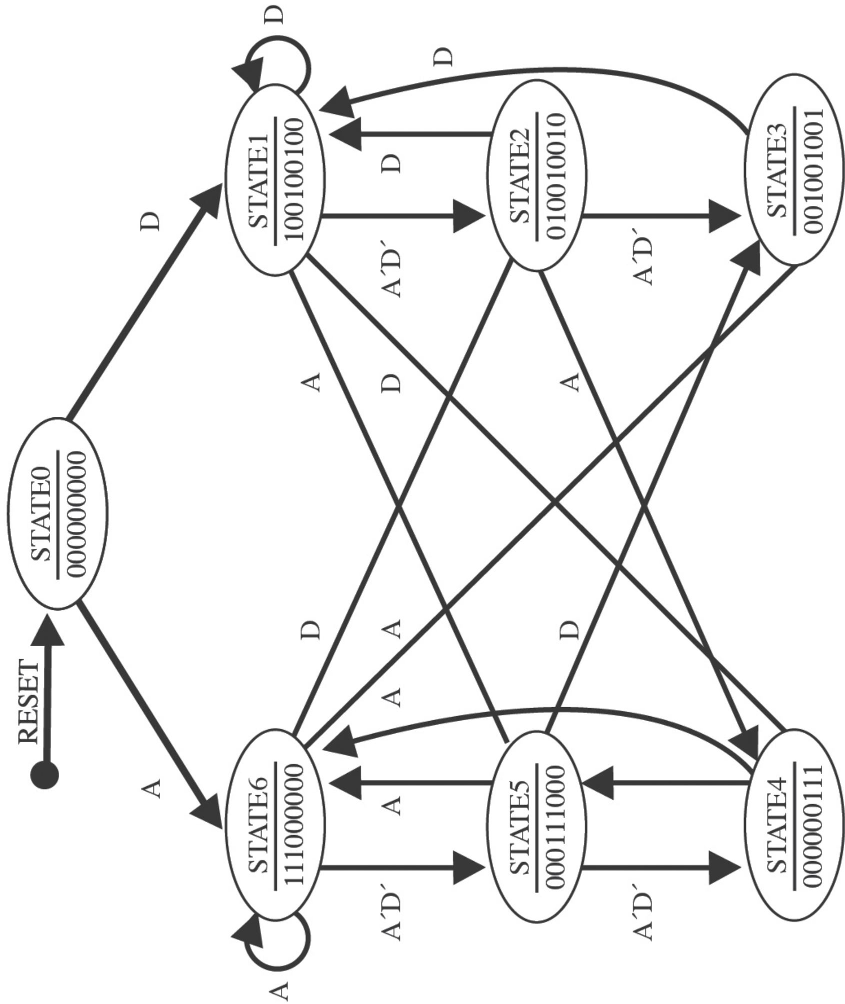


Figura 9.58 Máquina de Moore para seguir el patrón de iluminación para matriz de luces

Otra opción para este mismo problema consiste en una máquina de Mealy. En esta opción de diseño, un "-" en la entrada representa A'D', es decir 00, y un "-" en la salida corresponde a 000000000. Para no trazar líneas cruzadas se repitieron dibujos de estados.

©Editorial Digital Tecnológico de Monterrey

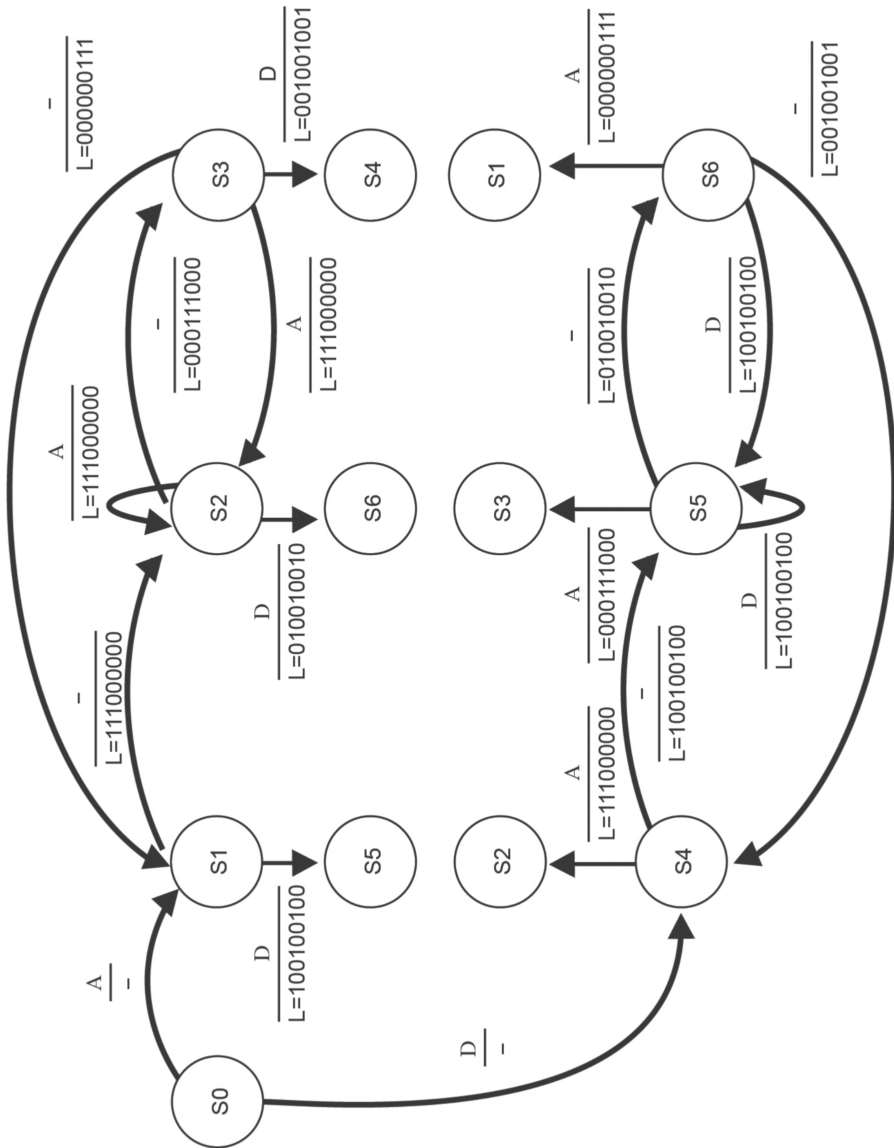


Figura 9.59 Máquina de Moore para seguir el patrón de iluminación para matriz de luces

©Editorial Digital Tecnológico de Monterrey

Barreras de seguridad de una vía de ferrocarril

Este problema y su gráfica está basado en un problema que se presenta en el libro "Sistemas digitales y electrónica digital, prácticas de laboratorio", del M.C. Juan Ángel Garza.

Una vía de ferrocarril, con tráfico en ambos sentidos, corta una carretera en donde se colocan barreras activadas por una salida Z a través de un sistema secuencial (ver figura inferior).

A 500 m del punto de cruce se colocan detectores (X1 y X2, respectivamente) en ambas direcciones.

El estado inicial es Z = 0, este valor debe pasar al estado 1 (Z = 1) cuando se acerca un ferrocarril en cualquier sentido, el cual, a los 500 m del cruce debe volver al estado cero; lo anterior sucede justo cuando el último vagón se aleja más de dicha distancia, independientemente de la longitud del ferrocarril. Considere que no está permitido hacer maniobras, es decir, no hay cambio de dirección, y que solo pasa un tren a la vez.

Casos que pueden ocurrir:

a) Tren corto de X1 a X2 (tren menor a 1,000 metros).

b) Tren corto de X2 a X1 (tren menor a 1,000 metros).

c) Tren largo de X1 a X2 (tren mayor a 1,000 metros).

d) Tren largo de X2 a X1 (tren mayor a 1,000 metros).

©Editorial Digital Tecnológico de Monterrey

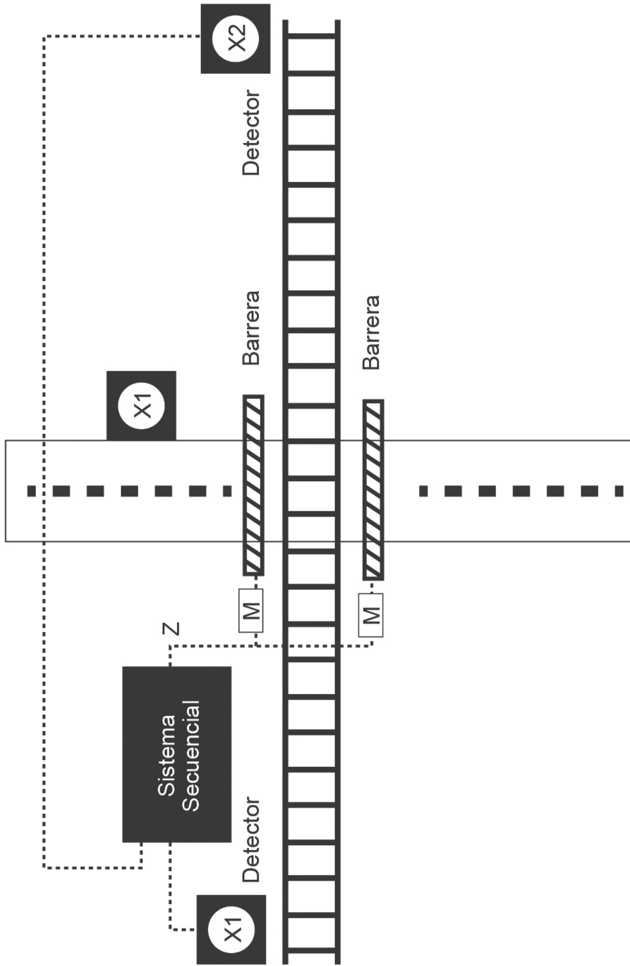


Figura 9.60 Diagrama de controlador de barerras para vías de ferrocarril

Para este problema solo se muestra el diseño de su autómata, una máquina de Moore.

©Editorial Digital Tecnológico de Monterrey

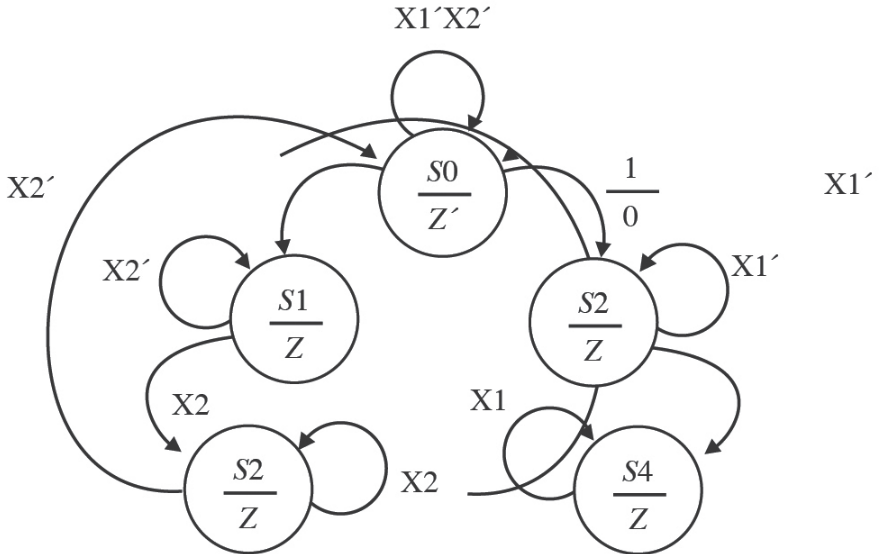


Figura 9.61 Máquina de Moore para seguir control de peso de ferrocarril

Elevador

El diagrama de bloques del controlador de un elevador en un edificio de dos pisos se muestra a continuación. Las entradas FB1 y FB2 son los botones para seleccionar el piso dentro del elevador. Las entradas CALL1 y CALL2 son los botones localizados en la entrada y sirven para llamar al elevador. Las entradas FS1 y FS2 son *switches* de piso que generan un 1 cuando el elevador se encuentra en el primer o segundo piso. Las salidas UP y DOWN controlan el motor, el elevador se detiene cuando UP = DOWN = 0. N1 y N2 son *flip flops* que indican cuando el elevador se necesita en el primer o segundo piso. R1 y R2 son señales que reinician (reset) estos *flip flops*. DO = 1 hace que la puerta del elevador se abra y DC = 1 indica que la puerta del elevador está cerrada.

©Editorial Digital Tecnológico de Monterrey

El diagrama esquemático general es el siguiente:

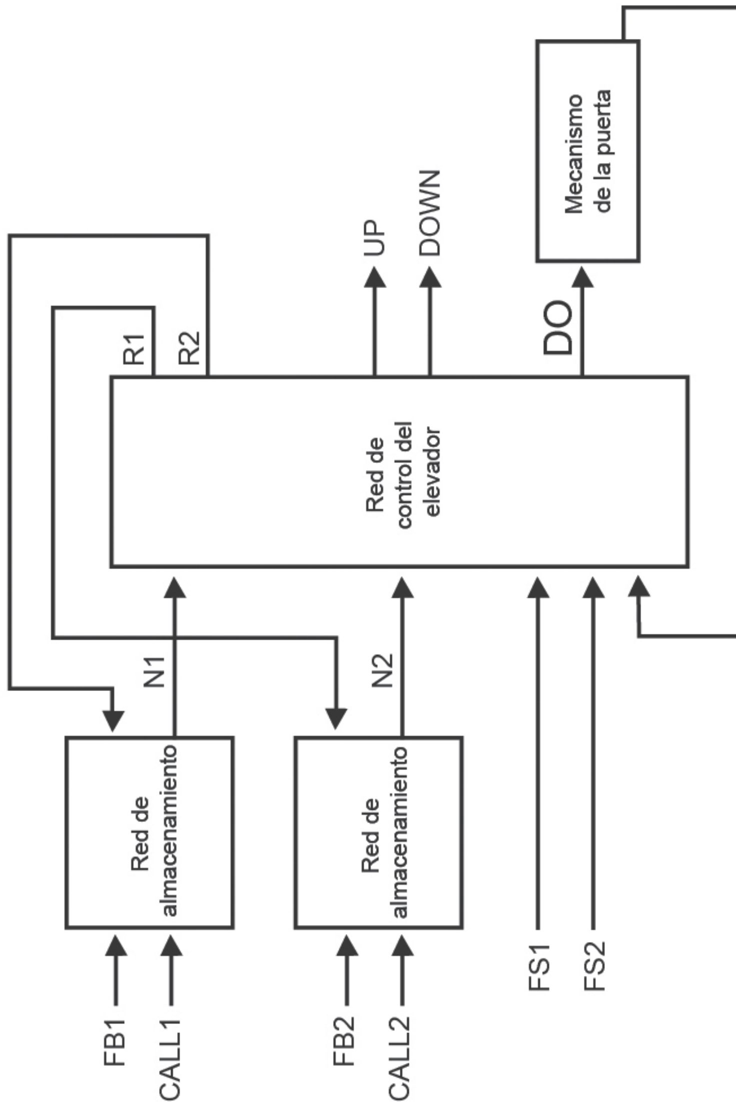


Figura 9.62 Diagrama del controlador del elevador

©Editorial Digital Tecnológico de Monterrey

La máquina de Mealy que lo resuelve es:

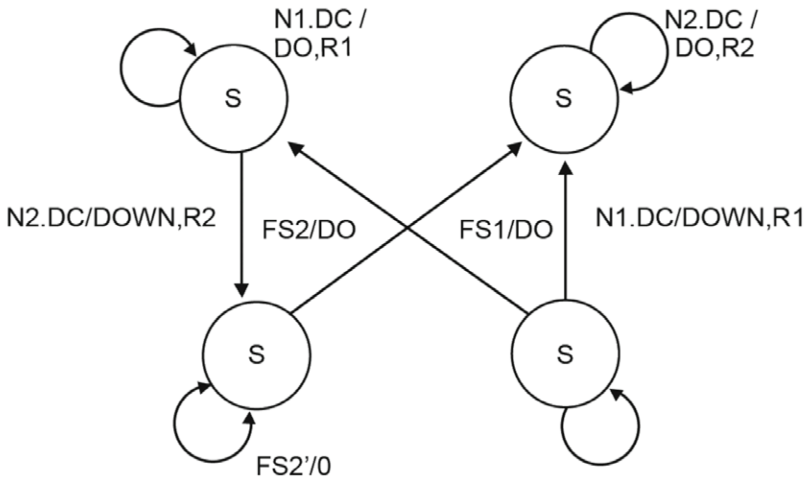


Figura 9.63 Máquina de Mealy del controlador del elevador

Controlador de ratón

Se requiere diseñar el controlador de un ratón electrónico equipado para grabar si existen personas dentro de las oficinas de la compañía ACSA CORPORATION S.A. DE C.V. de las 12:00 p.m. a las 5:00 p.m. El ratón en su modo de encendido siempre se encuentra grabando, la forma en la que el ratón se desplaza dentro de las instalaciones es la siguiente:

El ratón debe maniobrar girando. La nariz del ratón tiene un sensor cuya salida es X = 1, a menos que se encuentre en contacto con un obstáculo, en cuyo caso es X = 0.

La operación default del ratón es caminar derecho, sin embargo, el ratón tiene algunas líneas de entrada control, entre ellas Z1 = 1, que lo hacen girar hacia la izquierda, Z2= 1 hace girar el ratón hacia la derecha.

©Editorial Digital Tecnológico de Monterrey

Cuando el ratón encuentra un obstáculo deberá girar hacia la derecha hasta no detectar obstáculo alguno; la siguiente vez que detecte otro obstáculo el ratón deberá girar hacia la izquierda hasta que no haya obstáculo, y así sucesivamente.

Cuando el tiempo de grabación ha transcurrido (suponga que el ratón genera una señal de control T = 1), el ratón enviará una señal inalámbrica al puesto de control y permanecerá en 1 hasta el momento en el cual e inspector de seguridad lo apague y lo recoja.

En primera instancia, se identificarán las entradas y salidas del controlador del ratón:

La entrada del circuito controlador es la salida del sensor del ratón (X), y las salidas del controlador son las señales que requiere el ratón para girar (Z1, Z2). El ratón tiene cierto funcionamiento autónomo que le permite grabar y la señal que genera T no requiere conectarse al controlador. Con estas especificaciones, el diseño resulta:

©Editorial Digital Tecnológico de Monterrey

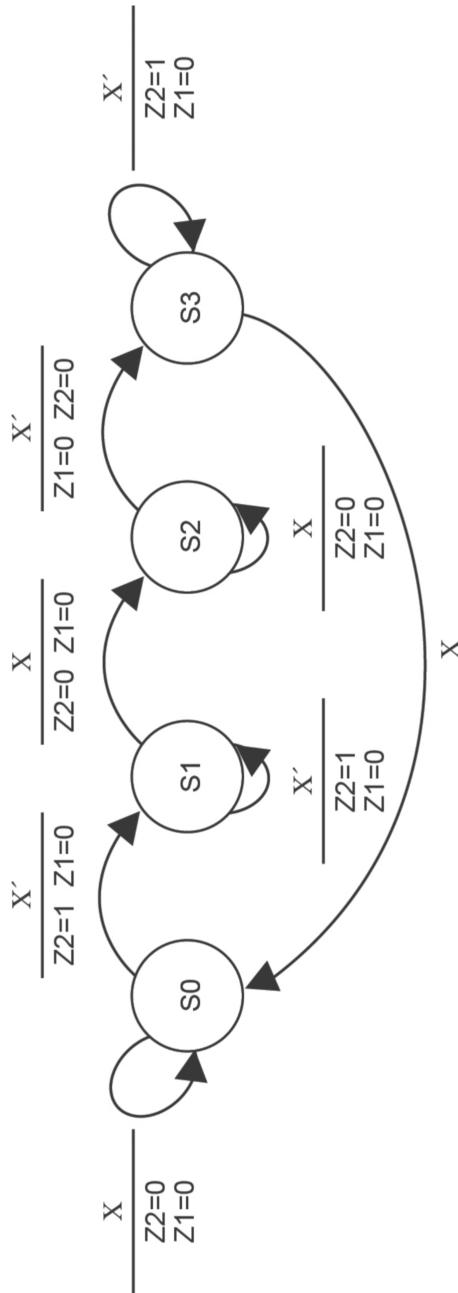


Figura 9.64 Máquina de Mealy del controlador del ratón

©Editorial Digital Tecnológico de Monterrey

Bit de paridad par

Se propone diseñar un circuito que reciba una cadena de *bits* en forma serial. El circuito debe generar un 1 cuando la cantidad de 0 recibida por el circuito sea par, y además también la cantidad de 1 sea par. No tienen que ser iguales las cantidades de 0 y 1, solo se debe cumplir que ambas sean pares. Se considera que una cantidad nula es par.

Para este problema se presenta tanto un diseño de Moore como de Mealy; es probable que el de Moore sea el primero que se le ocurra al diseñador.

Máquina de Moore:

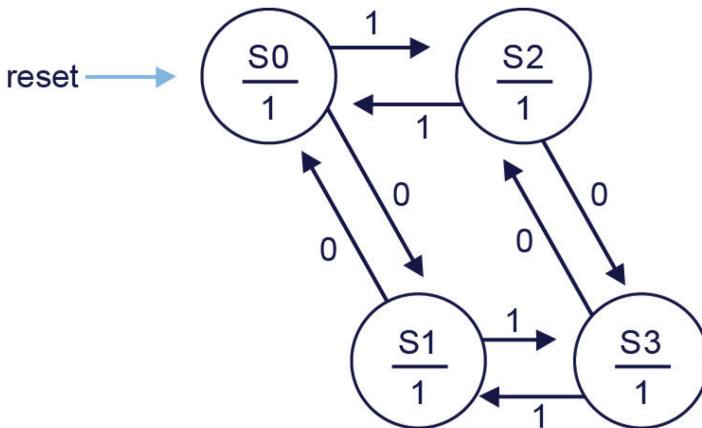


Figura 9.65 Máquina de Moore para generador de *bit* de paridad par

©Editorial Digital Tecnológico de Monterrey

Máquina de Mealy:

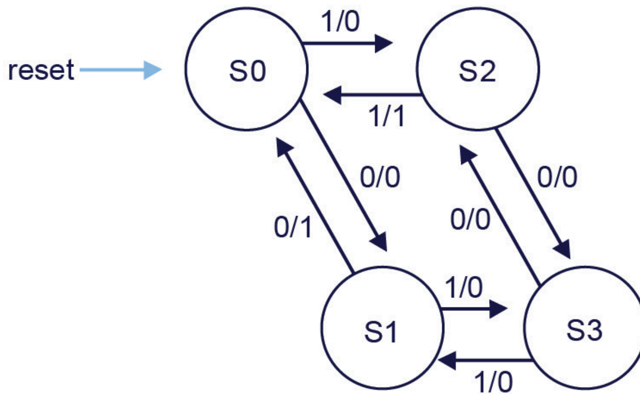


Figura 9.66 Máquina de Mealy para generador de *bit* de paridad par

Secuencia de bits I

Se propone diseñar un circuito que reciba una cadena de *bits* en forma serial. El circuito solo tiene una entrada (X) y una salida (Y). Si en la secuencia de entrada ocurre 0101 o 0110 se debe generar una salida de dos 1 seguidos. El primero de estos 1 debe ocurrir coincidentemente con la última entrada de la secuencia 0101 o 0110. El circuito debe regresar al estado inicial una vez que se genere a la salida el segundo 1. Por ejemplo:

```
X  010011101010101101 ...
Y  000000000011000011 ...
```

El mejor diseño requiere seis estados con una máquina de Mealy.

Nota: Este problema fue tomado del libro: "Fundamentals of Logic Design", de Charles Roth.

©Editorial Digital Tecnológico de Monterrey

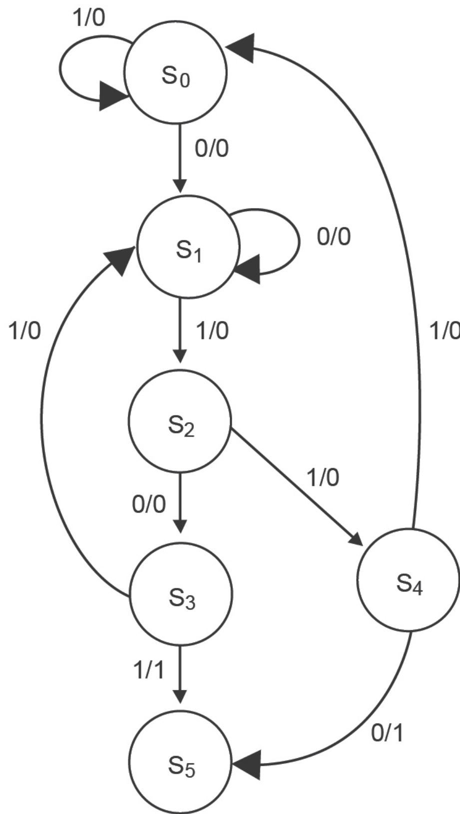


Figura 9.67 Máquina de Mealy para secuencia de *bits*

Secuencia de bits II

Se propone diseñar un circuito que reciba una cadena de *bits* en forma serial. El circuito solo tiene una entrada (X) y dos salidas que ocurren al mismo tiempo (Z1 y Z2). La salida Z1 = 1 ocurre cada vez que se completa la secuencia de entrada 010, dado que la secuencia 100 no ha ocurrido de manera alguna. La salida Z2 = 1 se genera cada vez que ocurre la secuencia de entrada 100. Note que una vez que Z2 sea 1, nunca más debe generarse Z1 = 1, pero no viceversa.

El mejor diseño requiere ocho estados con una máquina de Mealy.

Nota: Este problema está basado en uno propuesto en el libro: "Fundamentals of Logic Design", de Charles Roth.

©Editorial Digital Tecnológico de Monterrey

Figura 9.68 Máquina de Mealy para detectar secuencia de *bits*

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 9

1. Diseñe una máquina de estados a la que ingresa de manera serial números 4 *bits*, a partir del *bit* menos significativo. La salida de esta máquina, llamada Z, representa, en todo momento, un *bit* de paridad impar. Es decir, si el número total de unos que ha ingresado es impar Z=1, de otro modo, Z=0. Considere al cero como par. Cada cuatro *bits* se regresa al estado inicial porque comienza un nuevo número.

a) Diseñe una máquina que cumpla este propósito.

b) Represente esta máquina en VHDL utilizando cualquier nivel de descripción.

2. A un circuito ingresan secuencias de unos y ceros de manera serial. Diseñe una máquina de Mealy que genere una salida igual a 1 cuando la cantidad de unos que ha ingresado es impar y cero cuando es par. A continuación, se muestra un ejemplo:

Entrada: 00010010100011

Salida: 00011100111101

Solo muestre en este espacio el diseño de la máquina.

©Editorial Digital Tecnológico de Monterrey

3. A un circuito ingresan secuencias de unos y ceros de manera serial. Diseñe una máquina de Mealy que genere una salida igual a 1 cuando la cantidad de ceros que ha ingresado es mayor a dos y cero cuando sea menor. A continuación, se muestra un ejemplo:

Entrada: 00010010100011

Salida: 00111111111111

Solo muestre en este espacio el diseño de la máquina.

4. A un circuito ingresan secuencias de unos y ceros de manera serial. Diseñe una máquina de Mealy que genere una salida igual a 1 cuando la cantidad de ceros que ha ingresado es mayor o igual a dos y ha ocurrido la secuencia 101. A continuación, se muestra un ejemplo:

Entrada: 10111111100011

Salida: 00000000011111

Solo trabaje en el diseño de la máquina.

5. Un automóvil modelo 2011 tiene dos focos en cada uno de los dos faroles traseros que se encienden intermitentemente cuando se activan las direccionales. Los focos izquierdos se llaman L1 y L2. Los focos derechos se llaman L3 y L4. Hay una palanca que se utiliza para activar las direccionales. La palanca equivale a contar con dos interruptores (*switches*) uno llamado I (Izquierda) y otro D (Derecha). Si se selecciona direccional izquierdo el *switch* I se pone en 1. El diseño de la palanca hace que nunca puedan estar activados los dos interruptores. Cuando se selecciona con la palanca la

©Editorial Digital Tecnológico de Monterrey

direccional izquierda se enciende primero L1 luego L2 luego L1 etc. hasta que se deja de seleccionar el *switch* I (baja a 0). Cuando los *switches* están en cero las luces están apagadas. Se cuenta con un reloj. La frecuencia con que se prenden y apagan las luces es la frecuencia del reloj.

Para este problema defina una máquina de Moore que cuente con un estado inicial, un estado en que encienda L1, otro con el que encienda L2, y así sucesivamente, cuando los *switches* están en 0 la máquina debe estar en el estado inicial

6. En un tanque hay un líquido para un proceso industrial. Hay dos sensores, S1 y S2 que monitorean el nivel del líquido como se observa en la figura. Estos sensores generan un 1 cuando el líquido cubre estos sensores. El nivel inicial del agua se muestra en la gráfica.

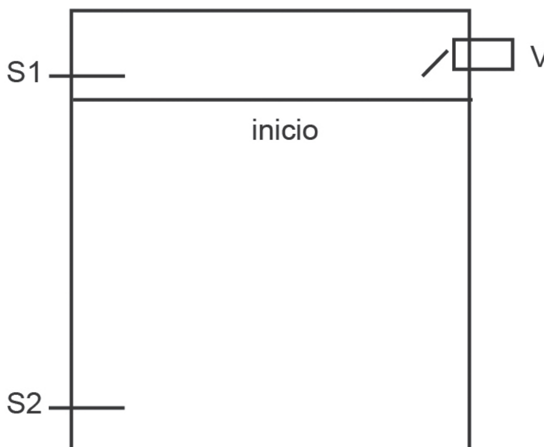


Figura 9.69

©Editorial Digital Tecnológico de Monterrey

El líquido es utilizado (se gasta) en el proceso. Diseñe un circuito que abra una válvula para volver a meter líquido. La válvula se abre al colocarle un 1 en su entrada digital (V). El funcionamiento del circuito debe ser el siguiente. Si el líquido no está cubriendo S2 (que es el nivel mínimo de líquido que debe tener el tanque), entonces hay que abrir la válvula hasta que cubra S1. Dado que con solo dos estados puede implementarse este circuito, utilice solo un *flip flop* D con entrada D1 para diseñarlo. Con Q=0 se modelaría al estado inicial. Diseñe una máquina de Mealy. Como S1 y S2 no pueden tomar el valor 1 y 0 respectivamente, estas combinaciones generan salidas X (que no importan).

Indique el valor óptimo de D1

a) S2’+QS1’

b) S1+QS2’

c) Q’

d) Q

e) Q+S2S1

Indique el valor óptimo de V

f) S2’+QS1’

g) S1+QS2’

h) Q’

i) Q

j) Q+S2S1

k) S1+S2

l) S1’+S2’

©Editorial Digital Tecnológico de Monterrey

7. Se le solicita diseñar un circuito que ilumine LEDs que están configurados por renglones como se muestra en la figura:

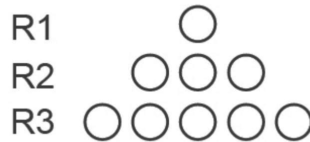


Figura 9.70

La idea es que la salida R2 ilumine los tres leds del renglón 2 y así R3 con sus cinco leds.

Este circuito debe tener conectado a la entrada un *switch* "m" (modo) y las salidas son R1, R2 y R3. El circuito debe iniciar con todos los leds apagados y verificar el valor de m. Si m=0 seguir una secuencia1 si no una secuencia2. Las secuencias se mostrarán como estados del circuito: (diseñe una máquina de Moore)

Estado 0: todo apagado

Si m=0 seguir la secuencia:

Estado 1: iluminar R1

Estado 2: iluminar R1 y R2

Estado 3: iluminar R1, R2 y R3

Continuar con Estado 1

Si m=1 seguir la secuencia:

Estado 4: iluminar R2

Estado 5: iluminar R1 y R3

Continuar con Estado 4

©Editorial Digital Tecnológico de Monterrey

Si en cualquier momento el *switch* m cambia de posición se cambiará al estado 0 en el que se apagarán todos los leds y enseguida se tomará la secuencia que corresponda.

Es importante considerar las salidas que "no importan, X" en la construcción de su tabla y al simplificar las funciones.

Dado que son 5 estados, utilice 3 *flip flops* D con salidas Q1, Q2 y Q3 (Atención, están numerados como Q1, Q2 y Q3 y no como Q2, Q1 y Q0.

Dadas las especificaciones del problema, a la salida R1 (la de en medio) se le conectaría la función:

a) Q1'+Q2

b) 1

c) Q2'+Q3Q1'Q2'Q3m' + Q2Q3'm'

d) Q1'Q2'Q3 + Q2Q3'

e) 0

f) Q

A la entrada D2 se le conectaría la función:

a) 1

b) Q1'Q2'Q3 + Q2Q3'

c) 0

d) Q1'Q2'Q3m' + Q2Q3'm'

e) Q1'+Q2

f) Q2+Q3

g) Q2'+Q3

©Editorial Digital Tecnológico de Monterrey

8. Se desea construir un circuito secuencial que siga la siguiente secuencia 00, 01, 10, 01, 00 … resuelva este problema utilizando una máquina de Moore. Para los estados utilice dos *flip flops* D con salidas Q1 y Q0, codificándolos iniciando en 00. La secuencia de salida se produce con dos funciones F1 y F0.

Para este problema:

Encuentre la función que se debe conectar a Q1:

a) Q1Q0

b) Q1Q0’

c) Q1’Q0

d) Q1

e) Q1’Q0 + Q1Q0’

f) Q0’

Encuentre la función F1

a) Q1Q0

b) Q1Q0’

c) Q1’Q0

d) Q1

e) Q1’Q0 + Q1Q0’

f) Q0’

©Editorial Digital Tecnológico de Monterrey

9. En este problema grafique el valor de Z en el espacio correspondiente.

```
entity comp2 is
        Port (X : in std_logic;
              Z : out std_logic;
              clk : in std_logic);
end comp2;

architecture Behavioral of comp2 is
signal State, Nextstate: integer := 0;
begin
process(State,X) --Combinational Network
begin
case State is
when 0 =>
if X='0' then Z<='0'; Nextstate<=1;
else Z<='1'; Nextstate<=2;
else null;
end if;
when 1 =>
if X='0' then Z<='0'; Nextstate<=3;
else Z<='1'; Nextstate<=4; else null;
end if;
when 2 =>
if X='0' then Z<='1'; Nextstate<=4;
else Z<='0'; Nextstate<=4;
else null;
end if;
when 3 =>
if X='0' then Z<='0'; Nextstate<=0;
else Z<='1'; Nextstate<=0;
else null;
end if;
when 4 =>
if X='0' then Z<='1'; Nextstate<=0;
else Z<='0'; Nextstate<=0;
```

©Editorial Digital Tecnológico de Monterrey

```
else null;
end if;
when others => null;
end case;
end process;
process(CLK ) -- State Register
begin
if (clk ='1')and clk 'event then
State <= Nextstate;
end if;
end process;
end Behavioral;
```

Figura 9.71

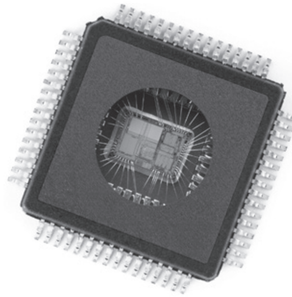
©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 9

La habilidad para diseñar secuencias se desarrolla con la práctica. Los problemas que fueron presentados son representativos de problemas en los que la salida depende únicamente del diseño de la máquina de estados.

Si los problemas fueron resueltos sin ver la solución, y usando esta solo como referencia, ya es seguro su dominio de este tema.

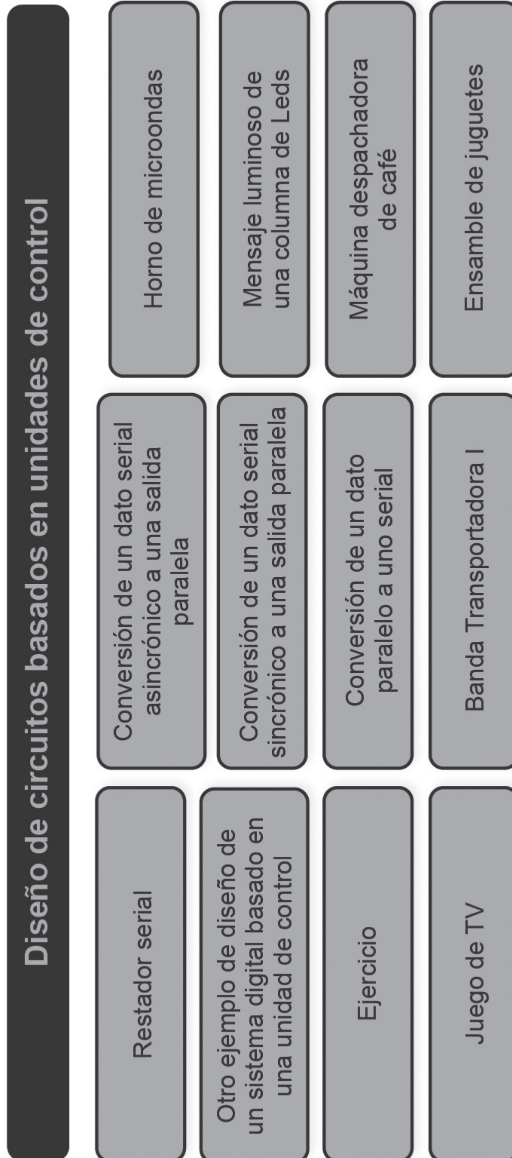


©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 10. Diseño de circuitos basados en unidades de control



©Editorial Digital Tecnológico de Monterrey

Como ya se abundó en el **capítulo 9**, una máquina de estados produce las salidas de un circuito. Al tipo de problemas que se han resuelto se agregará otro concepto. Un autómata también puede producir señales de control para administrar la secuencia de operaciones de un conjunto de registros, de *flip flops*, habilitadores (*enables*) o selectores de diversos componentes. Esto se logra de la siguiente manera: con la lógica combinacional de una máquina de estados se producen salidas que pueden ser conectadas a las entradas de control de registros, *flip flops*, etc., para producir cierto funcionamiento (como cargar, incrementar, decrementar, recorrer, poner en ceros).

En este capítulo se plantean problemas como los del **capítulo 8**, solo que con más operaciones aún, y en lugar de diseñar la lógica combinacional "artesanalmente", es decir, sin una herramienta de diseño, se utiliza una máquina de estados para producir las señales de control. A este tipo de máquina de estados se le denomina Unidad de Control.

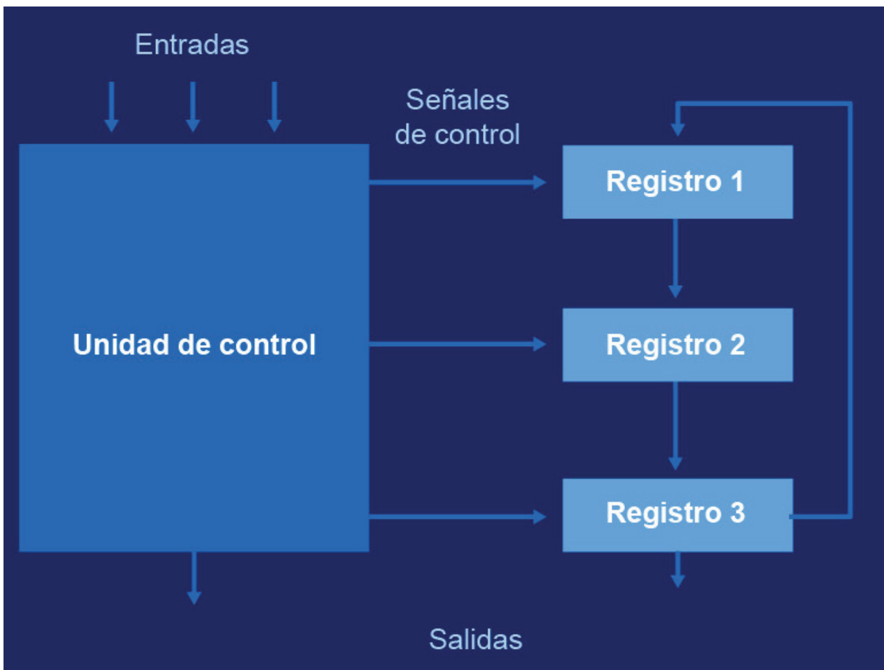


Figura 10.1 Esquema general de un circuito basado en una unidad de control

©Editorial Digital Tecnológico de Monterrey

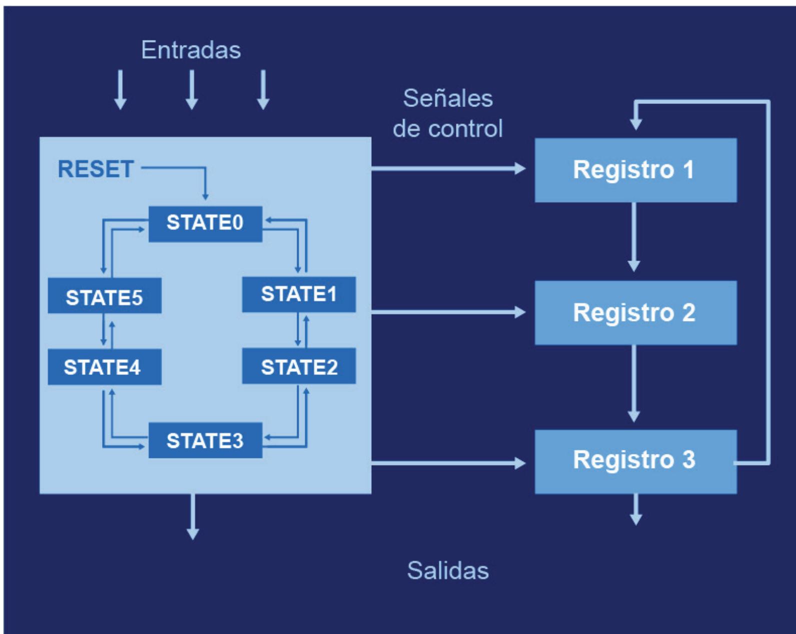


Figura 10.2 Especificación de la unidad de control

Se utilizará un problema muy simple para la operación que se precisa realizar, pero que a la vez resulta complejo debido los recursos que utiliza y la precisión que requiere. Este ejemplo es representativo para la siguiente estrategia de diseño.

©Editorial Digital Tecnológico de Monterrey

10.1 Restador serial

Se plantea realizar la resta de dos números de 16 *bits*, pero en lugar de utilizar 16 *full adders* como es requerido en este circuito, se requiere utilizar solamente un full adder, de tal manera que esta operación típicamente combinacional se resolverá de manera serial, sumando posición por posición de ambos números y propagando el acarreo a la siguiente posición.

El siguiente es un diagrama de bloques del circuito que realiza la resta serial utilizando la técnica de complementos a 2 entre los números que se cargan a los registros X, Y, ambos de 16 *bits*. Como usted recordará, la resta en complementos a 2 puede ser resuelta como X-Y = X + Y* = X+Y'+1. Tal y como se observa en el diseño esquemático, para que se efectúe la resta se suma el *bit* menos significativo de X con el *bit* menos significativo de Y negado. La suma ocurre en un *full adder*, el primer *carry* que se suma es 1 para cumplir con: X-Y = X+Y'+1. El *bit* que representa la suma de los tres *bits*, el menos significativo de X, el de Y negado más el acarreo ingresa a X por la entrada serial. Este circuito recibe una señal de *start* (st), que cuando es recibida por la **unidad de control** (control), los datos XIN y YIN son cargados a los registros X y Y respectivamente. El contador [llamado C, de cuatro *bits*, en la figura se señala como C(4)] se pone en ceros cuando ocurre la señal de load y se incrementa en uno cuando la señal de *shift* está en uno. El *carry* requiere un *flip flop* para recorrerse adecuadamente, porque si se conecta directo el *carry* de salida al de entrada, el *full adder* (que es un circuito combinacional) no se estabilizaría ni tendrá el resultado adecuado en cada pulso de reloj. Cuando el contador llega a 15 se produce la señal C15. La salida del *flip flop carry* debe inicializarse en 1 cuando ocurre la señal de *start* (st). Los registros X, Y y C, así como el *flip flop* D señalado como FF tienen conectado al reloj. Esto no se señala en el esquemático. Suponga que la señal de *start* permanece en 1 al menos durante un ciclo de reloj.

©Editorial Digital Tecnológico de Monterrey

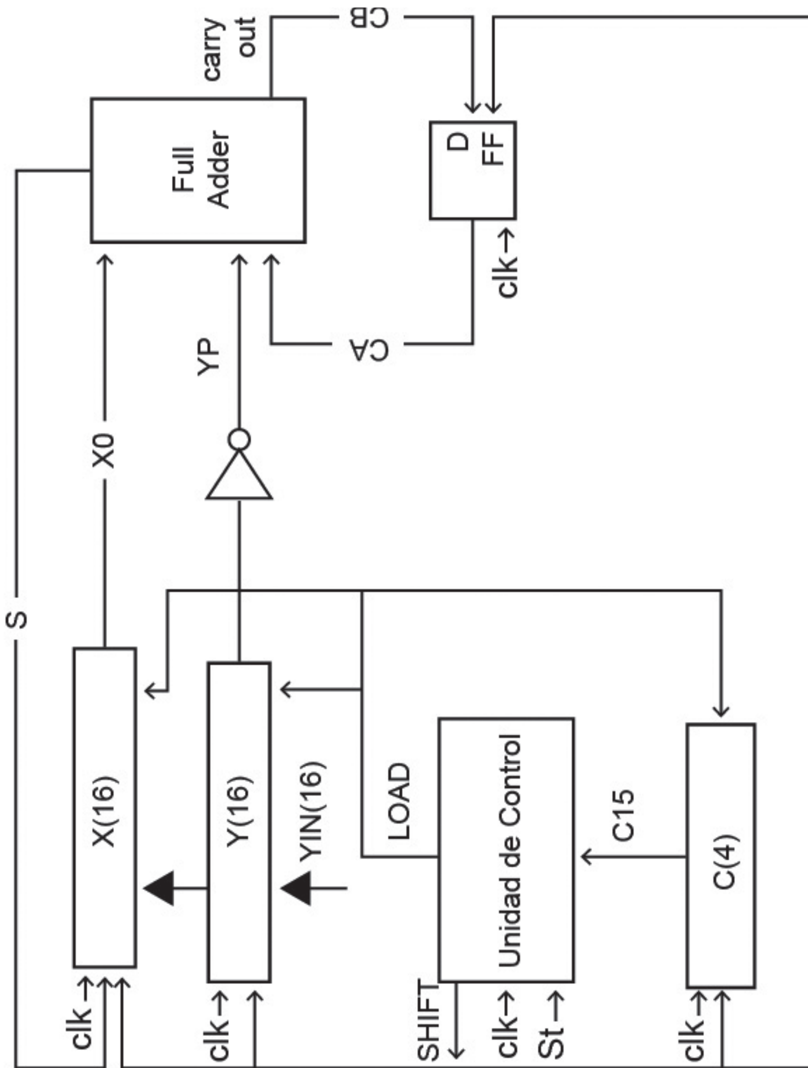


Figura 10.3 Circuito esquemático para resta serial

La secuencia de operaciones que debe realizar el circuito es:

1) Al ocurrir la señal de *start* (st) se cargan los datos de entrada XIN(16 *bits*) y YIN(16 *bits*), para esto la unidad de control debe emitir la señal de LOAD. También al emitir esta señal, el contador C(de 4 *bits*) debe inicializarse a 0.

©Editorial Digital Tecnológico de Monterrey

2) Una vez cargados los datos en los registros X, Y (de 16 *bits*), la suma S y el *carry out* CB quedan listos después del retraso del circuito combinacional *Full Adder*.

3) La unidad de control debe producir un 1 en la señal *SHIFT* que debe estar en 1 durante 16 ciclos de reloj para que la suma S sea cargada por la entrada serial por el *bit* más significativo del registro X (que se usa como acumulador) y el registro se recorra para que la suma se acomode.

4) Cuando hay un 1 en la señal C15, esto ocurre cuando el contador llega a 15 y se habrán efectuado 15 corrimientos, solo faltará uno para terminar el proceso de suma y a continuación la señal de *SHIFT* debe bajar a 0.

El procedimiento descrito se traduce en el siguiente diagrama de estados de la unidad de control. Note que las salidas de esta máquina son las señales de control requeridas por los elementos de este circuito:

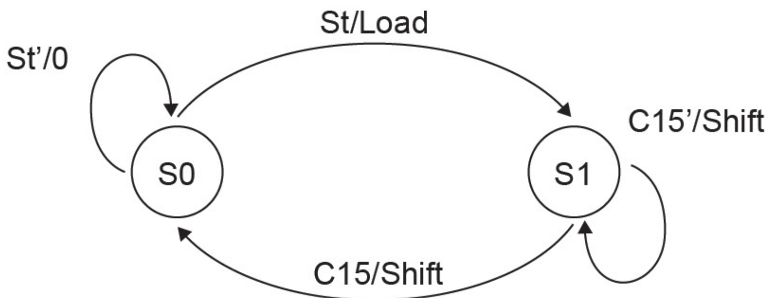


Figura 10.4 Unidad de control del restador serial

©Editorial Digital Tecnológico de Monterrey

Hay dos alternativas principales en la descripción de este sistema digital en VHDL:

Alternativa A: modelar los elementos del diseño esquemático de manera independiente, con las señales de control explícitas.

Alternativa B: modelar todos los elementos en conjunto bajo el diseño de la unidad de control; las señales de control se infieren de la descripción.

Alternativa A

En la primera opción los elementos del circuito que se modelan por separado son:

- La lógica combinacional de la unidad de control, dónde explícitamente se generan las señales *Shift, Load* y *Nextstate.*
- El registro State.
- El registro X,
- El registro Y,
- El registro C,
- El *flip flop,*
- La lógica combinacional del *full adder*: las ecuaciones del *full adder* (para entradas a, b, c) son: Suma = a X or b X or c; *Carry out* = bc + ab + ac.
- La lógica combinacional para generar la señal C15.

©Editorial Digital Tecnológico de Monterrey

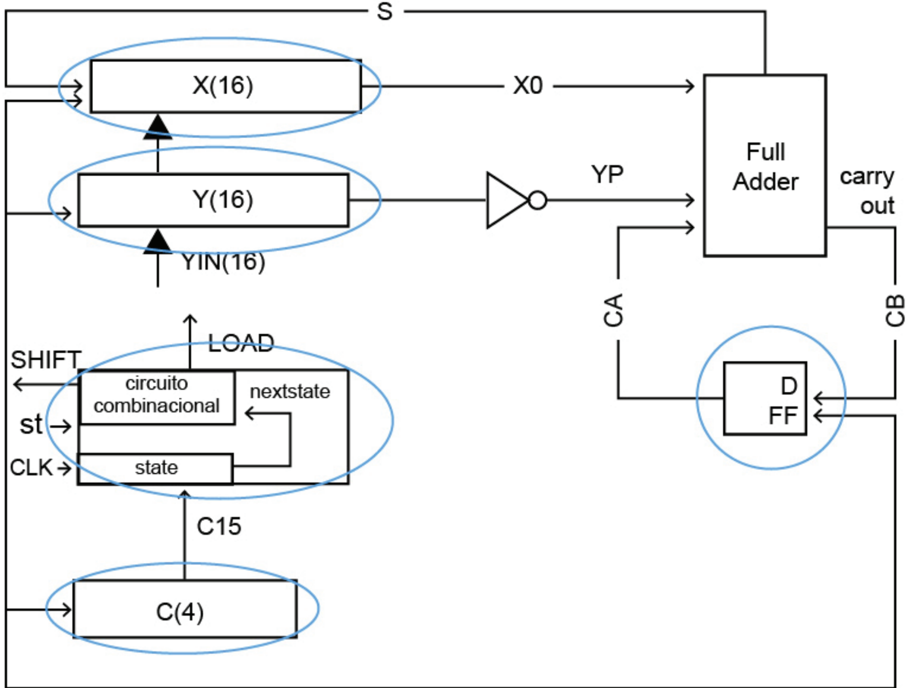


Figura 10.5 Diagrama esquemático de los resgitros y sus señales de control

A continuación, se presenta esta alternativa de modelación, en la que se dejan explícitas las señales de control.

©Editorial Digital Tecnológico de Monterrey

```
entity serial_subtracter is
    port( Xin: in std_logic_vector (15 downto 0 );
           Yin: in std_logic_vector (15 downto 0 );
           St, clk, reset: in std_logic;
           resta: out std_logic_vector (15 downto 0 ));
end serial_subtracter;

architecture Behavioral of serial_subtracter is
signal Load, Shift, State, nextstate, C15, CB, S: std_logic;
signal CA:std_logic;
signal C: std_logic_vector (3 downto 0) ;
signal x, y: std_logic_vector (15 downto 0);
begin
--parte combinacional de la unidad de control
process ( state, St, C15)
     begin
          case state is
          when '0' =>
               Shift <= '0';
               if St = '0' then nextstate<= '0';
                         Load <= '0';
                    else nextstate <= '1';
                         Load <= '1';
               end if;
          when '1' =>
               if C15 = '0' then Shift <= '1';
                              Load <= '0';
                              nextstate <= '1';
                              Else Shift <= '1';
                               Load <= '0';
                               nextstate <= '0';
               end if;
          when others => null;
          end case;
     end process;
```

©Editorial Digital Tecnológico de Monterrey

```
--modelacion del registro X
RegX: process (clk, load, Shift)
      begin
          if load = '1' then
               X <= xin;
          elsif(clk = '1') and clk'event and (Shift = '1') then
               X <= S&x(15 downto 1);
          end if;
      end process;

--modelacion del registro Y
RegY: process(clk, load, Shift)
     begin
          if load='1' then
               Y <= yin;
          elsif(clk = '1') and clk'event and (Shift = '1') then
               Y <= '0'&y(15 downto 1);
          end if;
     end process;

--modelacion del registro C
RegC: process(clk, load, Shift)
      begin
          if(clk = '1') and clk'event then
          if load = '1' then
               C <= "0000";
          if  Shift = '1' then
               C <= C + 1;
          end if;
      end process;
```

©Editorial Digital Tecnológico de Monterrey

```
--modelacion del  flip flop D
process (clk, load, Shift)
     begin
          if load = '1' then
               CA <= '1';
          elsif(clk = '1') and clk'event and (Shift = '1') then
               CA <= CB;
          end if;
     end process;


     --MODELACION DEL REGISTRO (FLIP FLOP) "STATE"
     process (clk, reset)
     begin
          if reset = '1' then
             state <= '0';
          elsif clk = '1' and clk'event then
          state <= nextstate;
          end if;
     end process;

--FULL ADDER
     CB <= (x(0) and (not y(0))) or (x(0) and CA) or ((not y(0)) and CA);
     S <= (x(0) XOR (NOT y(0))) XOR CA;
--C15
     C15 <= C(0) AND (C(1)) and C(2) and C(3);
     resta <= x;
     end behavioral;
```

©Editorial Digital Tecnológico de Monterrey

A continuación, se muestra otra opción de modelación. Bajo este esquema de modelación de los componentes por separado, que modela state en un proceso y las señales de control en otro, se elimina *Nextstate*.

```
entity serial_subtracter is
    port(Xin: in std_logic_vector (15 downto 0 );
         Yin: in std_logic_vector (15 downto 0 );
         St, clk, reset: in std_logic;
                         resta: out std_logic_vector (15 downto 0 ));
end serial_subtracter;

architecture Behavioral of serial_subtracter is
signal Load, Shift, State, nextstate, C15, CB, S: std_logic;
signal CA:std_logic;
signal C: std_logic_vector (3 downto 0) ;
signal x, y: std_logic_vector (15 downto 0);
begin
--parte combinacional de la unidad de control
process(state, St, C15)
     begin
          case state is
          when '0' =>
               Shift <= '0';
               if St = '0' then     Load<='0';
                     else Load <= '1';
               end if;
          when '1' =>
               if C15 = '0' then Shift <= '1';
                              Load <= '0';
                              Else Shift <= '1';
                               Load <= '0';
               end if;
          when others => null;
          end case;
     end process;
```

©Editorial Digital Tecnológico de Monterrey

```
--modelacion del registro X
RegX: process (clk, load, Shift)
      begin
          if load = '1' then
               X <= xin;
          elsif(clk = '1') and clk'event and (Shift = '1') then
               X <= S&x(15 downto 1);
          end if;
      end process;

--modelacion del registro Y
RegY: process (clk, load, Shift)
      begin
          if load='1' then
               Y <= yin;
          elsif(clk = '1') and clk'event and (Shift = '1') then
               Y <= '0'&y(15 downto 1);
          end if;
      end process;

--modelacion del registro C
RegC: process(clk, load, Shift)
      begin
          if load = '1' then
               C <= "0000";
          elsif(clk = '1') and clk'event and (Shift = '1') then
               C <= C + 1;
          end if;
     end process;

--modelacion del flip flop D
process (clk, load, Shift)
     begin
     if load = '1' then
               CA <= '1';
```

©Editorial Digital Tecnológico de Monterrey

```
elsif(clk = '1') and clk'event and (Shift = '1') then
                CA <= CB;
            end if;
        end process;


        --MODELACION DEL REGISTRO (FLIP FLOP) "STATE"
        process (clk, reset)
        begin
            if reset = '1' then
               state <= '0';
            elsif clk = '1' and clk'event then
            case state is
            when '0' =>
                if St = '0' then state <= '0';
                        else state <= '1';
                end if;
            when '1' =>
                if C15 = '0' then state <= '1';
                            else state <= '0';
                end if;
            when others => null;
            end case;
        end process;
            end if;
        end process;

--FULL ADDER
        CB <= (x(0) and (not y(0))) or (x(0) and CA) or ((not
y(0)) and CA);
        S <= (x(0) XOR (NOT y(0))) XOR CA;
--C15
        C15 <= C(0) AND (C(1)) and C(2) and C(3);
        resta <= x;
end behavioral;
```

©Editorial Digital Tecnológico de Monterrey

En la siguiente opción se utilizan dos procesos. En el primero se modela el circuito combinacional que determina tanto el siguiente estado como las señales de control, mientras que en el segundo se modelan los registros state, X, Y y el *flip flop* D que funcionan en la transición positiva del reloj. El resto del código muestra la modelación del *full adder* y de la señal C15. Esta opción también modela en forma explícita las señales de control, la diferencia radica en modelar varios elementos bajo el mismo proceso.

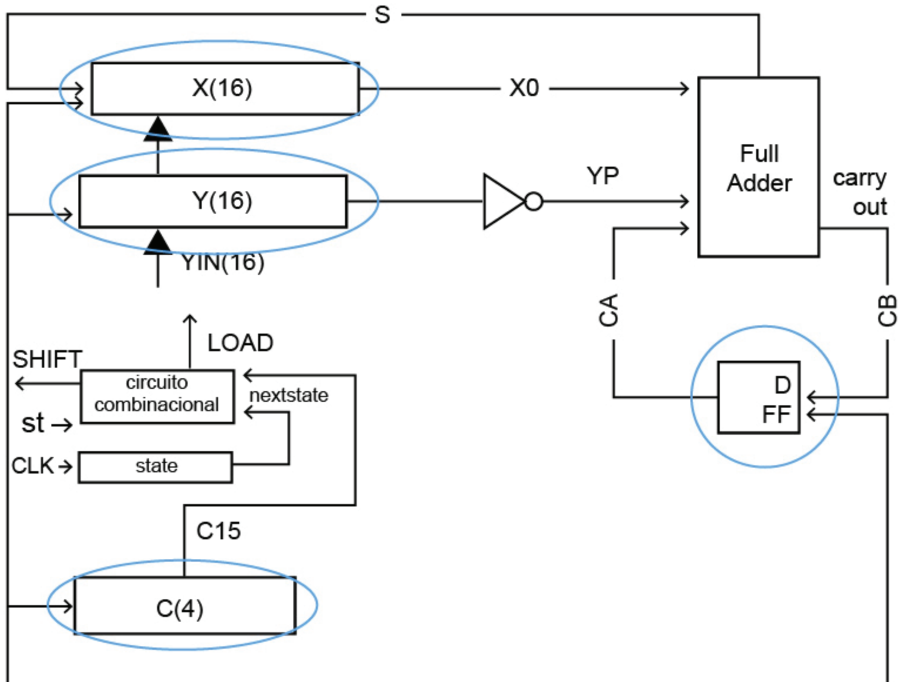


Figura 10.6 Diseño esquemático del restador serial

©Editorial Digital Tecnológico de Monterrey

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity serial_subtracter is
    port(Xin: in std_logic_vector (15 downto 0);
         Yin: in std_logic_vector (15 downto 0);
         St, clk, reset : in std_logic;
                     resta: out std_logic_vector (15 downto 0));
end serial_subtracter;

architecture Behavioral of serial_subtracter is
signal Load, Shift, State, nextstate, C15, CB, S: std_logic;
signal CA: std_logic;
signal C: std_logic_vector (3 downto 0) ;
signal x, y: std_logic_vector (15 downto 0);
begin
     process ( state, St, C15)
     begin
          case state is
          when '0' =>
               Shift <= '0';
               if St = '0' then nextstate <= '0';
                         Load <= '0';
                     else nextstate <= '1';
                         Load <= '1';
               end if;
          when '1' =>
               if C15 = '0' then Shift <= '1';
                             Load <= '0';
                             nextstate <= '1';
                    Else Shift <= '1';
                              Load <= '0';
                              nextstate <= '0';
               end if;
```

©Editorial Digital Tecnológico de Monterrey

```
            when others=> null;
            end case;
        end process;

--modelacion de los registros, del contador y del flip flop
        process(clk, load, Shift)
        begin
            if load = '1' then
                CA <= '1';
                x <= xin;
                y <= yin;
                C <= "0000";
            elsif(clk = '1') and clk'event and (Shift = '1') then
                x <= S&x(15 downto 1);
                y <= '0'&y(15 downto 1);
                CA <= CB;
                C <= C + 1;
            end if;
        end process;

        --MODELACION DEL FLIP FLOP "STATE"
        process (clk, reset)
        begin
            if reset = '1' then
               state <= '0';
            elsif clk = '1' and clk'event then
            state <= nextstate;
            end if;
        end process;
        --FULL ADDER
        CB <= (x(0) and (not y(0))) or (x(0) and CA) or ((not y(0)) and CA);
        S <= (x(0) XOR (NOT y(0))) XOR CA;
--C15
        C15 <= C(0) AND (C(1)) and C(2) and C(3);
        resta <= x;
        end behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Alternativa B

La siguiente es la opción de modelación más sencilla y más directa. Consiste en describir los registros y sus funciones, realizadas con sus entradas de señales de control a través de la lógica de la unidad de control, describiendo las condiciones que deben ocurrir para que se efectúe una función. Las señales de control quedan implícitas, las infiere el sistema de desarrollo, y no tendrán un nombre. También la lógica combinacional puede quedar embebida en el proceso. A continuación, se muestra la descripción del **restador** bajo esta perspectiva.

```
signal State, C15, CB, S: std_logic;
signal CA: std_logic;
signal C: std_logic_vector ( 3 downto 0) ;
signal x, y: std_logic_vector (15 downto 0);
begin
     process (reset, clk)
     begin
     if reset = '1' then
              state <= '0';
elsif clk = '1' and clk'event then  --se modelarán los
registros, así que se requiere el
  --reloj
          case state is
          when '0' =>
               if st = '0' then state <= '0';
                     else state <= '1';
                         CA <= '1';
                         X <= xin;
                         Y <= yin;
                         C <= "0000";
               end if;
```

©Editorial Digital Tecnológico de Monterrey

```
when '1' =>
                if C15 = '0' then
                           x<=S&x(15 downto 1);
                           y<= '0'&y(15 downto 1);
                           CA <= CB;
                           C<=C+1;
                           state<= '1';
                               Else
                           x<=S&x(15 downto 1);
                           --ya no se genera la señal de
                           --shift, se modela el shift
                           y<= '0'&y(15 downto 1);
                           CA <= CB;
                           C<=C+1;
                            state <= '0';
                end if;
           when others=> null;
                -- no debe ocurrir debido al reset
           end case;
      else null;
     end if;  end process;
           --Full adder y C15
     CB<= (x(0) and (not y(0))) or (x(0) and CA) or ((not y(0))
and CA);
     S<= (x(0) XOR (NOT y(0))) XOR CA;
     C15 <= C(0) AND (C(1)) and C(2) and C(3);
     resta<=x;
     end behavioral;
```

©Editorial Digital Tecnológico de Monterrey

10.2 Otro ejemplo de diseño de un sistema digital basado en una unidad de control

El siguiente ejemplo presenta un circuito para realizar una división numérica de números sin signo (el algoritmo se explicará en el capítulo 14). Se utilizará este ejemplo para concentrar la atención en la codificación del circuito (el circuito esquemático se muestra en la figura). El propósito de este ejemplo no es entender el funcionamiento del circuito, sino observar la manera en que es posible describirlo.

Como ya se revisó en el circuito anterior, en un diseño basado en una unidad de control es posible describir su circuito con los siguientes enfoques:

a) Generar en el autómata las señales de control con un nombre propio y conectarlas a los componentes que requieren señales de sincronización, o bien,

b) Modelar los registros con las señales de control implícitas.

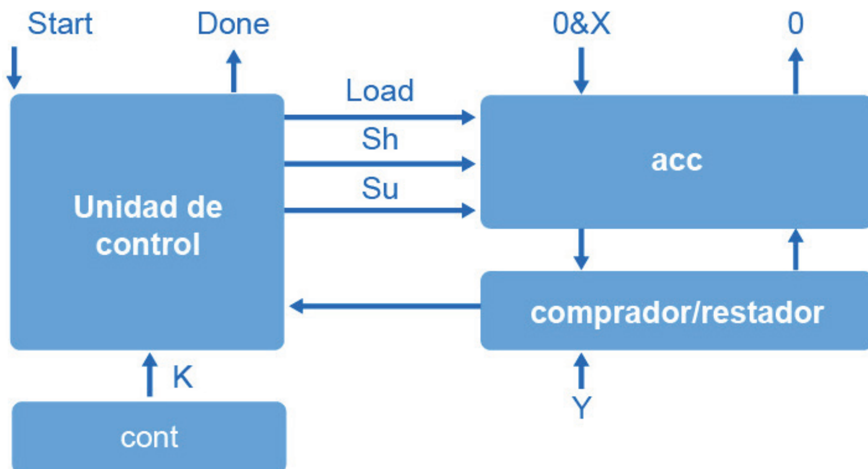


Figura 10.7

©Editorial Digital Tecnológico de Monterrey

Funcionalmente, ambos enfoques de descripción son totalmente equivalentes.

a) **Descripción del circuito con señales de control explícitas.**

El diagrama en el que se basa la descripción en VHDL se presenta a continuación:

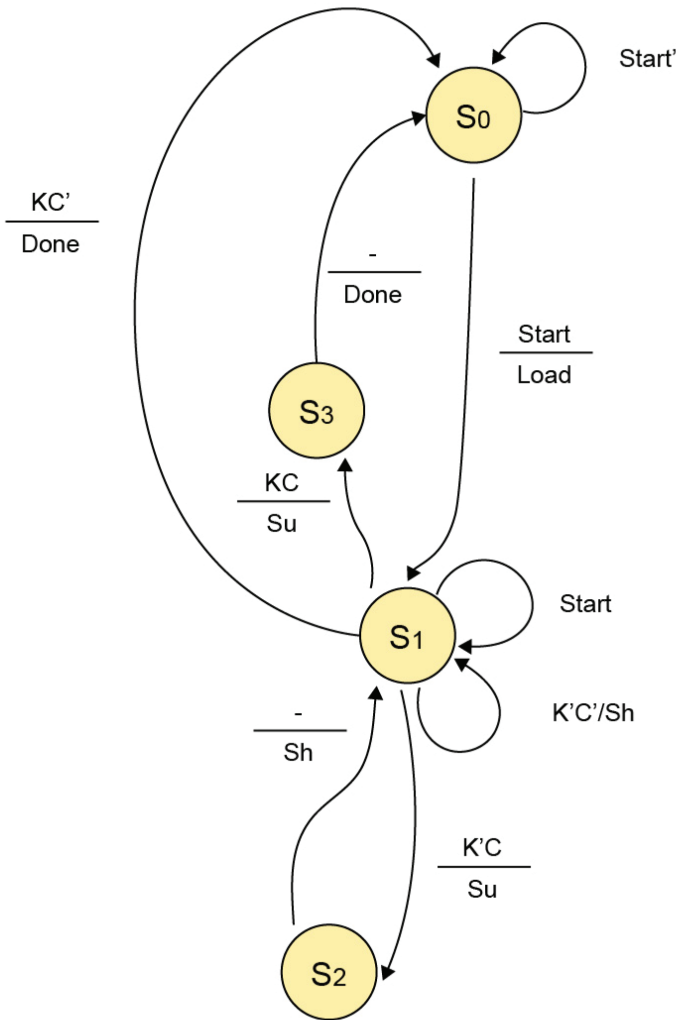


Figura 10.8 Autómata de divisor

©Editorial Digital Tecnológico de Monterrey

En el diagrama no se muestran las salidas en 0, solo se muestran las salidas en 1, pero en el diseño en VHDL sí se deben modelar la señal correctamente, es decir, el momento en que debe subir a 1 o bajar a 0.

La parte combinacional genera tanto a las señales Nextstate como a las señales de control. Las entradas a la parte combinacional provienen también de bloques auxiliares que llevan la cuenta de los *bits* del cociente y que generan la señal “c” del algoritmo de la división.

El código es el siguiente:

```
entity div is
    Port(x: in std_logic_vector(7 downto 0);
         y: in std_logic_vector(3 downto 0);
         acc: inout std_logic_vector(12 downto 0);
     reset, start: in std_logic;
     clk: in std_logic;
     done: out std_logic);
end div;

architecture Behavioral of div is
type STATE_TYPE is (S0, S1, S2, S3, S4);

attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of STATE_TYPE: type is "000 001 010 011 100";

signal state, nextstate: STATE_TYPE;
signal su,sh,load,k, c: std_logic;
signal count: std_logic_vector(3 downto 0);
begin
process(state, k, c)
begin
case state is
```

©Editorial Digital Tecnológico de Monterrey

```
when S0 =>      if Start = '1' then
                sh <= '0';
                su <= '0';
                nextstate <= S1;
                load <= '1';
                done <= '0';
                else null;
                end if;
     when S1 => if k = '0' and c = '0' then
                sh <= '1';
                su <= '0';
                nextstate <= S1;
                elsif k = '0' and c = '1' then
                su <= '1';
                sh <= '0';
                nextstate <= S2;
                elsif k = '1' and c = '0' then
                su <= '0';
                sh <= '0';
                nextstate <= S0;
                done <= '1';
                elsif k = '1' and c = '1' then
                su <= '1';
                sh <= '0';
                nextstate <= S3;
                end if;
     when S2 =>     sh <= '1';
                su <= '0';
                nextstate <= S1;

           when S3 =>     done <= '1';
                sh <= '0';
                su <= '0';
                nextstate <= S0;
      end case;
  end process;
```

©Editorial Digital Tecnológico de Monterrey

```
  process(start, clk)
 begin
  if reset = '1' then
       state <= S0;
       elsif clk = '1' and clk'event
     then
        state <= nextstate;
     end if;
     end process;
  process(clk)
  begin
  if clk = '1' and clk'event then
   if load = '1' then acc <= "0000"&x&'0';
        elsif sh = '1' then acc <= acc(11 downto 0)&'0';
          elsif su = '1' then acc <= (acc(12 downto 8)-('0'&y))&acc(7 downto 1)&'1';
          else null;
     end if;
  end if;
  end process;
  c <= '1' when (acc(12 downto 8) >= ('0'&y)) else '0';
  process(load, clk)
  begin
    if load = '1' then
        count <= "0000";
          elsif (sh = '1') and (clk = '1') and clk'event
           then
            count <= count+1;
      end if;
  end process;
  k <= '1' when (count = "1000") else '0';
end Behavioral;
```

b) Modelación de los registros con las señales de control implícitas.

Este circuito es funcionalmente equivalente al anterior, pero tiene algunas modificaciones, como el primer *shift* hecho desde la carga del acumulador y la señal de *start* implementada tanto como reset externo, como señalada dentro del diagrama de estados.

©Editorial Digital Tecnológico de Monterrey

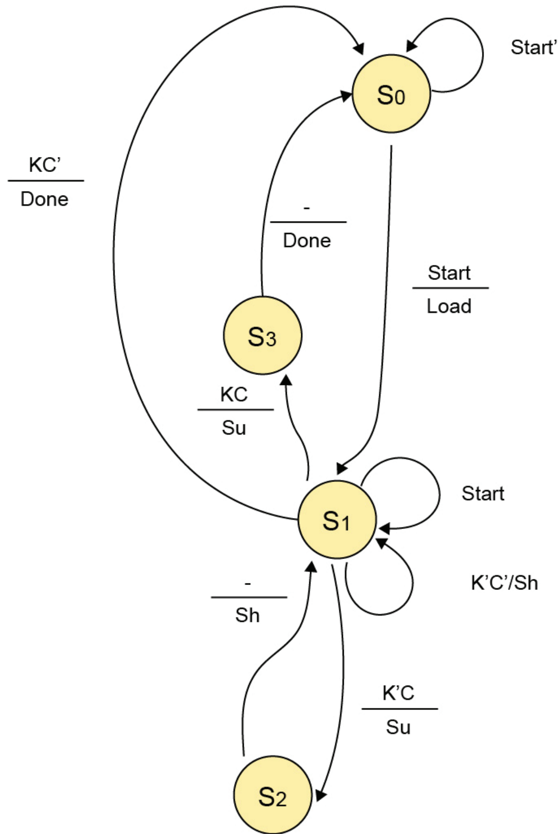


Figura 10.9 Autómata de divisor con señales de control

©Editorial Digital Tecnológico de Monterrey

El código es el siguiente:

```
entity div2 is
    Port(x: in std_logic_vector(7 downto 0);
         y: in std_logic_vector(3 downto 0);
         acc: inout std_logic_vector(12 downto 0);
                    start: in std_logic;
                    clk: in std_logic;
                    done: out std_logic);
end div2;

architecture Behavioral of div2 is
type STATE_TYPE is (S0, S1, S2, S3);

attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of STATE_TYPE: type is "00 01 10 11";

signal state: STATE_TYPE;
signal k, c: std_logic;
signal count:s td_logic_vector(2 downto 0);
begin
process(start, clk)
begin
  if start = '1' then
    state <= S1;
      acc <= "0000"&x&'0';
      count <=  "000";
      done <= '0';
      elsif clk = '1' and clk'event
        then
          case state is
                                when S0 => if start = '1' then
```

©Editorial Digital Tecnológico de Monterrey

```
                        state <= S1;
                        acc <= "0000"&x&'0';
                        count <= "000";
                        done <= '0';
                        else state<=S0;
                   end if;
              when S1 => if start = '1' then state <= S1;
                   elsif k = '0' and c = '0' then
                        acc <= acc(11 downto 0)&'0';
                        count <= count  + 1;
                        state <= S1;
                        elsif k = '0' and c = '1' then
                   acc <= (acc(12 downto 8)-('0'&y))&acc(7 downto 1)&'1';
                        state <= S2;
                        elsif k =' 1' and c = '0' then
                        state <= S0;
                        done <=' 1';
                        elsif k= '1' and c = '1' then
                   acc <= (acc(12 downto 8)-('0'&y))&acc(7 downto 1)&'1';
                        state <= S3;
                   end if;
         when S2 => acc <= acc(11 downto 0)&'0';
                   count <= count + 1;
                   state <= S1;
                   when S3 => done <= '1';
                   state< = S0;
end case;
  end if;
  end process;
  c<='1' when (acc(12 downto 8) >= ('0'&y)) else '0';
  k<='1' when (count = "111") else '0';
end Behavioral;
```

Este estilo de diseño es más fácil de codificar.

©Editorial Digital Tecnológico de Monterrey

10.3 Ejercicio

Analice los problemas anteriores y mencione si hay alguna diferencia entre el circuito que se produce, si se sigue la alternatva de descripción a) o b), y cuál de estas alternativas de descripción es la óptima.

©Editorial Digital Tecnológico de Monterrey

10.4 Juego de TV

En este problema se diseñará un sistema para llevar el funcionamiento de un programa de televisión en el que se lleva a cabo un concurso entre tres participantes. La prueba que tienen que pasar es la de contestar unas preguntas eligiendo una de las tres opciones de respuestas que se les dan. Para proporcionar su respuesta, cada jugador cuenta con tres botones. Hay una persona que opera el sistema y se encuentra detrás del escenario; este operador dispone de tres interruptores para indicar cuál es la respuesta correcta. A estos tres interruptores los manejaremos como un vector que llamaremos "correcto". También hay un botón *start* que sirve para iniciar el juego, otro botón "pregunta" para iniciar una nueva pregunta y un botón de *reset* para inicializar todo en ceros. Antes de formular una pregunta a los jugadores, el operador coloca la respuesta correcta en el interruptor correspondiente, luego se formula la pregunta y los jugadores indican su respuesta. Solo se toma en cuenta la respuesta del primer participante que pulse su respuesta, en ese momento se evalúa si es correcta, de ser así se le suman 10 puntos en su marcador, pero si falla se le restan 5 puntos (los circuitos tienen un tiempo de retraso muy pequeño, así que sería imposible considerar que dos jugadores opriman un botón al mismo tiempo). Si ningún jugador contesta en 3 segundos, entonces se les restan 5 puntos a todos.

El circuito determinará si la respuesta ha sido correcta al compararla con la que el operador haya indicado en los interruptores "correcto" antes de iniciar cada pregunta, y que cambiará entre pregunta y pregunta antes de pulsar "pregunta". Cuando algún participante llegue a 100 puntos o más, entonces habrá ganado y el juego se detendrá, activándose la salida correspondiente al jugador que haya ganado. Así todo se queda en este estado hasta que el operador oprima *reset*.

La frecuencia del reloj es de 1024 Hz.

Dadas las señales que se definieron, la entidad quedaría de la siguiente manera:

©Editorial Digital Tecnológico de Monterrey

```
Entity juego is
Port(
---reloj de freguencia fija 1024 Hz.
    Clk: in std_logic;
---diferentes botones del operador
    start, reset, pregunta: in std_logic;
---contiene la respuesta correcta:
    correcto: in std_logic_vector (1 to 3);
---botones de los jugadores A, B, C respectivamente:
    pulsaA, pulsaB, pulsaC: in std_logic_vector (1 to 3);
---marcadores de cada jugador A, B, C:
    marcaA, marcaB, marcaC: out std_logic_vector(7 downto 0);
---señales para indicar el ganador:
    ganaA, ganaB, ganaC: out std_logic;
end juego;
```

©Editorial Digital Tecnológico de Monterrey

La unidad de control en la que se muestran señales de control que luego se modelan como operaciones sobre sus respectivos registros, es la siguiente:

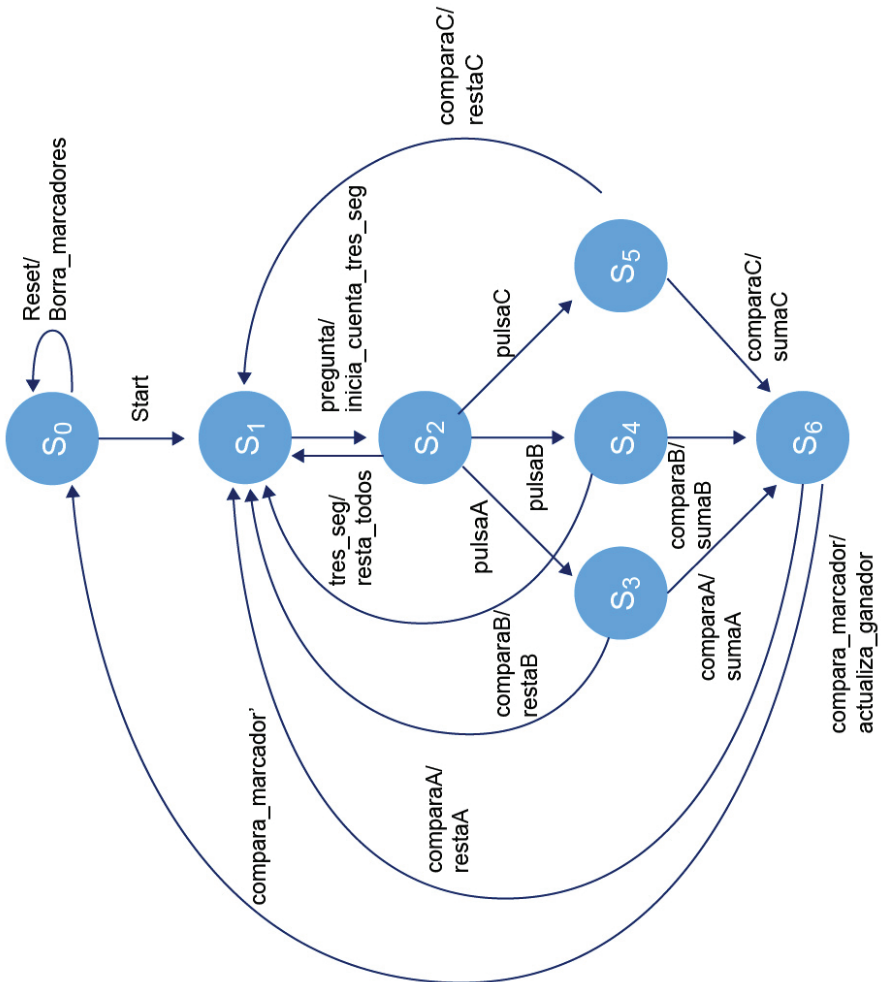


Figura 10.10 Unidad de control juego TV

©Editorial Digital Tecnológico de Monterrey

La descripción del circuito en VHDL es:

```
entity juego is
    Port(clk: in std_logic;  reset: in std_logic; start: in std_logic; pregunta: in std_logic;
          pulsaA, pulsaB, pulsaC: in std_logic_vector(1 to 3);
          correcto: in std_logic_vector(1 to 3);
   marcaA, marcaB, marcaC: inout integer;
          ganaA: out std_logic;
          ganaB: out std_logic;
          ganaC: out std_logic);
end juego;
architecture Behavioral of juego is
signal State, contador: integer;
signal tres_seg, Cuenta: std_logic;
begin
process(reset, clk) --modelacion de máquina de estados y registros
begin
if reset = '1' then
     contador <= 0; marcaA <= 0; marcaB <= 0; marcaC <= 0; ganaA <= '0';ganaB <= '0';ganaC <= '0';
     State <= 0;
elsif clk='1' and clk'event then
     case State is
     when 0 =>
          if Start='1' then State <= 1; else null; end if;
     when 1 =>
          if pregunta='1' then  State <= 2; else State <= 1; end if;
     when 2 =>
          contador<=contador+1;
          if (pulsaA(1) = '1') or (pulsaA(2)='1') or (pulsaA(3) = '1') then State <= 3;
          elsif (pulsaB(1) = '1') or (pulsaB(2) = '1')o r (pulsaB(3) = '1') then State <= 4;
          elsif (pulsaC(1) = '1') or (pulsaC(2) = '1') or (pulsaC(3) = '1') then State <= 5;
          elsif tres_seg = '1' then  marcaA <= marcaA-5; marcaB <= marcaB-5;
```

©Editorial Digital Tecnológico de Monterrey

```
marcaC <= marcaC - 5; State <= 1;
          else State <= 2;
          end if;
     when 3 =>
          if pulsaA = correcto then marcaA <= marcaA + 10; State <= 6;
          else marcaA <= marcaA-5; State<=1;end if;
     when 4 =>
          if pulsaB = correcto then marcaB <= marcaB + 10; State <= 6;
          else marcaB<=marcaB-5; State<=1;end if;
     when 5 =>
          if pulsaC = correcto then marcaC <= marcaC +1 0; State <= 6;
          else marcaC<=marcaC-5; State<=1;end if;
     when 6 =>
          if marcaA > 100 then ganaA <= '1';    State <= 0;
          elsif marcaB > 100 then ganaB <=' 1'; State <= 0;
          elsif marcaC > 100 then ganaC <=' 1'; State <= 0;
          else State <= 1;
          end if;
     when others => null;
     end case;
end if;
end process;
tres_seg <= '1' when (contador = 3072) else '0';
end Behavioral;
```

Note que los marcadores se cambiaron a inout para que fuera posible operar con ellos en el código (sumar o restar puntos) y a la vez presentarlos como salida del puerto.

©Editorial Digital Tecnológico de Monterrey

10.5 Conversión de un dato serial asincrónico a una salida paralela

Los circuitos que apoyan la conexión con dispositivos periféficos de entrada y salida (I/O por sus siglas en inglés) representan un área de oportunidad para los circuitos construidos en circuitos configurables como son los FPGAs. Muchos dispositivos periféricos de entrada, como los ratones y los teclados, envían sus datos en forma serial.

En este problema se pide diseñar un circuito que reciba un dato que ingresa *bit* a *bit* en serie y lo convierta en una salida paralela. También con este problema se introduce el manejo de periféricos. El dato de entrada consta de un *bit* de *start* y 8 bits consecutivos. Se reconoce que se inicia la recepción del dato serial porque la entrada S, que normalmente es 1, baja a 0, se queda en 0 por 1 mseg y luego comienza la recepción de los ocho *bits*. La recepción del dato ocurre a una frecuencia de 1*bit*/mseg. Los *bits* que ingresan no necesariamente se encuentran en fase con Clk. Para evitar errores en la lectura del **dato serial**, cada *bit* se muestrea 8 veces, es decir, el valor de la señal de entrada se lee y se procesa 8 veces. La primera lectura se desperdicia, y de las otras siete lo que haya ocurrido el mayor número de veces entre 1 y 0 es el valor que se considera para el *bit*. La salida paralela se forma con los ocho *bits* elegidos. Se cuenta con un reloj Clk de 8 períodos por mseg, es decir, 8000 Hz (8 KHz).

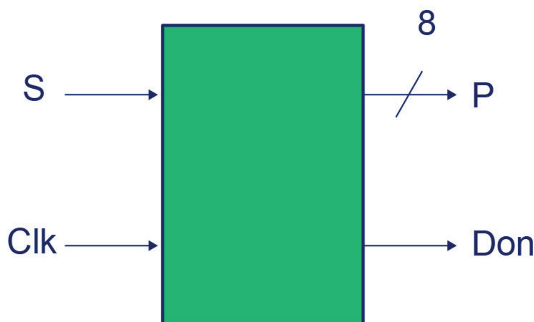


Figura 10.11 Puerto de recepción serial

©Editorial Digital Tecnológico de Monterrey

El tiempo es un factor importante en la sincronización con un transmisor serial.

Una solución para este problema se plantea en el siguiente autómata, el cual no debe tener cambios de estado sin considerar la entrada serial porque la entrada está ocurriendo en todo momento, durante la recepción de los ocho *bits* y, por lo tanto, en todos los ciclos del reloj. Note que se utiliza un estilo diferente para definir la máquina de estados. En vez de usar como salidas los nombres de las señales de control se están indicando las acciones que realizan, para que sean explícitas las acciones correspondientes a estas señales.

©Editorial Digital Tecnológico de Monterrey

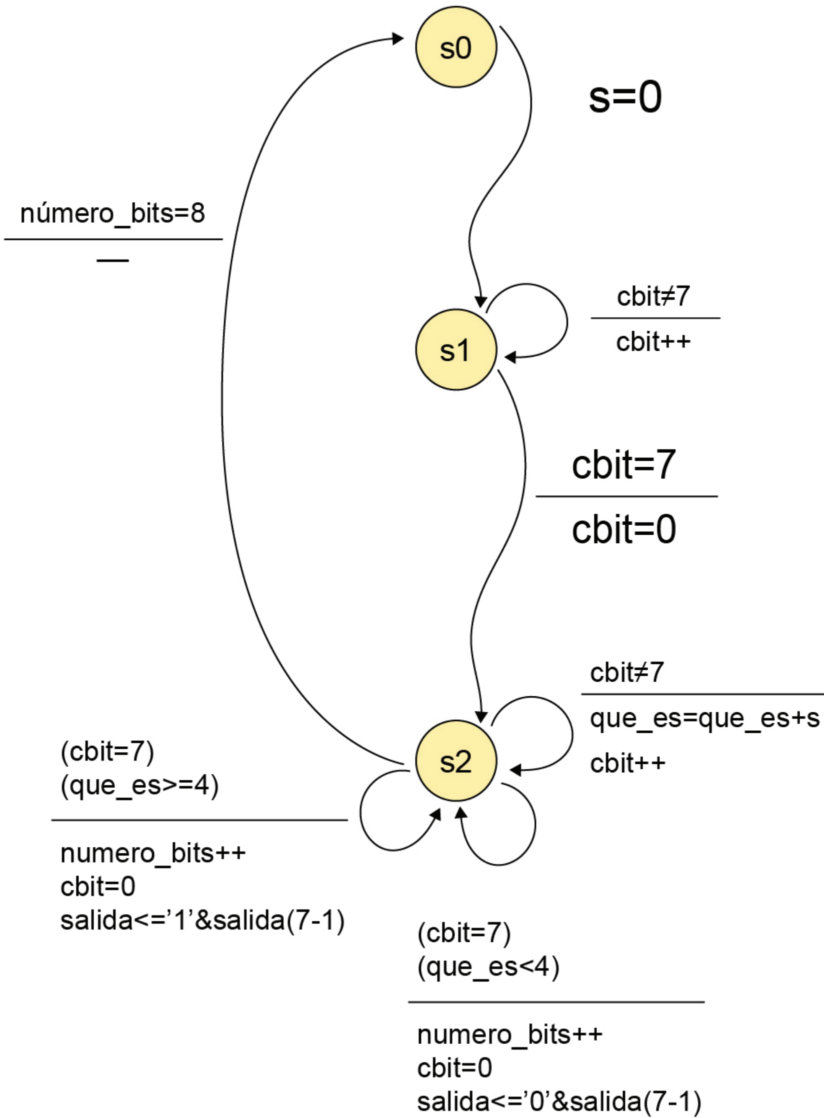


Figura 10.12 Autómata de recepción serial

Note que la condición *cbit* = 7 está de sobra en dos ramas del estado s2, sin embargo, se dejó esta condición para mayor claridad.

©Editorial Digital Tecnológico de Monterrey

El código de la arquitectura equivalente a este diseño es el siguiente:

```
architecture Behavioral of serial_paralelo is
signal numero_bits: std_logic_vector(3 downto 0);
signal que_es, cbit: std_logic_vector(2 downto 0);
signal state: std_logic_vector(1 downto 0);
signal salida: std_logic_vector(7 downto 0);
begin
process(reset, clk)
begin
if reset = '1' then state <= "00";
   elsif clk = '1' and clk'event then
     case state is
     when "00" => if s='1'      then  state <= "00";
                        else  cbit <= "000";
                              state <= "01";
                              numero_bits <= "0000";
                  end if;
     when "01" => if not(cbit = "111" )   then  state <= "01";
                                              cbit <= cbit+1;
                                        else  cbit <= "000";
                                              state <= "10";
                  end if;
```

©Editorial Digital Tecnológico de Monterrey

```
      when "10" => if numero_bits = "1000" then  state <= "00";
                   elsif not(cbit = "111")    then  state <= "10";
                                        que_es <= que_es + S;
                                        cbit <= cbit + 1;
                   elsif que_es < 4      then  cbit <= "000";
                                   salida <= '0'&salida(7 downto 1);
                              numero_bits <= numero_bits + 1;
                                        state <= "10";
                   else                 salida <= '1'& salida(7 downto 1);
                                        cbit <= "000";
                                   numero_bits <= numero_bits + 1;
                                        state <= "10";
                end if;
      when others => null;
      end case;
    else null;
   end if;
  end process;
 P <= salida;
 end Behavioral;
```

Y su simulación:

La cual muestra un manejo adecuado de los ciclos de reloj.

©Editorial Digital Tecnológico de Monterrey

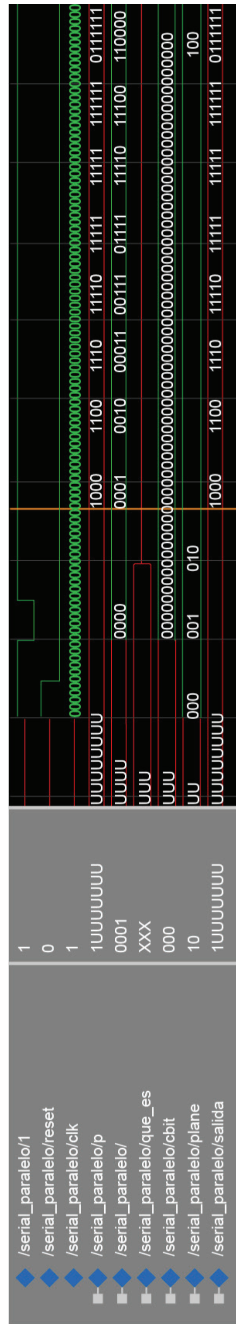


Figura 10.13 Simulación de recepción serial

©Editorial Digital Tecnológico de Monterrey

10.6 Conversión de un dato serial sincrónico a una salida paralela

Este es otro problema de recepción serial, pero más simple, pues el componente que envía los datos seriales también envía la señal de reloj con la que efectúa la transmisión.

En este problema se pide diseñar un circuito que reciba un dato serial y lo convierta a una salida paralela. El dato de entrada consta de un *bit* de *start* y 8 *bits* de datos. Se reconoce que se inicia la recepción del dato serial porque la entrada S, que normalmente está en 1 excepto en la recepción de un dato, baja a 0, se queda en cero por 1 mseg y luego comienza el dato. La recepción del dato ocurre a una frecuencia de 1*bit*/mseg. La salida paralela se forma con los ocho *bits* que ingresen.

El Puerto del circuito es el siguiente:

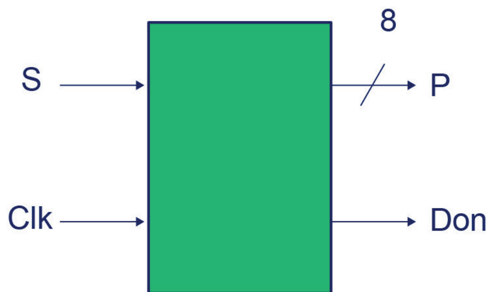


Figura 10.14 Puerto de recepción serial sincrónica

Descrito en VHDL es:

```
entity serialtoparallel is
    Port(S: in std_logic;
         Clk: in std_logic;
         P: out std_logic_vector(7 downto 0);
         Done: out std_logic;);
end serialtoparallel;
```

©Editorial Digital Tecnológico de Monterrey

A diferencia del problema anterior, la señal de Clk proviene del mismo componente que el que genera el *bit* serial, entonces es posible precisar la sincronía sin problema.

La máquina de estados que resuelve el problema es:

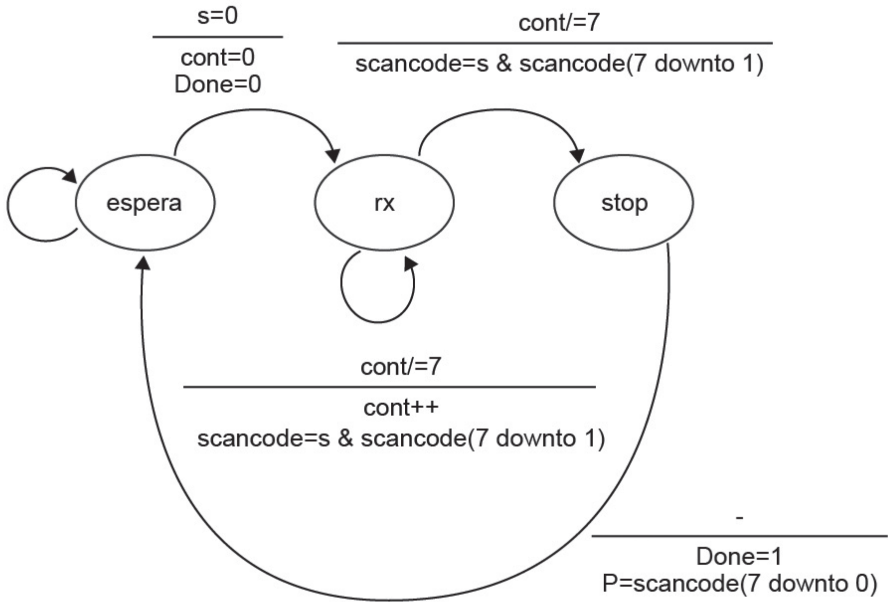


Figura 10.15 Autómata de recepción serial sincrónica

El código equivalente es:

```
entity serialtoparallel is
    Port(S: in std_logic;
         Clk: in std_logic;
         P: out std_logic_vector(7 downto 0);
         Done: out std_logic);
end serialtoparallel;

architecture Behavioral of serialtoparallel is
```

©Editorial Digital Tecnológico de Monterrey

```
type state_type is (espera, rx, stop);
signal state: state_type;
signal scancode: std_logic_vector(8 downto 0);
signal cont: std_logic_vector(3 downto 0);
begin
     process(Clk)
     begin
          if(Clk'event and Clk = '1') then
               case state is
                    when espera =>
                         if S = '1' then
                         state <= espera;
                         else
                         state <= rx;
                         cont <= "0000";
                         Done <= '0';
                         end if;
                    when rx =>
                              if cont = 7 then
                              state <= stop;
                              scancode <= S & scancode(7 downto 1);
                              else
                              state <= rx;
                              cont <= cont + 1;
                              scancode <= S & scancode(7 downto 1);
                         end if;
                    when stop =>
                         state <= espera;
                         Done <= '1';
                         P <= scancode(7 downto 0);
               end case;
          end if;
     end process;
end Behavioral;
```

La definición de scan code en la sección signal debe ser de 7 a 0, sin embargo, si se define excedida no genera error, solo un *warning* que indica que es una señal sin uso en el circuito, mejor corregirla.

©Editorial Digital Tecnológico de Monterrey

10.7 Conversión de un dato paralelo a uno serial

En este circuito se efectúa una transmisión serial simple. El problema consiste en diseñar un circuito que transmita serialmente el dato P de 8 *bits*, en forma sincrónica con la señal clk. La transmisión debe iniciar cuando la señal de entrada valid sea 1. La señal de salida es S. La transmisión debe ocurrir de la siguiente manera: por S debe salir 1, cuando suba la señal valid S deberá bajar a 0 durante un ciclo de reloj, luego comenzará la transmisión serial de P. Al terminar la transmisión, la señal Done deberá subir a 1. El Puerto del circuito es:

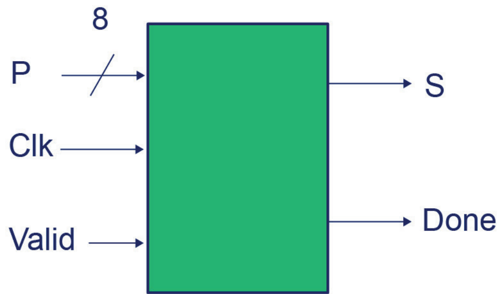


Figura 10.16 Puerto de transmisión serial

©Editorial Digital Tecnológico de Monterrey

El diagrama de la unidad de control es:

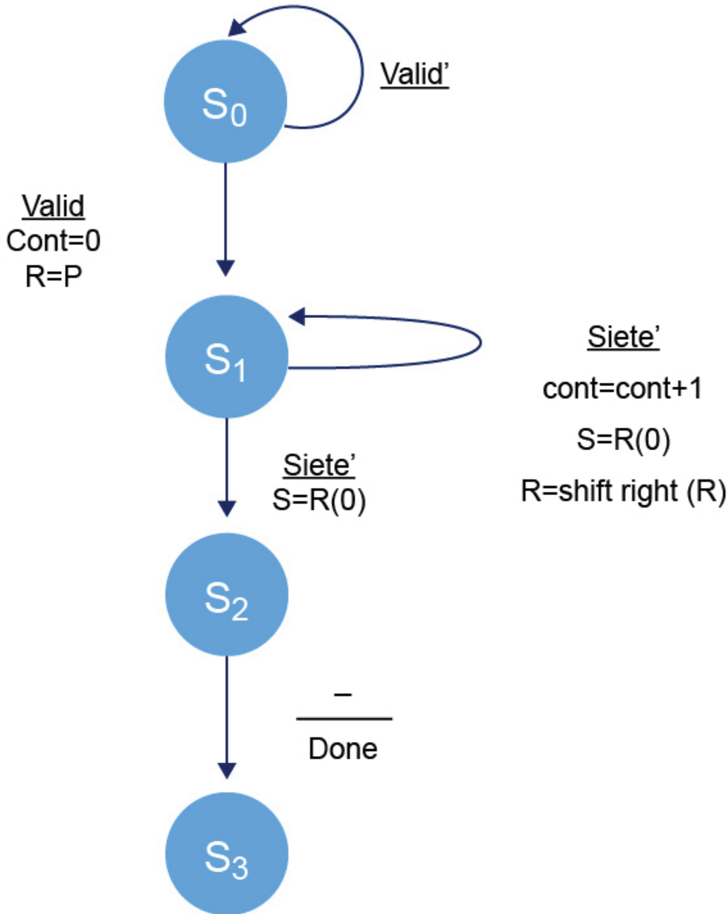


Figura 10.17

©Editorial Digital Tecnológico de Monterrey

El código es:

```
entity Paralleltoserial is
    Port(P: in std_logic_vector(7 downto 0);
         Valid: in std_logic;
    Clk: in std_logic;
    S: out std_logic;
    Done:out std_logic);
end Paralleltoserial;

architecture Behavioral of paralleltoserial is
type STATE_TYPE is (S0, S1, S2, S3);
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of STATE_TYPE: type is "00 01 10 11";

signal State: STATE_TYPE: = S0;
signal cont: std_logic_vector(2 downto 0);
signal siete: std_logic;
signal R: std_logic_vector(7 downto 0);
begin
    process(Clk, Valid)
    begin
       elsif(Clk'event and Clk = '1') then
       case State is
            when S0 =>
if (Valid = '0') then
                 State <= S0;
                 S <= '1';
     else
          cont <= "000";
          R <= P;
                S <= '0';
                State <= S1;
```

©Editorial Digital Tecnológico de Monterrey

```
when S1 =>
if siete = '0' then
         cont<=cont+1;
         S<=R(0);
              R <= '0'&R(7 downto 1);
              State <= S1;
else
     S <= R(0);
     State<=S2;
end if;
when S2 =>
     Done<='1';
     State<=S0;
     S<='1';
     when others => null;
    end case;
  end if;
  end process;
 siete <= '1' when(cont = 7) else '0';
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

10.8 Banda transportadora I

Este problema está basado en un problema presentado en el libro: "El arte de programar sistemas digitales", de David Maxinez y Jessica Alcalá.

Se requiere diseñar, en VHDL, un sistema digital integrado en un circuito que lleve el control de un proceso de una planta de fármacos en el llenado de frascos con tres diferentes tipos de sustancias químicas (todas líquidas). Cada tipo de estas sustancias está almacenado en tres recipientes diferentes, denominados A, B y C. El llenado debe realizarse de la siguiente forma: cada frasco recibe primero el líquido del recipiente A, posteriormente el B y finalmente el C, tal y como se observa en la siguiente figura.

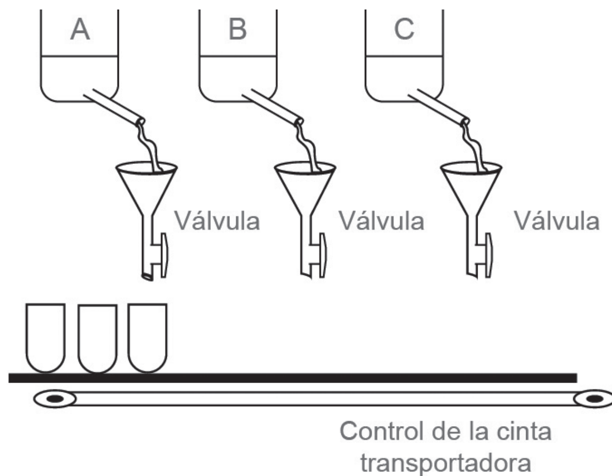


Figura 10.18 Sistema industrial de banda transportadora

En cada frasco debe vaciarse el líquido del recipiente A durante 1 segundo, el del recipiente B por 2 segundos y el del recipiente C por 3 segundos. Cuando las tres válvulas se hayan cerrado, el circuito que diseñará deberá enviar un pulso al control del motor de la banda transportadora. Este motor de pasos está calibrado para que al recibir un 1 lógico la banda avance hasta que los frascos se coloquen debajo del siguiente embudo, repitiéndose entonces el proceso anterior. Solo habrá que controlar esta parte del proceso, pues el resto del proceso, el cerrado y retiro de los frascos es manual.

©Editorial Digital Tecnológico de Monterrey

Notas:

- El proceso inicia operando solo al primer alimentador, luego al primero y al segundo, y luego funciona para una secuencia infinita de frascos, es decir, no se diseñará para la condición que es cuando quedan uno o dos frascos solamente.

- Su circuito debe generar la señal necesaria para que el motor reciba un pulso para desplazar los frascos al siguiente alimentador. El controlador del motor envía una señal cuando este se ha detenido.

- Suponga que al recibir un 1 lógico una válvula se abre y permanece abierta hasta que recibe un cero lógico (son válvulas industriales que normalmente están cerradas, aunque también las hay normalmente abiertas). El circuito que diseñará debe contener los componentes y señales necesarias para abrir o cerrar cada válvula.

- Se requiere diseñar el *hardware* que genere el 1 que necesita el motor para avanzar, y esto es hasta que las tres válvulas se hayan cerrado.

- Se cuenta con un reloj de 100 Hz.

En este problema se muestra el diseño de la máquina de estados (la unidad de control) que genera las señales necesarias para el control de la banda, así como su descripción en VHDL.

Para este problema se presentan dos soluciones: una en la que se utilizan registros y *latches* auxiliares, y la unidad de control envía sus señales de control para que de estos componentes se deriven los controles para la banda; y otra más simple en la que las señales de control de la banda se generan desde la unidad de control. La segunda es la mejor.

©Editorial Digital Tecnológico de Monterrey

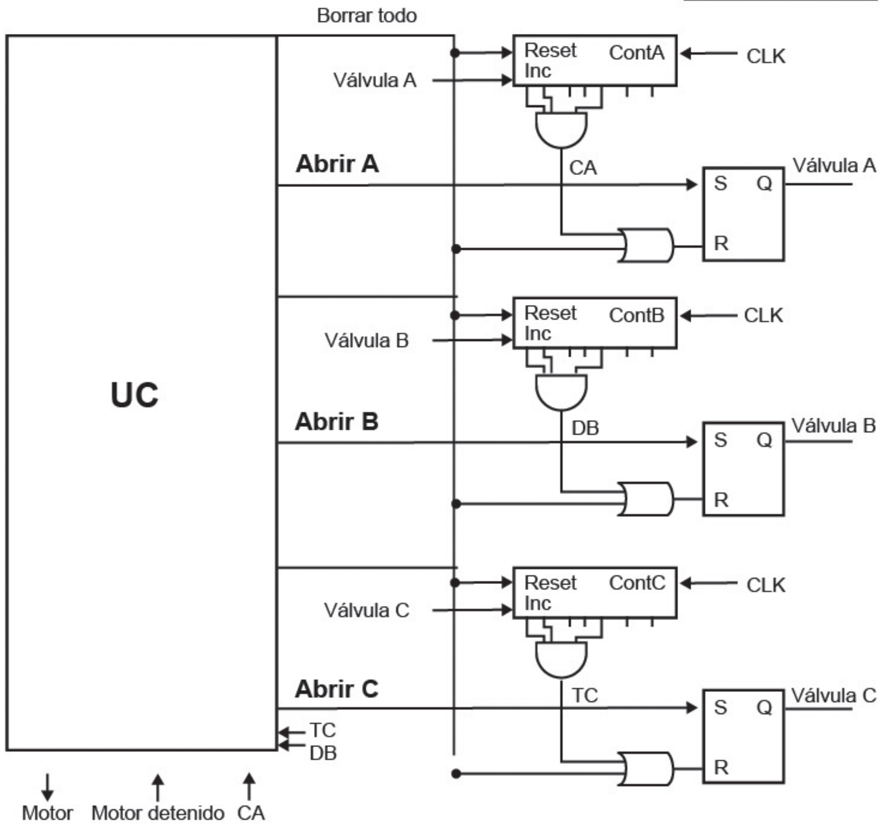


Figura 10.19 Diagrama esquemático para banda transportadora basado en tres registros y latches

©Editorial Digital Tecnológico de Monterrey

El diseño de la unidad de control es:

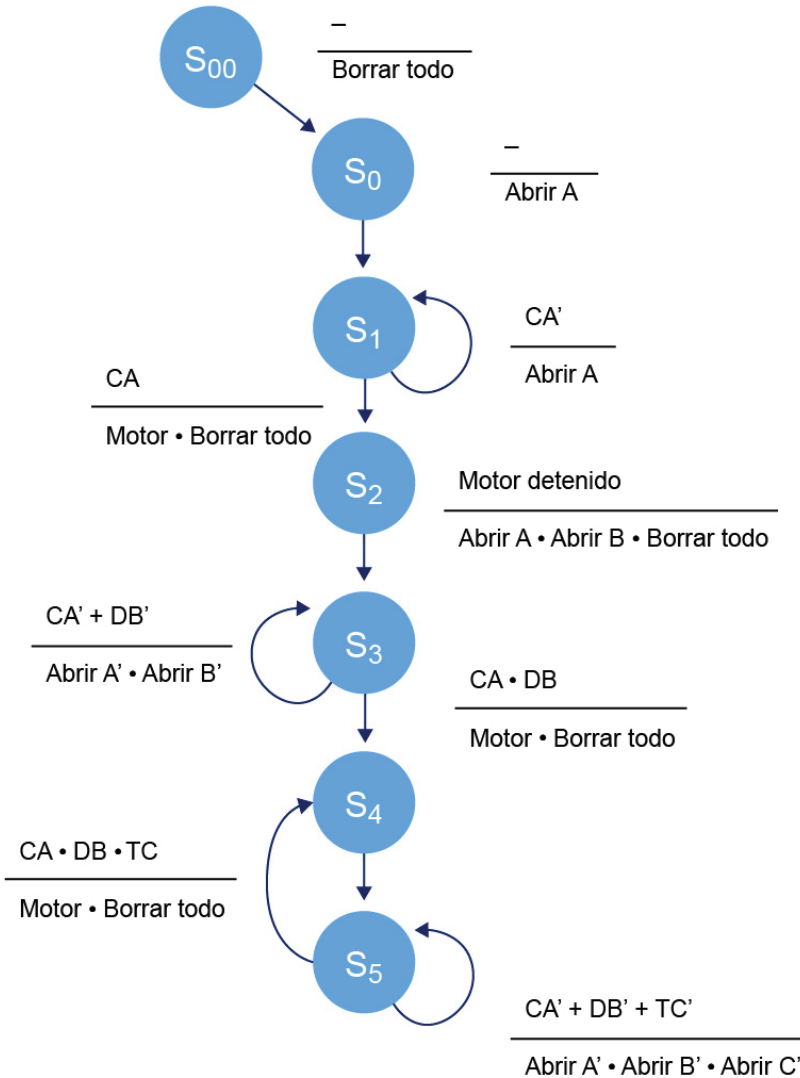


Figura 10.20 Unidad de control incluido en el diseño esquemático para banda transportadora

Solución alterna, sin *latches* y con solo un contador.

©Editorial Digital Tecnológico de Monterrey

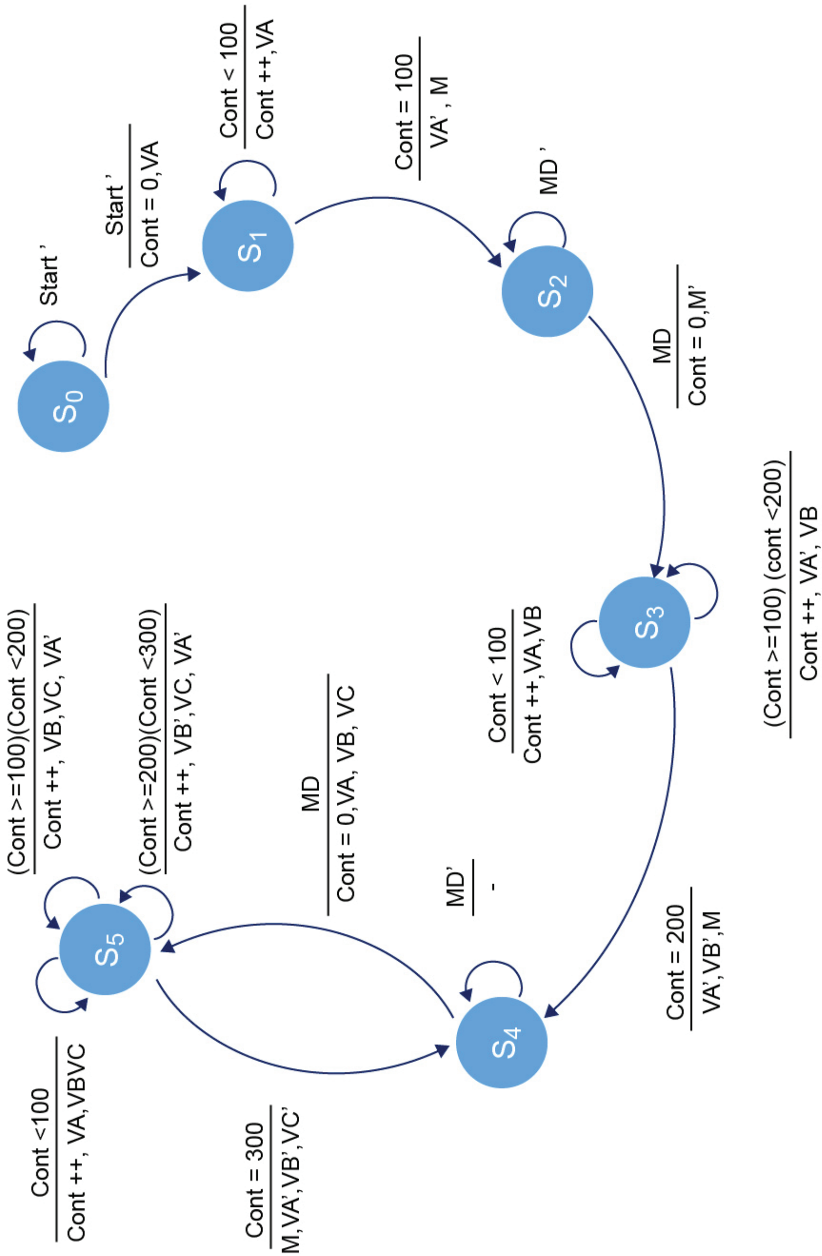


Figura 10.21 Diseño de la unidad de control de banda que muestra las operaciones a realziar sobre un registro

En este problema ya se omite la descripción en VHDL.

©Editorial Digital Tecnológico de Monterrey

10.9 Horno de microondas

En este problema se solicita diseñar un circuito de control de un horno de microondas, similar al que se resolvió en el **capítulo 8**. Las funciones que realizará este controlador se explican a partir de sus entradas y salidas.

Las entradas al sistema son:

Minuto: es un botón que incrementa el contador del tiempo de cocción en 60 segundos.

Marcha: cuando este botón se oprime se inicia la marcha del horno, y no se parará hasta que se abra la puerta o la cuenta llegue al final o bien se oprima el botón de stop_reset.

Stop_Reset: es un botón que si se pulsa con el horno en marcha lo detiene, pero la cuenta conserva su valor Si se pulsa con el horno apagado la cuenta se pone a cero.

Puerta: esta señal proviene de un sensor colocado en la puerta que cuando está a uno indica que la puerta está abierta, y a cero indica que está cerrada.

Clk: es el reloj de entrada con un periodo de 125 ms, es decir, 8 ciclos por segundo= 8Hz.

Las salidas del circuito son:

Segundos (9..O): estas 10 líneas le indican a una pantalla el número de segundos de la cuenta. La codificación es binaria, por lo que se pueden programar hasta 1023 segundos.

Calentar: la salida de uno es para que el horno caliente.

Luz: en uno enciende la luz interna del horno. Esta luz debe estar encendida mientras la puerta esté abierta y mientras el horno esté calentando.

Alarma: en uno suena. Se debe generar esta salida durante 3 segundos cuando la cuenta ha llegado a cero después de que el horno termina de calentar.

©Editorial Digital Tecnológico de Monterrey

A continuación, el diseño de su unidad de control, indicando tanto salidas directas como operaciones sobre los registros auxiliares.

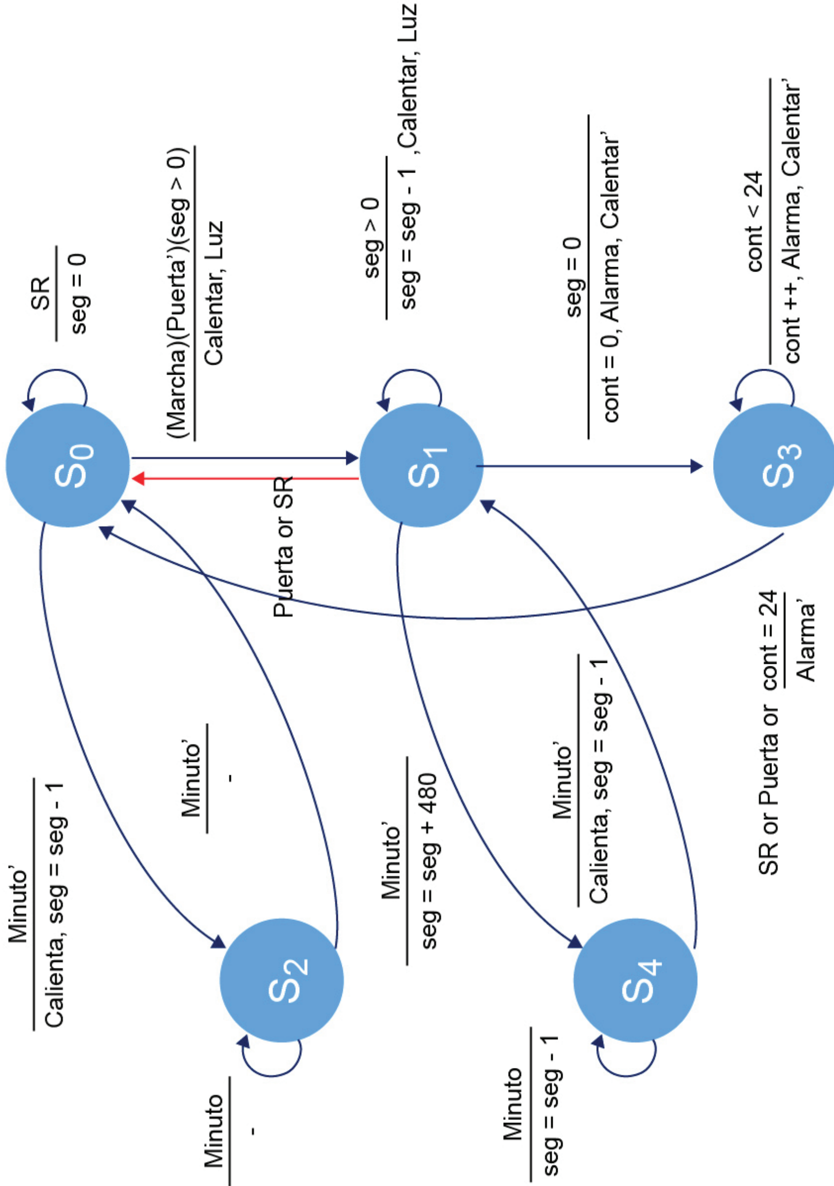


Figura 10.22

©Editorial Digital Tecnológico de Monterrey

10.10 Mensaje luminoso de una columna de LEDs

Se desea desplegar un mensaje iluminando que utilice solamente una columna de 5 LEDs, misma que será barrida por todas las columnas correspondientes a los caracteres que se vayan a desplegar (esto es posible gracias a un fenómeno que ocurre con los ojos, pues guardan memoria de lo que ya se desplegó y se forman los caracteres, aunque se ilumine columna por columna).

Por ejemplo, la palabra LO se desplegaría, en el
tiempo = 0 se despliega en los leds la columna 0
tiempo = 1 se despliega en los leds la columna 1
tiempo = 2 se despliega en los leds la columna 2
etc.

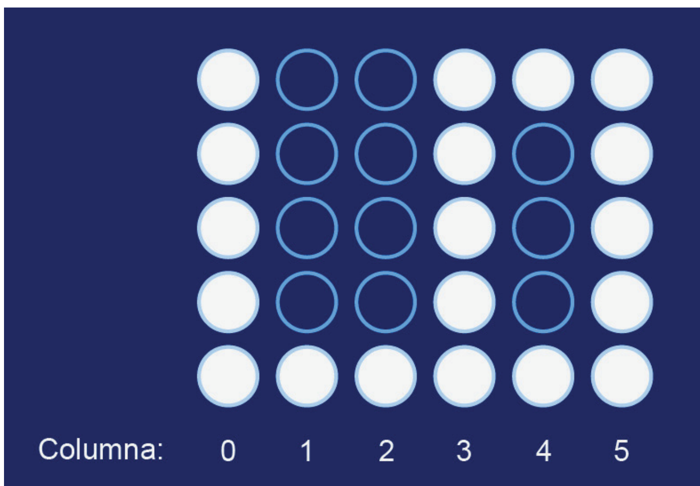


Figura 10.23

El mensaje a desplegar es de 10 caracteres. Se desplegaría del caracter 1 al 10 y luego de nuevo el 1, 2, etc.

Los caracteres están codificados en 8 *bits* (código similar al ASCII).

La configuración de un caracter en LEDs se obtendrá de un componente (ya hecho, solo lo tiene que referenciar) que se describe a continuación:

©Editorial Digital Tecnológico de Monterrey

Configuración de un caracter en la columna de LEDs. Se utilizarán 15 LEDs para desplegar un caracter (3*5), tres columnas de 5.

Suponga que para diseñar el circuito iluminador de LEDs cuenta con la definición de un componente que puede utilizar y que se incluirá como un componente:

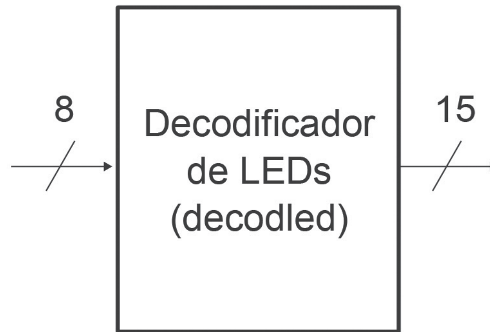


Figura 10.24 Puerto de desplegador de mensajes

Este decodificador recibe el código de 8 *bits* de los caracteres y proporciona la configuración, con 1s y 0s, de los 15 *bits* que se utilizarán para desplegar un caracter columna por columna. Los primeros 5 *bits* corresponden a la primera columna y los últimos 5 a la tercera.

La frecuencia en que se desplegarán columnas en los LEDs está dada por el reloj externo.

La definición de este componente para incluirlo y referenciarlo es la siguiente:

```
Component decodled is
Port(caracter: in std_logic_vector(7 downto 0);
     config: out std_logic_vector (1  to 15));
end component;
```

©Editorial Digital Tecnológico de Monterrey

Suponga que el mensaje de caracteres se encuentra en la siguiente memoria interna al desplegador:

```
type memoria is array (integer range <>, integer range <>) of std_logic;
signal mensaje: memoria (1 to 10, 7 downto 0);
```

El puerto del circuito y la definición de señales auxiliares y componente externo son los siguientes:

```
entity desplegador is
    Port(clk: in std_logic;
         leds: out std_logic_vector(1 to 5));
end desplegador;

architecture Behavioral of desplegador is
type memoria is array (integer range <>, integer range <>) of std_logic;
signal mensaje: memoria (1 to 10, 7 downto 0);
Component decodled is
Port(caracter: in std_logic_vector(7 downto 0);
     config: out std_logic_vector (1  to 15));
end component;
begin
```

©Editorial Digital Tecnológico de Monterrey

El diseño de la máquina de estados de la unidad de control es el siguiente:

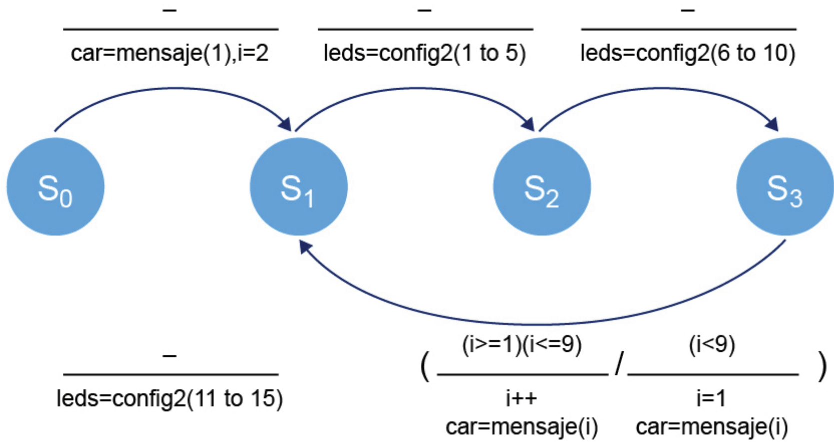


Figura 10.25 Autómata de desplegador de mensajes

Cabe notar que, en este sencillo diseño, a partir del arranque en el estado1 (S1) no debe haber tiempos muertos sin utilizar los LEDs porque se perderá el efecto que se busca al desplegar solo en 5 LEDs. Siempre se debe estar desplegando en los LEDs, por eso en el estado 3 se debe obtener el siguiente caracter, mientras se despliega la última columna del caracter en turno. Si ya se desplegaron los 10 caracteres a continuación se regresa al primero. Note que cuando i = 10 se obtiene el décimo caracter e i regresa a 1.

©Editorial Digital Tecnológico de Monterrey

La codificación del circuito es la siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity desplegador is
    Port(clk: in std_logic;
         leds: out std_logic_vector(1 to 5));
end desplegador;

architecture Behavioral of desplegador is
     type memoria is array(1 to 10) of std_logic_vector(7 downto 0);
     signal mensaje: memoria;
     signal state: integer:=0;
     signal i: integer: = 1;
     signal car: std_logic_vector(7 downto 0);
     signal config2: std_logic_vector(1 to 15);

     Component decodled is
          Port(caracter: in std_logic_vector(7 downto 0);
                      config1: out std_logic_vector (1 to 15));
     end component;
```

©Editorial Digital Tecnológico de Monterrey

```
begin

   process(clk)
   begin
        if clk = '1' and clk'event then
               case state is
                    when 0 => car <= mensaje(i);
                              i <= 2;
                    when 1 => leds <= config2(1 to 5);
                              state <= 2;
                    when 2 => leds <= config2(6 to 10);
                              state <= 3;
                    when 3 => leds <= config2(11 to 15);
                              if (i >= 1) and (i < 9) then
                                  i <= i + 1;
                                  car <= mensaje(i);
                                  state <= 1;
                              elsif i > 9 then
                                  i <= 1;
                                  car <= mensaje(i);
                                  state <= 1;
                              end if;
                    when others => null;
               end case;
        end if;
     end process;

Et1: decodled port map(car,config2);
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

10.11 Máquina despachadora de café

Un fabricante de máquinas despachadoras de café nos pide implementar el circuito de una máquina de café. La máquina cuenta con un tablero de cinco botones que determinan el producto a servir, llamados B1, B2, B3, B4 y enter, como se muestra a continuación:

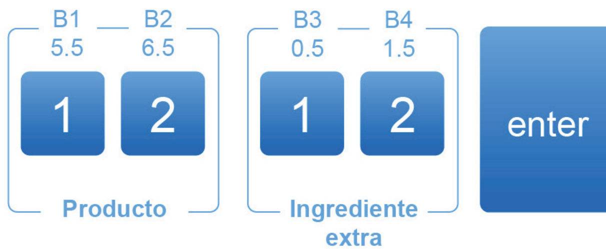


Figura 10.26 Descripción del problema de despachadora de café

Como información al margen, los productos son:

1. Café capuchino

2. Café moka

Los ingredientes extras son:

1. Azúcar

2. Sustituto de azúcar

El costo de cada opción se encuentra señalado en la parte superior de cada botón.

©Editorial Digital Tecnológico de Monterrey

Por otra parte, la máquina cuenta con dos luces, cada una indica al usuario la siguiente operación a realizar:

- L1: “seleccione el producto”
- L2: “seleccione ingredientes extras, al terminar pulse enter” (puede pulsar varias veces cualquiera de los botones, e incluso el mismo botón).

La operación a implementar para la máquina es la siguiente:

- Encender L1.
- Capturar: Producto.
- Encender L2.
- Capturar: los ingredientes extra hasta Enter. Tenga cuidado en procesar solo una vez el ingrediente por cada vez que se oprime un botón. Sería un grave error procesar más de una vez por cada vez que se oprime el botón.
- Apagar L2.
- Desplegar Costo Total en los displays el registro T.
- Lo que sigue por explicar es:
- Pagar con botones B5, B6 y B7.
- Encender L3 e iluminar la cantidad que representa el cambio a devolver usando de nuevo T.
- Devolver (simulando las monedas con el encendido de indicadores).
- Apagar L3.

©Editorial Digital Tecnológico de Monterrey

El usuario insertará monedas para pagar su producto. Para simular este mecanismo utilizaremos otros tres botones. Cada botón representará la moneda que se muestra en la parte superior de la siguiente figura:

Figura 10.27 Descripción del problema

El usuario oprimirá botones para pagar lo que eligió. El costo total que se está desplegando se actualizará a medida que se inserten monedas. Cuando ya haya llegado a la cantidad o la haya sobrepasado, se deberá desplegar cuál es su cambio; lo anterior se hará desplegando en los *displays* de nuevo el registro T.

Para devolver el cambio habrá que accionar mecanismos que suelten monedas.

Con M1 se suelta una moneda de 10 pesos.

Con M2 se suelta una moneda de 1 peso.

Con M3 se suelta una moneda de 0.5 pesos.

Las monedas se deben soltar con un espacio de tiempo de 1 segundo entre una moneda y la siguiente, y esto debe ser controlado por el circuito. Primero se deben soltar las monedas de $10, luego las de $1 y finalmente las de $0.5.

Maneje todos sus números como si fueran enteros, es decir, multiplique por 2 todos sus números. Por ejemplo: 6.5 represéntelo como 1101 (en vez de 110.1); 0.5 represéntelo como 1. Maneje todas sus operaciones en binario e instancie el componente

©Editorial Digital Tecnológico de Monterrey

ConvierteyDespliega(Registro), el cual se encarga de convertir a BCD el valor del Registro, así como de iluminar este valor en *display*s de 7 segmentos, iluminando el punto en el lugar apropiado y asumiendo que el *bit* menos significativo del valor del registro es fraccional.

Se cuenta con un reloj de 2 Hz.

No se implementará la parte que se dedique a servir el café solicitado.

La máquina de estados con todos sus detalles respetando los nombres definidos para todos los elementos se presenta a continuación.

Nota: en la máquina de estados se mostrarán solo cantidades enteras, como 13 (en vez de 6.5), 11 (en vez de 5.5), 1 (en vez de 0.5), etc., suponiendo siempre que el *bit* menos significativo representa la parte fraccional. Como en este problema 0.5 es la única opción de parte fraccional, con un *bit* es suficiente para representarla. Así que en el diagrama de estados todas las cantidades estarán recorridas un *bit* a la izquierda, es decir, multiplicadas por 2.

En la solución que se presenta a continuación se están pasando por *one shots* las señales producidas por los botones, de esa forma solo estarán activas durante un ciclo de reloj. Si no se utilizan los *one shots* se requerirían estados adicionales para esperar que los botones bajen a cero.

Así mismo, si lo que se ilumina es lo que se encuentra en el registro T, entonces primero se iluminará el costo total del producto y después se iluminará el cambio que se debe regresar en monedas. Luego se irá iluminando lo que resta de cambio al estar en proceso la devolución de monedas, hasta que este monto quede en ceros.

Para la devolución del cambio se utiliza un estado auxiliar (S4, S5, S6) para que M1, M2 y M3 estén en 1 durante un segundo, ya que state permanece en cada estado durante medio segundo, dado que la frecuencia del reloj es de 2kHz.

©Editorial Digital Tecnológico de Monterrey

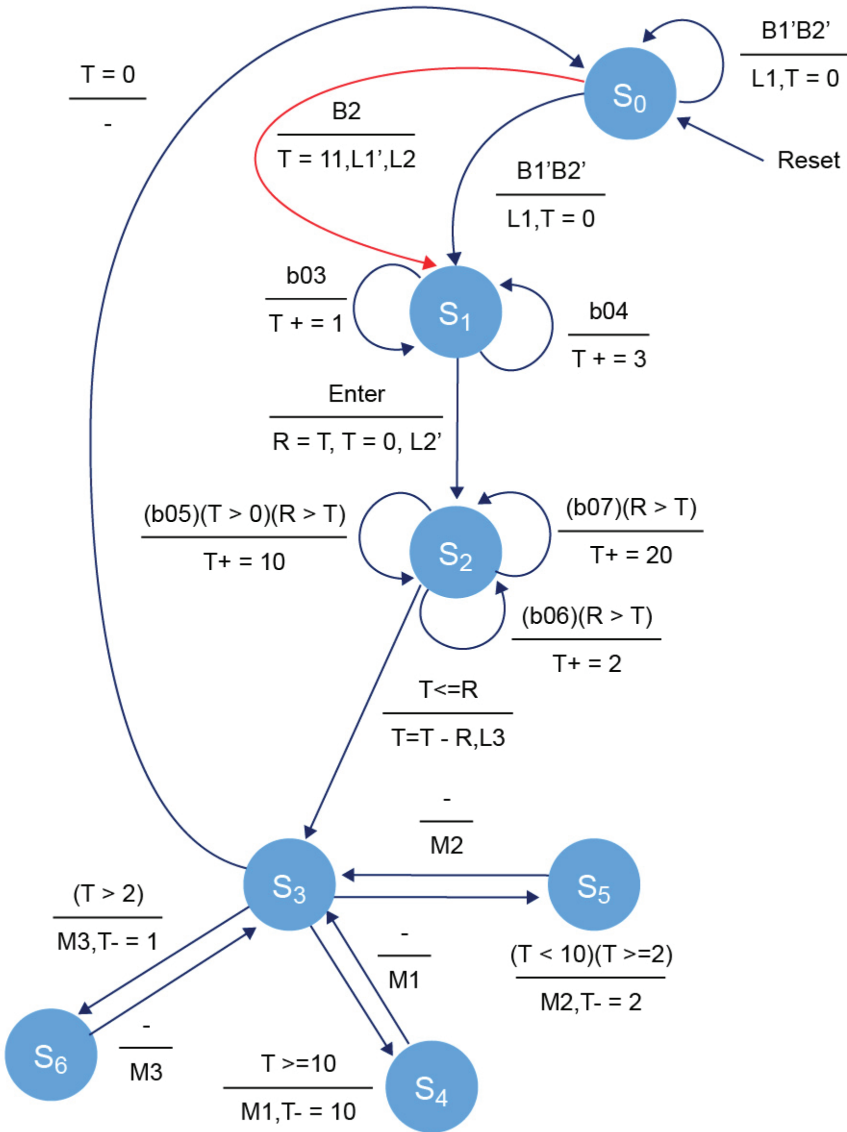


Figura 10.28 Unidad de control

©Editorial Digital Tecnológico de Monterrey

```
entity CAFE is
    Port ( Clk, B1,B2,B3,B4,Enter,B5,B6,B7 : in  STD_LOGIC;
               L1,M1,M2,M3 : out  STD_LOGIC);
end CAFE;

architecture Behavioral of serial is
component ConvierteyDespliega is
   Port (Reg: out std_logic_vector (7 downto 0));
end component;

component one_shot is
   Port (Clk, X,: in std_logic ;
      Z: out std_logic);
end component;


signal  T, R : std_logic_vector (7 downto 0);
signal cont : std_logic_vector (7 downto 0);
signal state : integer;
signal  bo3, bo4, b05, b06, b07: std_logic;

begin

B03: one_shot port map (clk, b3,bo3);
B04: one_shot port map (clk, b4,bo4);
B05: one_shot port map (clk, b5,bo5);
B06: one_shot port map (clk, b6,bo6);
B07: one_shot port map (clk, b7,bo7);

r1: ConvierteyDespliega port map(T);

sm: process (clk) begin
     if  Reset='1' then T<="00000000";
          L1<= '0';L2<='0';L3<='0';
          M1<= '0';M2<='0';M3<='0';
          clk = '1' and clk'event then
```

©Editorial Digital Tecnológico de Monterrey

```
        case state is
             when 0 => if b1 = '1' then
                        T <= "00001011";
                        L2 <= '1';
                        L1 <= '0';
                        state <= 1;
                         elsif b2 = '1' then
                        T <= "00001101";
                        L2 <= '1';
                        L1 <= '0';
                        state <= 1;
                  else  L1 <= '1';
                         T <= "00000000";
                        end if;
             when 1 => if Bo3 = '1' then
                        T <= T + "00000001";
                        state <= 1;
                        L2 <= '1';
                   elsif Bo4 = '1' then
                        T <= T + "00000011";
                        state<=1;
                        L2<='1';
                  elsif enter = '1' then
                        R <= T;
                        T <= "00000000";
                        state <= 2;
                        L2 <= '0';
                  else null;
                  end if;

             when 2 => if (b05 = '1') and (T > 0) and (R>T) then
                        T <= T + "00001010";
                         elsif (b6 = '1') and (R > T) then
                        T <= T + "00000010";
                        cont <= cont + "00000010";
                  elsif (b7 = '1') and (R > T) then
                        T <= T - "00010100";
```

©Editorial Digital Tecnológico de Monterrey

```
                                              elsif T >= R  then
                              T <= T - R;
                              L3 <= '1';
                              state <= 3;
                       else null;
                       end if;
    when 3 => if T < "00000010" then
                       state <= 6;
                       T <= T-1;
                       M3 <= '1'; M1<='0';M2<='0';
                       elsif T >= "00001010" then
                       M1 <= '1'; M2<='0';M3<='0';
                       T <= T - "00001010";
                       state <= 4;
                       elsif(T < "00001010") and (T >= "00000010")
then
                       M2 <= '1'; M1 <= '0';M3 <= '0';
                       T <= T - "00000010";
                       state <= 4;
                       else T = "00000000" then
                       M2 <= '0'; M1 <= '0';M3 <= '0';
                       state <= 4;
                       else null;
          when 4 => state <= 3;
                       M1 <= '1'; M2 <= '0';M3 <= '0';
          when 5 => state <= 3;
                       M1 <= '0'; M2 <= '1';M3 <= '0';
          when 6 => state <= 3;
                       M1 <= '0'; M2 <= '0';M3 <= '1'
                  end if;
    when others => state <= 0;
    end case;
  else null;
  end if;
end process;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

10.12 Ensamble de juguetes

Un proceso de ensamble de juguetes lleva dos pasos: armado y pintura. El armado tiene un activador que recibe una señal A que debe mantenerse en 1 durante 1 minuto. La pintura tiene un activador que debe recibir una señal P que debe mantenerse en 1 durante 3 minutos. El dispositivo que aplica la pintura empuja el juguete hacia afuera dentro de los 3 minutos que toma el proceso. Hay un robot que traslada un juguete de un paso al otro, es decir, recoge un juguete armado y lo coloca en el dispositivo de pintura. Para funcionar el robot debe recibir una señal R que no requiere estar en 1 todo el tiempo que dure en traslado, una señal de 1 minuto es suficiente. El robot envía una señal F cuando termina el traslado. El traslado toma entre 1 minuto con 45 segundos y 1 minuto 50 segundos, esta variabilidad se debe a que en ocasiones el robot encuentra obstáculos que debe franquear (ya que es un proceso semi-automatizado). Las fases de colocar un juguete en el paso de armado y la de recoger el juguete ya pintado son manuales (realizadas por una persona). Se cuenta con un reloj externo que tiene una frecuencia de 1 ciclo/minuto.

Para este problema se muestra solamente el diseño del diagrama de estados de la unidad de control, de manera que se maximice la cantidad de juguetes a producir.

©Editorial Digital Tecnológico de Monterrey

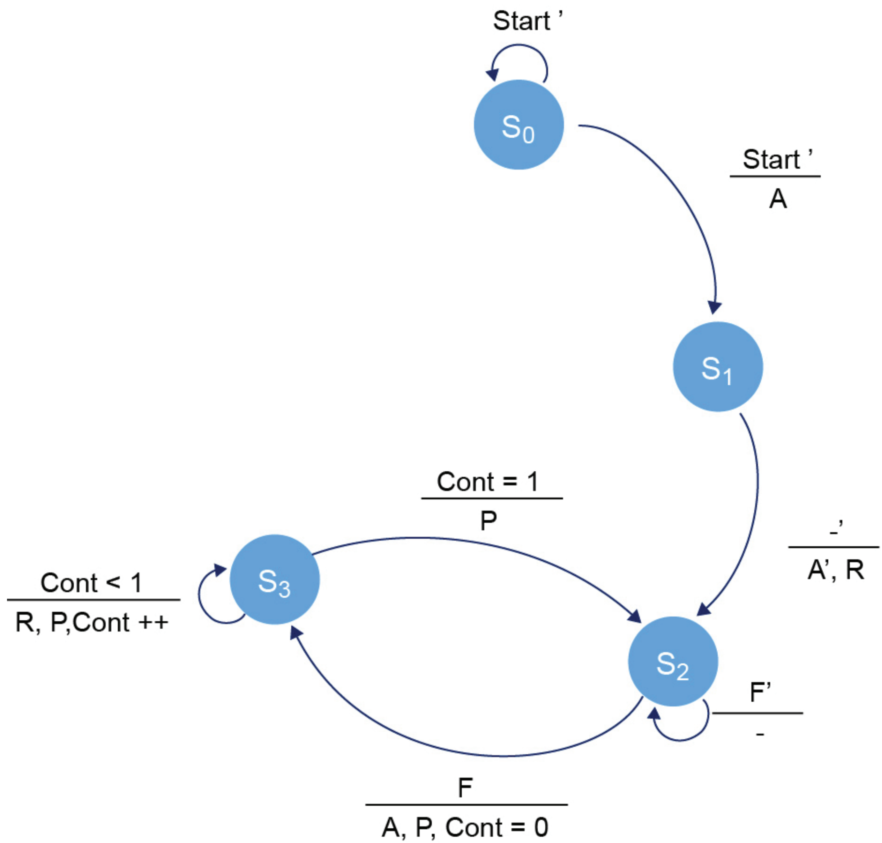


Figura 10.29 Autómata de ensamble de juguetes

En el diagrama solo se indican las señales de salida que están en 1, el resto estaría en 0. Con este diseño se tendrá un juguete listo cada tres minutos. Se debe notar que lo anterior se logra acomodando un juguete para la etapa de pintura al mismo tiempo que otro juguete para la etapa de ensamblado, solo en el arranque hay un solo juguete. Al manejar dos juguetes al mismo tiempo se optimiza el tiempo de producción. No se está dando tiempo a la puesta de un nuevo juguete en el tiempo de ensamble porque no se especificó en el problema, se está suponiendo que el tiempo de colocación ya está contemplado en el minuto de ensamble.

©Editorial Digital Tecnológico de Monterrey

Como se ve, el diseñador de un circuito de control de un proceso también debe involucrarse en el proceso mismo para que el resultado del control sea el óptimo. Se presenta una respuesta de muchas posibles, ya que el número de estados puede variar dando el mismo resultado de un juguete cada tres minutos.

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 10

Pruebe en el simulador e implemente en el laboratorio los circuitos de los siguientes problemas (no se presentarán las respuestas):

1. Minicalculadora

Diseñe un circuito para una calculadora que cuenta con las siguientes entradas:

- Una señal que viene de un botón0 para el dígito 0.
- Una señal que viene de un botón1 para el dígito 1.
- Una señal que viene de un botón2 para el dígito 2.
- (Omitimos el resto de los dígitos para recortar el problema)
- Una señal que viene de un botónA para la función +
- Una señal que viene de un botónS para la función -
- Una señal que viene de un botónE para la función =

La operación de la calculadora es la siguiente:

a) El usuario puede oprimir de uno a cuatro dígitos para formar el primer operando (que es un número BCD).

(Nota 1: al oprimir una tecla se reconoce solo un dígito, si se desea escribir de nuevo el mismo dígito se debe soltar la tecla y volverla a oprimir, es un gran error registrar más de una vez un dígito al oprimir su botón.

©Editorial Digital Tecnológico de Monterrey

Nota 2: si se oprime un quinto dígito ya no se toma en cuenta).

b) Se sabe que terminó de "escribir" el número al seleccionar una de dos funciones + o –.

c) De nuevo puede oprimir de uno a cuatro dígitos para formar el segundo operando (otro número BCD).

(Nota 1: al oprimir una tecla se reconoce solo un dígito, si se desea escribir de nuevo el mismo dígito se debe soltar la tecla y volverla a oprimir, es un gran error registrar más de una vez un dígito al oprimir su botón.

Nota 2: si se oprime un quinto dígito ya no se toma en cuenta).

d) Se sabe que terminó de "escribir" el número al seleccionar la tecla =.

e) Luego de oprimir = se despliega el resultado de la suma o la resta, según sea el caso. El resultado se despliega en LEDs en BCD.

Se cuenta con dos componentes ya hechos para instanciar.

- SUMABCD (operando1, operando2, resultado)
- RESTABCD(operando1, operando2, resultado)

Los operandos son de 16 *bits* y el resultado que genera es de 20 *bits*.

Considere que el reloj es mucho más lento que el rebote de los botones.

©Editorial Digital Tecnológico de Monterrey

Para este problema:

A) Diseñe la unidad de control.

B) Codifique en **VHDL**.

b.1) Los tres primeros estados

b.2) Los dos últimos estados

b.3) Las instancias a SUMABCD y a RESTABCD

El puerto se describe a continuación.

```
entity calculadora is
 port (boton1, boton2, boton3, botonA, botonS, botonS, clk: in
std_logic;
resultado; out std_logic_vector(19 downto 0));
 end calculadora;

 architecture behavioral of calculadora is
 component SUMABCD is
 port (operando1, operando2: in std_logic_vector(15 downto 0);
resultado: out std_logic_vector(19 downto 0));
 end component;
 component RESTABCD is
 port (operando1, operando2: in std_logic_vector(15 downto 0);
resultado: out std_logic_vector(19 downto 0));
 end component;
 SIGNAL
 Begin

 End behavioral;
```

©Editorial Digital Tecnológico de Monterrey

2. Banda transportadora II

Se requiere llenar frascos con 3 diferentes tipos de pastillas. Cada una de estas pastillas está almacenada en tres recipientes diferentes, denominados A, B y C. El llenado debe realizarse en orden, es decir, cada frasco recibe primero las pastillas del recipiente A, posteriormente el B y finalmente el C, tal y como se observa en la siguiente figura.

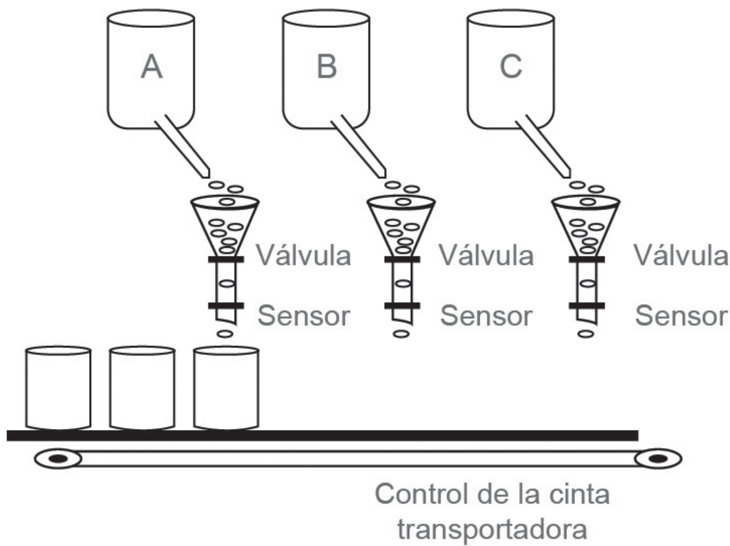


Figura 10.30

En cada frasco deben introducirse 10 pastillas de cada tipo, por lo que el alimentador de pastillas de cada recipiente debe detenerse (cerrando la válvula correspondiente) cuando su sensor ya haya detectado el paso de 10 pastillas. Los tres tipos de pastillas son diferentes, así que hasta que los tres sensores hayan detectado el paso de sus 10 pastillas se deberá enviar una señal de un 1 lógico al motor de la banda, para que esta desplace los frascos hasta el siguiente alimentador de pastillas, el cual repite el proceso anterior. Solo habrá que controlar esta parte del proceso, el resto, como el cerrado de los frascos, es llevado a cabo de forma manual. Se cuenta con un reloj externo de 100 Hz.

©Editorial Digital Tecnológico de Monterrey

Notas:

1) El proceso inicia operando solo al primer alimentador, luego al primero y al segundo, y luego funciona para una secuencia infinita de frascos, es decir, no diseñe para la condición final (no haga caso de que queden uno o dos frascos porque sea el final del proceso).

2) Diseñe el *hardware* necesario para que el motor reciba un 1 para avanzar y desplazar los frascos al siguiente alimentador de pastillas. Cuando la banda se detiene el controlador del motor envía una señal indicando que está detenido.

3) Suponga que al recibir un 1 lógico una válvula se abre y permanece abierta hasta que recibe un cero lógico. Diseñe la señal necesaria para abrir o cerrar la válvula.

4) Cada sensor detecta el paso de una pastilla y genera un 1 al paso de una pastilla. La duración de esta señal no es precisa.

5) Se requiere diseñar el *hardware* que genere el 1 que necesita el motor para avanzar, y esto es hasta que los tres sensores hayan detectado el paso de 10 pastillas.

3.Iluminación de mensaje en tablero de LEDs

Se desea desplegar un mensaje iluminando en un tablero que cuenta con 24x5 LEDs, que puede desplegar hasta 8 caracteres. Se le pide diseñar el circuito que accesa el mensaje de una memoria

©Editorial Digital Tecnológico de Monterrey

(ya definida por algún otro componente que no se describe en este problema) y lo despliega en el tablero.

- El mensaje a desplegar es de 10 caracteres.
- Se desea que el mensaje se observe recorriéndose a la derecha, para esto se desplegaría del carácter 1 al 8 y luego del 2 al 9, del 3 al 10, del 4 al 1, del 5 al 2, del 6 al 3, del 7 al 4, del 8 al 5, del 9 al 6, del 10 al 7, del 1 al 8, y así sucesivamente, aunque los caracteres pueden ser desplegados parcialmente porque aparecen y desaparecen columna a columna.
- El código que representa cada carácter es de 8 *bits*, es un código como el ASCII.
- La configuración de un carácter en LEDs se obtiene de un componente (ya hecho, solo lo tiene que referenciar)
- Una memoria denominada mensaje de tipo ROM aloja el mensaje de 10 caracteres.

4. Configuración de un carácter en LEDs (componente decodled):

Se utilizarán 15 LEDs para desplegar un carácter en tres columnas de 5 LEDs:

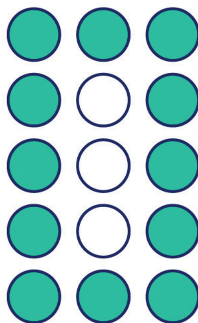


Figura 10.31

©Editorial Digital Tecnológico de Monterrey

El decodificador recibe el código de 8 *bits* de los caracteres y proporciona la configuración, con 1s y 0s, de los 15 *bits* que se utilizarán para desplegar un carácter columna por columna (primero la primera columna, luego la segunda, luego la tercera).

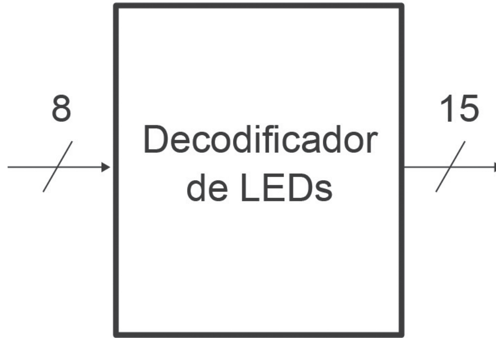


Figura 10.32

La definición de este componente para incluirlo y referenciarlo es:

```
Component decodled is
Port ( caracter: in std_logic_vector(7 downto 0);
config: out std_logic_vector (1 to 15));
end component;
```

Memoria que almacena el mensaje:

Suponga que el mensaje de los códigos de los caracteres se encuentra en la siguiente memoria interna al desplegador:

signal mensaje : array (1 to 10) of std_logic_vector (7 downto 0):=("10001100","11110000", "01010101" ….);

Conexión del tablero:

Conectar todos los LEDs del tablero requiere de un circuito 120 salidas, así que del FPGA solo salen los primeros 15 *bits*. Hay un decodificador externo al FPGA, que distribuye la configuración a los

©Editorial Digital Tecnológico de Monterrey

8 segmentos de 15 *bits* del tablero, así como 3 *bits* que corresponden al número de carácter, del 0 al 7, que se está desplegando en un momento dado. El decodificador sí tiene conexión a los 120 LEDs del tablero, es decir, a los 8 caracteres que se pueden desplegar.

Esta conexión se muestra a continuación:

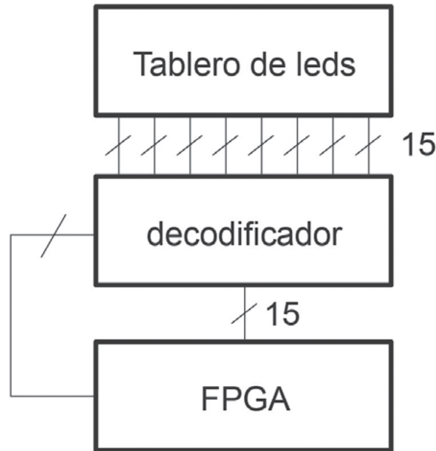


Figura 10.33

Los 15 *bits* que salen representan, en orden, las tres columnas a iluminar.

La señal de reloj que se utiliza es de 1 MHz.

Procedimiento a seguir:

1) Para facilitar el despliegue de los caracteres, es recomendable almacenar en una memoria con dimensiones (30x5).

Signal RAM: array (1 to 30) of std_logic_vector (1 downto 5)

la configuración por columnas de los 10 caracteres a desplegar.

©Editorial Digital Tecnológico de Monterrey

2) Algoritmo de barrido

Mostrar las columnas 1 a 3, durante un ciclo de reloj, luego de la 4 a la 6, durante un ciclo de reloj, luego de la 7 a la 9, durante un ciclo de reloj, y así hasta 22 a la 24, durante un ciclo de reloj.

Después de estos 8 ciclos, se repite estos pasos (columnas 1 a 24) 1000,000 de veces (un segundo).

Al terminar, se avanza una columna y se despliegan las columnas 2 a 4, durante un ciclo de reloj, luego de la 5 a la 7, durante un ciclo de reloj, luego de la 8 a la 10, durante un ciclo de reloj, y así hasta 23 a la 25, durante un ciclo de reloj.

Después de estos 8 ciclos, se repite el despliegue de las columnas 2 a 25, 1000,000 de veces.

Al terminar se avanza una columna y se despliega hasta la columna 26 y así se continúa hasta llegar a la columna 30.

Al terminar de desplegar de las columnas 7 a la 30 se regresa a desplegar las columnas 1 a 24.

En total, la máquina completa debe tener no más de cinco estados.

Para este problema:

a) Muestre el diseño de la unidad de control y explique con claridad qué función tiene cada registro que intervenga en el diseño. Es necesario considerar que en todo momento se están iluminando LEDs, no debe haber ramas de sobra.

b) Ya sea en el diagrama de estados o en un lado de este, señale los accesos a las memorias.

c) También señale a un lado del diagrama de estados la referencia a decodled.

©Editorial Digital Tecnológico de Monterrey

La entidad del circuito y de los componentes internos se muestran a continuación.

```
entity desplegador is
Port ( clk : in std_logic;
        leds : out std_logic_vector(1 to 15);
        dirección: out std_logic_vector(2 to 0));
end desplegador;

architecture Behavioral of desplegador is
signal mensaje : array (1 to 10) of std_logic_vector (7 downto
0):=("10001100","11110000", "01010101" ….);
signal RAM: array (1 to 30) of std_logic_vector (1 downto 5);
Component decodled is
Port ( caracter: in std_logic_vector(7 downto 0);
config: out std_logic_vector (1 to 15));
end component;
begin …
end Behavioral;
```

5. Iluminación de *displays* de LEDs

En este problema diseñaremos un circuito para manejar un tablero formado por 16 *displays* de LEDs, como el que se muestra en la siguiente figura:

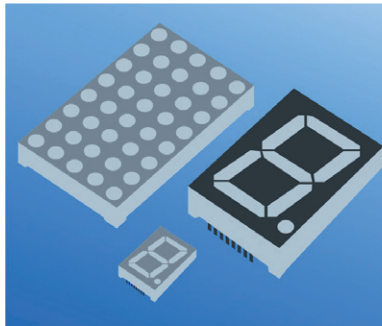


Figura 10.34

©Editorial Digital Tecnológico de Monterrey

En el tablero los *displays* no están interconectados. Un *display* tiene 40 entradas, cada una conectada a cada LED para indicar cuáles se prenden; también tiene una entrada de CS=Chip Select, que en caso de ser 0 haría que no se prendiera ninguno.

Nota: haga caso omiso de los *displays* de 7 segmentos que aparecen en la figura.

Se le solicita diseñar un controlador de tablero de mensajes, que cuenta con las siguientes conexiones:

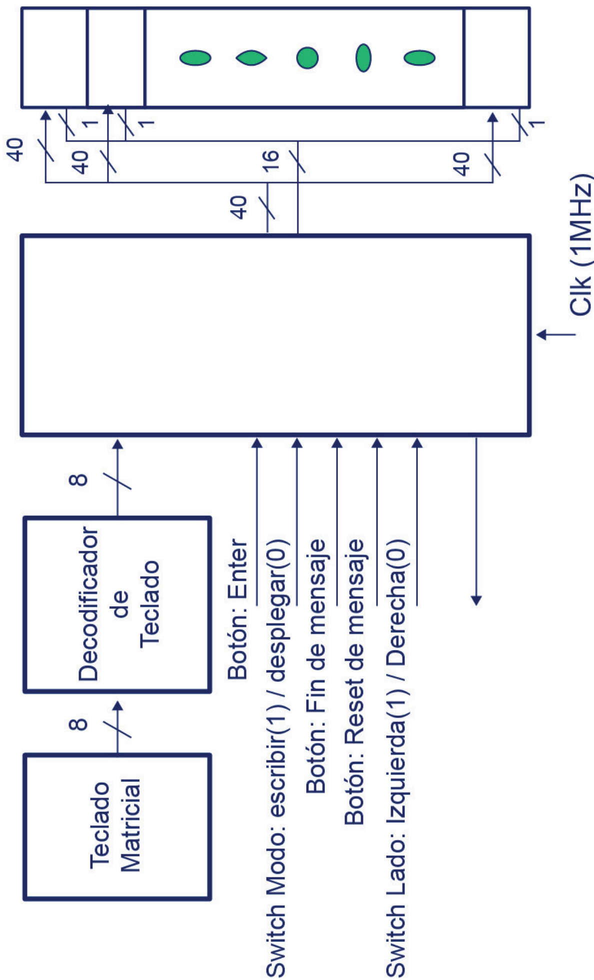


Figura 10.35

©Editorial Digital Tecnológico de Monterrey

La última salida corresponde a una señal de tope de mensaje.

Nota: Lo único que se debe diseñar es el circuito sombreado en gris.

La operación del controlador es la siguiente:

Se escribe un mensaje utilizando un teclado matricial, pero no se debe diseñar el decodificador de teclado. Suponga que ya cuenta con este circuito que proporciona 8 *bits* con un código para los caracteres, estos 8 *bits* son los que ingresan al controlador.

1. Para escribir se oprime una tecla, luego se oprime <enter>, luego otra tecla y así sucesivamente hasta que se oprime el botón de <Fin de mensaje>. El límite de caracteres del mensaje es de 64. Si se llegan a oprimir los 64 caracteres y no se ha oprimido <Fin de mensaje> entonces se enciende un LED de “tope de mensaje”. Un mensaje puede ser borrado si el *switch* de modo está en 1 y se oprime el botón de <Reset de mensaje>.

Para desplegar un mensaje, el *switch* de modo se baja a 0 y se selecciona un lado, para que el mensaje se desplace hacia la izquierda o hacia la derecha.

6. Configuración de un carácter en el tablero de LEDs:

Se utilizarán 40 LEDs para desplegar un carácter (8*5).

Suponga que para diseñar el controlador cuenta con la definición de un componente que puede utilizar y que se incluirá como un elemento interno al controlador:

©Editorial Digital Tecnológico de Monterrey

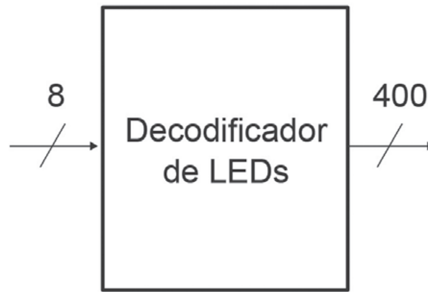


Figura 10.36

Este decodificador recibe el código de 8 *bits* de los caracteres y proporciona la configuración, columna tras columna, de los 40 *bits* que se utilizarán para desplegar un carácter en el tablero.

Utilice la siguiente definición para referenciar este componente:

Component decled is

Port (caracter: in std_logic_vector(7 downto 0); config: out std_logic_vector(0 to 39)); end decled;

Conexión del controlador al tablero de LEDs:

La salida del controlador consta de 40 *bits* para configurar un caracter (observar figura) y 16 *bits* para seleccionar uno de los 16 *displays* de leds (observar figura). Ya que solo se puede desplegar un caracter a la vez, se debe "refrescar" el tablero con la frecuencia dada por el reloj externo, que es de 1 MHz. Con esta frecuencia, parecerá que todo el tablero está encendido a la vez.

Desplazamiento de los caracteres:

El sentido del desplazamiento se determina por el *switch* de lado.

Los caracteres deben desplazarse cada medio segundo, tiene dos opciones, que se desplacen recorriéndose columna a columna (sería lo mejor) o bien, simplificar el problema y que se recorran moviendo caracteres completos (se verá raro, pero se considerará correcto para efectos del examen).

©Editorial Digital Tecnológico de Monterrey

Memoria interna:

Para almacenar la secuencia de caracteres, utilice una memoria interna:

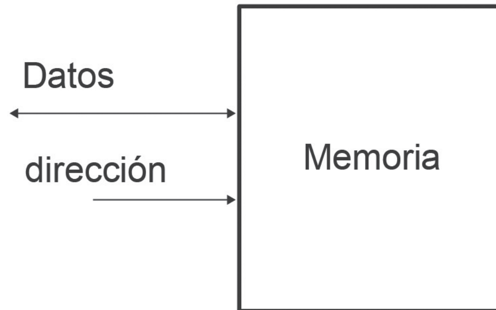


Figura 10.37

En **VHDL** puede definir esta memoria interna de la siguiente manera:

Signal Memoria: std_logic_vector(0 to 63, 7 downto 0);

Puede hacer referencia a esta memoria con solo un subíndice, por ejemplo Memoria(5), en cuyo caso se está señalando a los 8 *bits* de la dirección 5.

Es posible definir cualquier cantidad de memorias internas (si requiere más de una).

Para este problema:

a) Defina el puerto del controlador en **VHDL**.

b) Diseñe el controlador en forma esquemática, en base a elementos básicos (como registros, mux's, memorias, etc.) indique claramente todas las consideraciones de diseño que haga.

©Editorial Digital Tecnológico de Monterrey

c) Diseñe la unidad de control (si la va a utilizar en el siguiente inciso).

d) Diseñe el controlador en **VHDL** consistentemente con los incisos anteriores (es requisito para este inciso).

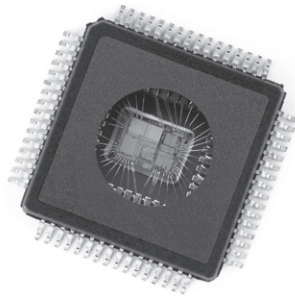
©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 10

Un circuito secuencial basado en una unidad de control constituye el tipo de diseño más poderoso en circuitos digitales. Este tipo de modelo es el que tienen los microprocesadores, procesadores de señales, de imágenes, entre otras aplicaciones.

En este capítulo se cubrieron una vasta diversidad de ejemplos para que el lector desarrolle su habilidad de construir una unidad de control. Con la práctica es posible dominar esta técnica.



©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 11. Memorias

Memorias	
Introducción	Más sobre memorias
Memorias ROM	Definición de una memoria
Ejercicio	Memoria en el FPGA
Memorias RAM	SRAM

©Editorial Digital Tecnológico de Monterrey

11.1 Introducción

Una memoria es un componente en el cual es posible almacenar y acceder datos. Cada dato puede tener una longitud de m *bits*; m es el ancho de la memoria en *bits*. A un dato almacenado en memoria también se le llama palabra.

Cada dato en una memoria se encuentra en una posición, a esta posición se le llama dirección.

Por ejemplo, esta es la representación de una memoria de 2 x 3 *bits* (palabras de 3 *bits*):

Dirección	Contenido
0 0	1 0 1
0 1	1 0 0
1 0	0 0 0
1 1	1 1 1

Figura 11.1 Contenido de una memoria

©Editorial Digital Tecnológico de Monterrey

11.2 Memorias ROM

Hay **memorias** con un contenido de datos fijo que solo se utilizan para lectura, estas se llaman **ROM** por sus siglas en inglés (*Read Only Memory*). El circuito de la memoria mostrada podría ser implementado de la siguiente manera:

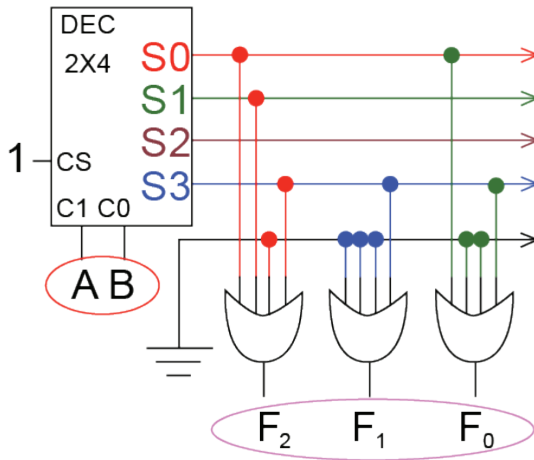


Figura 11.2 Ejemplo de circuito combinacional almacenado en una memoria

Lo cual nos indica que podría ser construido en los LUTs (look up table) del FPGA, como circuitos combinacionales. Sin embargo, los LUTs están distribuidos en los CLB del FPGA. Si la memoria es pequeña, digamos, de menos de 64 direcciones, cabe en un LUT; pero si es más grande, el sistema de desarrollo la distribuiría en diversos LUTs, haciendo más lento su acceso. Dado que hay sistemas digitales cuyo diseño se facilita con el apoyo de una memoria, en los FPGA se incorporaron bloques de memoria de acceso rápido, tan rápido como un circuito combinacional, y que pueden implementarse sin requerir su partición.

La forma más sencilla de describir una memoria en VHDL para que el sistema de desarrollo la reconozca y la implemente en un bloque de memoria es a través de definir un arreglo (array). Un arreglo debe ser definido con los límites inferior y superior de direcciones y se debe especificar su contenido (el formato de sus palabras). La dirección debe tener tipo entero.

©Editorial Digital Tecnológico de Monterrey

Como ejemplo se ilustra un problema cuyo objetivo es iluminar cuatro LEDs con la secuencia que está almacenada en una ROM de 16x4 (16 direcciones diferentes, es decir, 16 posiciones de memoria y 4 *bits* de ancho). La velocidad de cambio de iluminación estaría determinada por un reloj.

Dirección	Ancho (3 – 0)
0000	1001
0001	0001
0010	0111
0011	1001
0100	0001
0101	0111
0110	1001
0111	0001
1000	0111
1001	1001
1010	0001
1011	0111
1100	1001
1101	0001
1110	0111
1111	0000

Esta memoria ROM, en VHDL puede ser definida de la siguiente manera:

```
type rom_type is array (0 to 15) of std_logic_vector (3 downto 0);

constant Memo1:rom_type:=
("1001", "0001", "0111", "1001", "0001", "0111", "1001",
"0001", "0111", "1001", "0001", "0111", "1001", "0001",
"0111", "0000");
```

©Editorial Digital Tecnológico de Monterrey

Como el contenido de la ROM no cambia, entonces puede ser definido como una constante.

El circuito esquemático del problema es el siguiente:

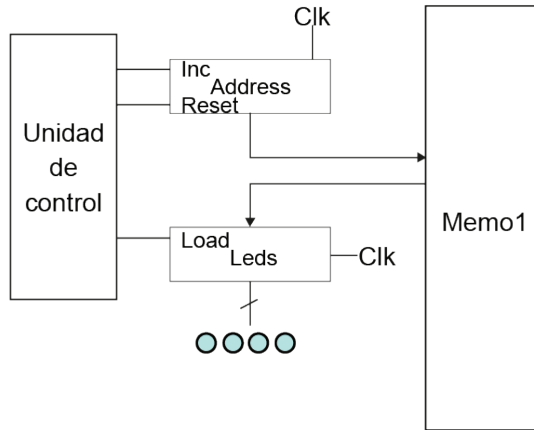


Figura 11.3 Diseño esquemático de lectura de memoria

El diseño de la unidad de control es el siguiente:

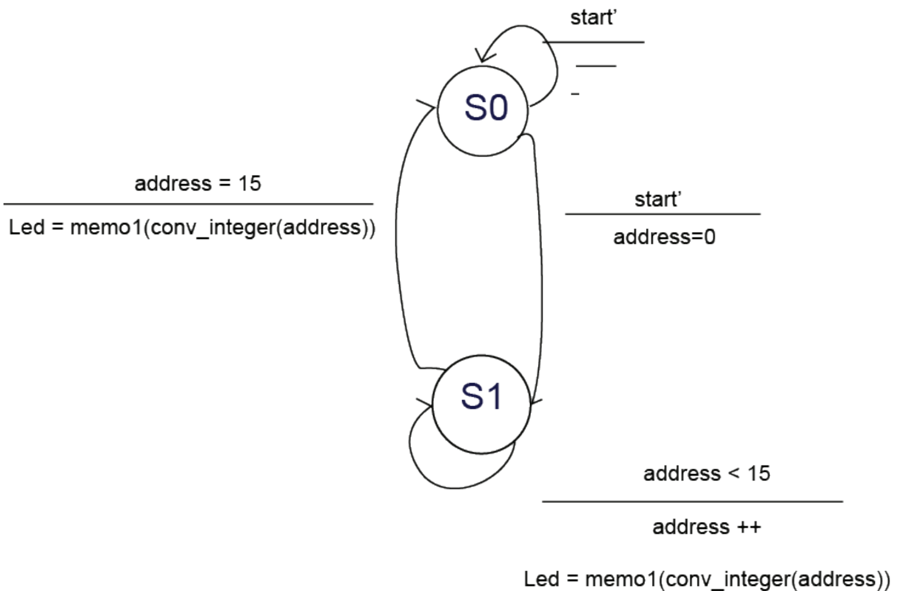


Figura 11.4 Autómata de lectura de memoria

©Editorial Digital Tecnológico de Monterrey

```
entity memoria is
    Port(start: in  STD_LOGIC;
           leds: out  STD_LOGIC_VECTOR (3 downto 0);
           clk: in  STD_LOGIC);
end memoria;
```

```
architecture Behavioral of memoria is
type rom_type is array (0 to 15) of std_logic_vector (3 downto 0);

constant Memo1:rom_type:=
("1001", "0001", "0111", "1001", "0001", "0111", "1001",
"0001", "0111", "1001", "0001", "0111", "1001", "0001",
"0111", "0000");
signal state: integer: = 0;
signal address: std_logic_vector (3 downto 0);
begin
process(clk, start)
  begin
  if clk = '1' and clk'event then
    case state is
    when 0 => if start='1' then state <= 1; else null; end if;
                address <= "0000";
    when 1 => if address < "1111" then
                                       leds <= Memo1(conv_integer(address));
                       state <= 1;
                       address <= address + 1;
                       else
                       leds <= Memo1(conv_integer(address));
                       state <= 0;
                       end if;
      when others => null;
      end case;
    else null;
     end if;
end process;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

El reporte de la síntesis de este circuito es el siguiente:

```
Found 4-bit register for signal <leds>.
    Found 4-bit comparator less for signal <$n0002> created at line 52.
    Found 2-bit 4-to-1 multiplexer for signal <$n0006> created at line 49.
    Found 4-bit adder for signal <$n0007> created at line 55.
    Found 4-bit register for signal <address>.
    Found 2-bit register for signal <state>.
    Summary:
        inferred   1 ROM(s).
        inferred  10 D-type flip-flop(s).
        inferred   1 Adder/Subtractor(s).
        inferred   1 Comparator(s).
        inferred   2 Multiplexer(s).
Unit <memoria> synthesized.
```

11.3 Ejercicio

Elimine el registro LEDs del esquemático y del código.

©Editorial Digital Tecnológico de Monterrey

11.4 Memorias RAM

En esta sección se mostrará el uso de memoria **RAM** en un circuito, para esto se resolverá el siguiente problema:

- Colocar un número de 4 *bits* en 4 *switches* y oprimir un botón de <Enter>.
- Repetir esta operación hasta que se oprima un botón de <Fin>.
- En ese momento se desplegarán los números en 4 LEDs en el mismo orden en que fueron capturados.

Para resolverlo, los números que se capturen se almacenarán en una RAM y enseguida se obtendrán de la memoria y se desplegarán en LEDs.

El diseño de la unidad de control es el siguiente:

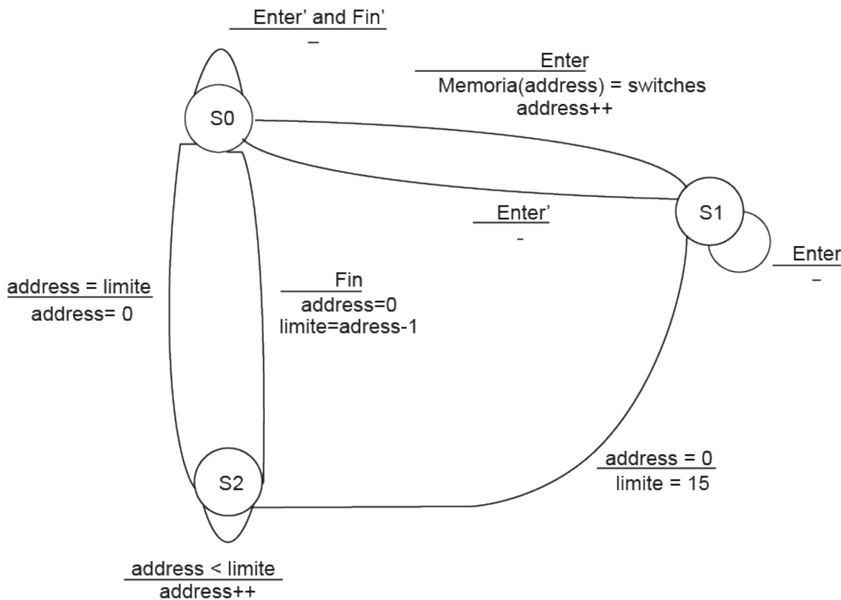


Figura 11.5 Autómata de escritura de memoria

©Editorial Digital Tecnológico de Monterrey

Con los estados S0 y S1 se realiza la captura y con S2 se despliega. La captura espera a que se suelte el botón <Enter> para almacenar el siguiente número. Se usará una memoria de 16 posiciones, así que si se llega a sobrepasar la dirección de 15 (que si se maneja en 4 *bits* se convertiría en un valor de 0), entonces termina la captura y se fija el límite en 15.

El diseño esquemático del circuito, que corresponde al diseño de la unidad de control y los registros es el siguiente:

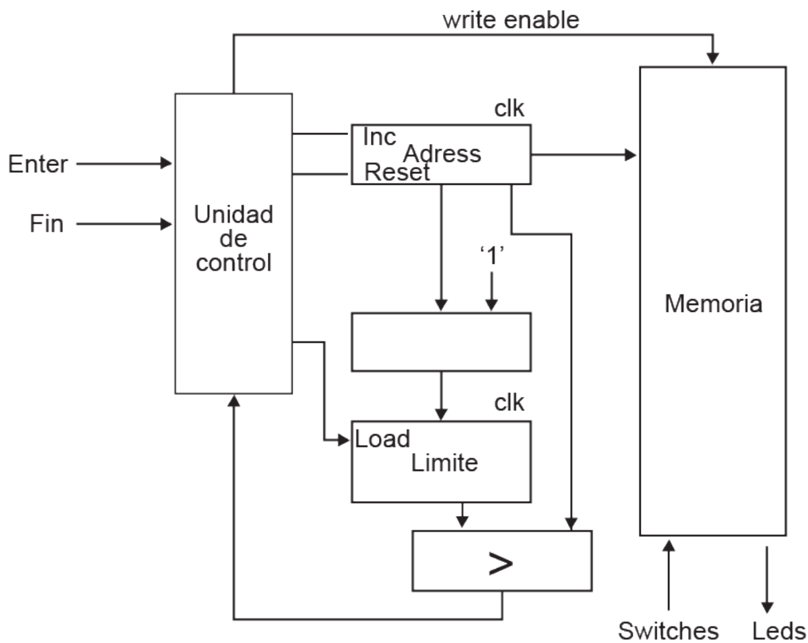


Figura 11.6 Diseño esquemático de escritura de memoria

©Editorial Digital Tecnológico de Monterrey

Finalmente, el código del circuito es:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity memoria_captura is
     port(clk: in STD_LOGIC;
          switches: in STD_LOGIC_VECTOR(3 downto 0);
          Leds: out STD_LOGIC_VECTOR(3 downto 0);
          enter: in STD_LOGIC;
          fin: in STD_LOGIC);
end memoria_captura;

architecture Behavioral of memoria_captura is
signal state: std_logic_vector(1 downto 0): = "00";
type ram_type is array(0 to 15) of std_logic_vector(3 downto 0);
signal memoria: ram_type;
signal address, limite: std_logic_vector(3 downto 0);
begin


process(clk)
begin
     if clk = '1' and clk'event then
          case state is
          when "00" => if (enter = '0') and (fin='0') then state <= "00";
                    elsif enter = '1' then
memoria(conv_integer(address)) <= switches;
                         address <=a ddress + 1;
                         state <= "01";
```

©Editorial Digital Tecnológico de Monterrey

```
                    elsif (fin = '1') then address <= "0000";
                              state <= "10";
                              limite <= address-1;
                               else null;
                    end if;
          when "01" => if address = 0 then
                               state <= "10";
                               limite <= "1111";
                    elsif enter = '0' then
                               state <= "00";
                               else null;
                               end if;
          when "10" => if (address < limite) then    address <= address + 1;
                                   state <= "10";
                                 else    address <= "0000";
                                   state <= "00";
                          end if;
          when others => null;
          end case;
     Leds <= memoria(conv_integer(address));
     end if;
end process;
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

11.5 Más sobre memorias

Para utilizar una memoria en un circuito existen varias posibilidades, a continuación, se presentan algunas de ellas:

a) Utilizar una plantilla y hacer las variaciones adecuadas.

b) Crear una instancia de un componente que ya esté definido en una biblioteca.

Figura 11.7

En el sistema ISE, en la opción *Edit/Language templates*, se encuentran diferentes plantillas para memorias RAM y ROM. Por ejemplo, la siguiente es para definir una RAM con salida asincrónica.

Es necesario contar con un *enable* para habilitar la escritura y debe ser sincrónica. La lectura no requiere reloj.

```
process (<clock>)
begin
   if (<clock>'event and <clock> = '1') then
      if (<write_enable> = '1') then
         <ram_name>(conv_integer(<write_address>)) <= <input_data>;
      end if;
   end if;
end process;

<ram_output> <= <ram_name>(conv_integer(<read_address>));
```

Crear una instancia de un componente que ya esté definido en una biblioteca.

©Editorial Digital Tecnológico de Monterrey

Por ejemplo, la siguiente es una ROM de 128 x 1 que, por ser tan pequeña, el sistema de desarrollo la coloca en un LUT.

```
Library UNISIM;
use UNISIM.vcomponents.all;

-- <----Cut code below this line and paste into the architecture body---->

   -- ROM128X1: 128 x 1 Asynchronous Distributed => LUT ROM
   --           Virtex-II/II-Pro, Spartan-3/3E
   -- Xilinx HDL Language Template version 8.1i

   ROM128X1_inst: ROM128X1
   generic map (
      INIT => X"00000000000000000000000000000000")
   port map (
      O => O,    -- ROM output
      A0 => A0, -- ROM address[0]
      A1 => A1, -- ROM address[1]
      A2 => A2, -- ROM address[2]
      A3 => A3, -- ROM address[3]
      A4 => A4, -- ROM address[4]
      A5 => A5, -- ROM address[5]
      A6 => A6  -- ROM address[6]
   );

End of ROM128X1_inst instantiation
```

©Editorial Digital Tecnológico de Monterrey

11.6 Definición de una memoria

Aunque una memoria se accesa a través de una sola dirección, en VHDL es posible crear espacios de memoria de múltiples dimensiones, como es el caso de una matriz que se maneja a través de dos direcciones. El sistema de desarrollo se encarga de realizar "desdoblar" la matriz y convertirla en vector a través de la conversión matemática de los dos índices en uno.

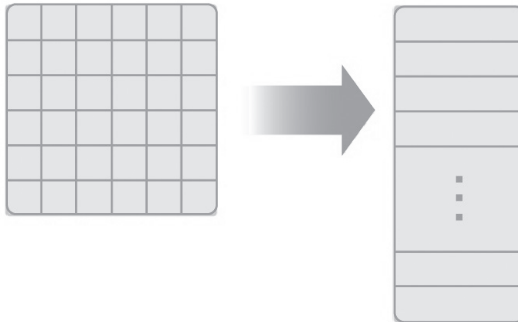


Figura 11.8 Abstracción de una memoria

Para múltiples aplicaciones es muy útil definir matrices o cubos. A continuación, se muestra un ejemplo de la definición de una matriz que se manejará como una memoria ROM.

©Editorial Digital Tecnológico de Monterrey

```
Ejemplo de una matriz de dos dimensiones

type matrix4x3 is array (1 to 4, 1 to 3)
                of integer;
     constant matrixA: matrix4x3: =
((1, 2, 3), (4, 5, 6), (7, 8, 9), (10,11,12));

      La matrixA tendría almacenados
         los siguientes datos:
                 1 2 3
                 4 5 6
                 7 8 9
               10 11 12

    Esta estructura puede ser muy útil,
     por ejemplo para el manejo del
     VGA, en caso de querer producir
        gráficas o animaciones.

   El acceso a matrixA sería como sigue:
        Salida <= matrixA (y, x)
```

©Editorial Digital Tecnológico de Monterrey

11.7 Memoria en el FPGA

El FPGA cuenta con 2 tipos de memorias que se pueden usar dependiendo de lo que le convenga más al usuario para su aplicación, ya sea memoria de bloque o memoria distribuida.

La memoria distribuida es rápida e ideal para el almacenamiento de pocos datos, ya que está compuesta por los LUTs de los bloques de celdas lógicas. El LUT puede ser configurado como una RAM sincrónica de 16 *bits* y la unión de diferentes LUTs en cascada puede formar una memoria más extensa. Cabe destacar que en ella se escribe de manera sincrónica y se lee de manera asincrónica y de ser necesario, se puede leer sincrónicamente usando los registros asociados con cada LUT. Por otro lado, cuenta con 30K *bits* de memoria, un número relativamente pequeño comparado con la memoria de bloque RAM. Además, debido a que la memoria distribuida ocupa celdas lógicas, también se reduce el número de espacio para operaciones lógicas y es por eso que se usa en aplicaciones que requieren un número pequeño de almacenamiento de datos.

La memoria de bloque RAM es más extensa que la memoria distribuida y se encuentra separada de las celdas lógicas. El Spartan3 XC3S200 cuenta con 12 bloques, donde cada bloque RAM contiene 18,432 *bits* de SRAM (Static Random Memory Access), de los cuales 16K *bits* están diseñados para el almacenamiento de datos y los 2K *bits* restantes son para paridad.

Esta memoria se puede utilizar en diferentes configuraciones, de puerto sencillo o doble, definiendo el tamaño de su bus de datos y direcciones como más convenga.

En VHDL se puede definir una memoria ROM de la siguiente manera:

```
Type tipo_rom is (3 downto 0) of std_logic_vector(3 downto 0);
constant Memo:tipo_rom:= (“1000”,”0100”,”0010”,”0001”);
```

©Editorial Digital Tecnológico de Monterrey

En VHDL se puede definir una memoria RAM de la siguiente manera:

```
Type tipo_ram is (3 downto 0) of std_logic_vector(3 downto 0);
signal Memo: tipo_ram: = (others => (others => '0'));
                                    -- la memoria queda inicializada con
                                    --ceros y es posible modificar su valor
```

Esta memoria RAM tiene 4 posiciones para guardar 4 *bits* en cada dirección.

En el siguiente ejemplo se manipula una memoria RAM que contiene 16 posiciones de 4 *bits*:

```
we- write enable
re- read enable
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ejemplo is
      port(we,re,clk: in std_logic;
              dato,direccion: in std_logic_vector(3 downto 0);
              Leds: out std_logic_vector(3 downto 0));
end ejemplo;

architecture Behavioral of ejemplo is
type tipo_ram is array(15 downto 0) of std_logic_vector(3 downto 0);
signal Memo: tipo_ram: = (others => (others=>'0'));
begin
process(we,clk,dato)
begin
      if clk = '1' and clk'event then
            if we = '1' then
                  Memo(conv_integer(direccion)) <= dato;
            else
                  null;
            end if;
      end if;
end process;

Leds <= Memo(conv_integer(direccion)) when re = '1' else (others => 'Z');

end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

En este ejemplo es importante destacar que al momento de leer o escribir sobre la memoria es necesario convertir a entero la dirección en donde se desea almacenar. Por ejemplo:

```
Memo(conv_integer(direccion)) <= dato;
```

©Editorial Digital Tecnológico de Monterrey

11.8 SRAM

A diferencia de la memoria distribuida, la cual ocupa espacio para almacenar datos y que bien se podría usar para operaciones lógicas, la memoria de bloque contiene espacio especialmente designado para almacenamiento único y total de datos, siendo así memoria de bloque. Los dos bloques de memoria **SRAM (Static Random Access Memory)** justamente entran en la categoría de memorias de bloque. Esta cuenta con una capacidad de memoria de 1 MB, dividido físicamente en dos bloques de 256K-por-16, más cabe destacar que también se pueden tratar ambos bloques como un bloque de 256K-por-32. Cada uno de los bloques cuenta con su propio chip select y sus propios habilitadores de *byte* superior (ub) e inferior (lb). Sin embargo, tanto el bus de direcciones y de datos, como el habilitador de escritura y lectura son compartidos para ambos bloques tal y como se muestra en la siguiente figura.

Cabe mencionar que, para habilitar ce, we, oe, ub o lb su salida debe ser ‘0’.

©Editorial Digital Tecnológico de Monterrey

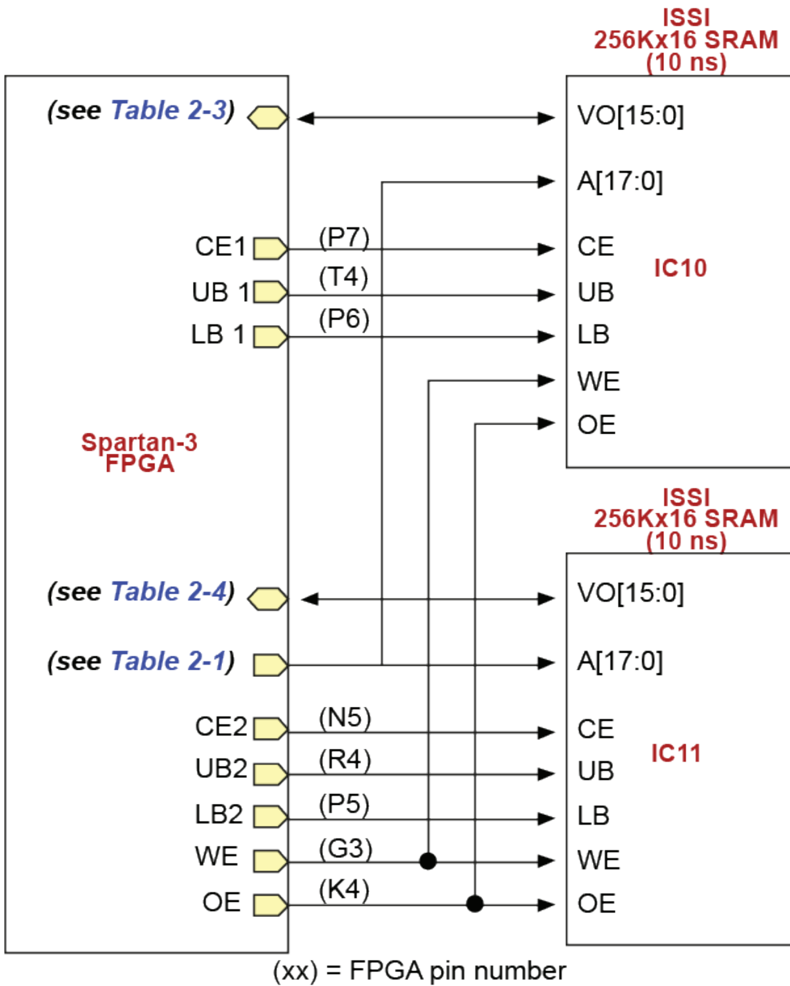


Figura 11.9 Conexiones de FPGA a SRAM (obtenido del Datasheet del Spartan 3)

La SRAM se basa en los habilitadores de escritura (we) y lectura (oe) para saber como usara el bus de datos. A continuación, se describen las diferentes combinaciones posibles:

- Para escribir una palabra de 16 *bits* del FPGA a la SRAM, el habilitador de escritura debe estar en '0' sin importar el valor del habilitador de lectura.

©Editorial Digital Tecnológico de Monterrey

- Para leer una palabra de 16 *bits* de una dirección de la SRAM, el habilitador de escritura debe estar en '1' y el habilitador de lectura en '0'.

- Para poner en alta impedancia el bus de datos ambos habilitadores deberán estar en '1'.

Código ejemplo. El siguiente ejemplo consiste en introducir y extraer datos de un bloque de la memoria SRAM, por lo tanto, solo usamos un chip select, un habilitador de byte superior y un habilitador de *byte* inferior.

La manera en cómo funcionan las entradas y salidas es la siguiente.

SW	Botón (3)
Consiste en 8 interruptores con los cuales se obtendrá información sobre el valor de la palabra que se quiere escribir o la dirección en donde dicha palabra se quiere escribir. Además, si swl(7) es '1' se deslplegaran en los LEDs los últimos 8 bits, en caso de ser '0', se desplegaran los primeros 8 bits	Se encarga de almacenar el valor de los los sw en un registro para después almacenar ese valor en los últimos 8 bits de la palabra.
Botón (2)	**Botón (1)**
Almacena el valor de los sw en un registro para después almacenar ese valor en los primeros 8 bits de la palabra.	Toma el valor de sw como la dirección donde almacenara la palabra de 16 bits ya generada anteriormente por botón(3) y botón(2).
LED	**Dir**
Ilumina las palabras almacenadas en la memoria **SRAM.**	Dirección de la **SRAM.**

Figura 11.10

©Editorial Digital Tecnológico de Monterrey

Además, es importante mencionar que se hacen dos estados para lograr un óptimo desarrollo y funcionamiento de la SRAM, para así lograr escribir y leer correctamente.

Asimismo, se hace uso del oneshot para eliminar todo tipo de rebotes.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SRAM is
     port(clk: in std_logic;
          sw: in std_logic_vector(7 downto 0);
          boton: in std_logic_vector(3 downto 0);
          Led: out std_logic_vector(7 downto 0);
          dir: out std_logic_vector(17 downto 0);
          we, oe: out std_logic;
          datosram: inout std_logic_vector(15 downto 0);
          ce, ub, lb: out std_logic);
end SRAM;

architecture Behavioral of SRAM is
   type statetype is (espera, escritura1, escritura2, lectura1, lectura2);
   signal state: statetype;
     signal datoescritura: std_logic_vector(15 downto 0);
     signal datolectura: std_logic_vector(15 downto 0);
     signal osboton: std_logic_vector(3 downto 0);

begin

     oneshot0: entity work.oneshot
          port map(
               clk => clk, e => boton(0),
               qout => osboton(0));
```

©Editorial Digital Tecnológico de Monterrey

```
     oneshot1: entity work.oneshot
          port map(
               clk => clk, e => boton(1),
               qout => osboton(1));

     oneshot2: entity work.oneshot
          port map(
               clk => clk, e => boton(2),
               qout => osboton(2));

     oneshot3: entity work.oneshot
          port map(
               clk => clk, e => boton(3),
               qout => osboton(3));
process(clk,state)
begin
     if rising_edge(clk) then
          case state is
               when espera =>
                    we <= '1';
                    oe <= '1';
                    datosram <= (others => 'Z');
                         if osboton(3) = '1' then
                              datoescritura<= sw & datoescritura(7 downto 0);
                         elsif osboton(2) = '1' then
```

©Editorial Digital Tecnológico de Monterrey

```
datoescritura <= datoescritura(15 downto 8) & sw;
                              elsif osboton(1) = '1' then
                                    state <= escritura1;
                              elsif osboton(0) = '1' then
                                    state <= lectura1;
                              end if;
                  when escritura1 =>
                                    we <= '0';
                                    datosram <= datoescritura;
                                    state <= escritura2;
                  when escritura2 =>
                                    we <= '1';
                                    state <= espera;
                  when lectura1 =>
                                    oe <= '0';
                                    state <= lectura2;
                  when lectura2 =>
                                    datolectura <= datosram;
                                    state< = espera;
                  when others =>
                  end case;
            end if;
      end process;

ce <= '0';
ub <= '0';
lb <= '0';
dir<= "0000000000" & sw;
Led <= datolectura(15 downto 8) when sw(7) = '1' else datolectura(7 downto 0);

end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 11

1. Diseñar un circuito que contenga un contador, empleado como dirección de memoria, que sea incrementado o decrementado mediante el uso de dos botones y que el valor de este se muestre en los *display*s de siete segmentos. Los otros dos botones serán utilizados como habilitadores de escritura y lectura. Al presionar el botón para habilitar la escritura se guardará el valor de los interruptores en la dirección de memoria SRAM señalada por los *display*s de siete segmentos. Por otro lado, al presionar el botón para habilitar la lectura se obtendrá la palabra de la dirección de la memoria SRAM señalada en los *display*s de siete segmentos y posteriormente se mostrarán los primeros ocho *bits* en los Leds.

2. Se desea desplegar un mensaje iluminando solamente una columna de 5 leds que será barrida por todas las columnas correspondientes a los caracteres que se vayan a desplegar.

Por ejemplo, la **palabra** LO se desplegaría, en el

tiempo=0 se despliega en los leds la columna 0

tiempo=1 se despliega en los leds la columna 1

tiempo=2 se despliega en los leds la columna 2

etc.

©Editorial Digital Tecnológico de Monterrey

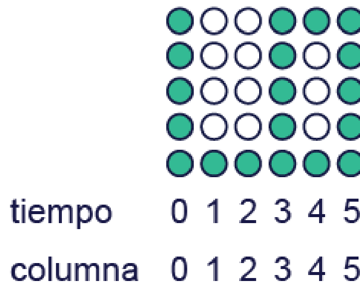


Figura 11.11

El mensaje a desplegar es de 10 caracteres. Se desplegaría del carácter 1 al 10 y luego de nuevo el 1, 2, etc.

Los caracteres están codificados en 8 *bits* (código similar al ASCII).

La configuración de un carácter en LEDs se obtendrá de un componente (ya hecho, solo lo tiene que referenciar) que se describe a continuación:

Configuración de un carácter en la columna de LEDs:

Se utilizarán 15 leds para desplegar un carácter (3*5), tres columnas de 5.

Suponga que para diseñar el circuito iluminador de LEDs cuenta con la definición de un componente que puede utilizar y que se incluirá como un componente:

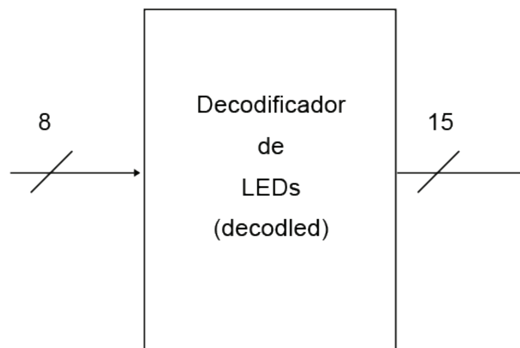


Figura 11.12

©Editorial Digital Tecnológico de Monterrey

Este decodificador recibe el código de 8 *bits* de los caracteres y proporciona la configuración, con 1s y 0s, de los 15 *bits* que se utilizarán para desplegar un carácter columna por columna. Los primeros 5 *bits* corresponden a la primera columna y los últimos 5 a la tercera.

La definición de este componente para incluirlo y referenciarlo es:

```
Component decodled is
Port ( caracter: in std_logic_vector(7 downto 0);
config: out std_logic_vector (1 to 15));
end component;
```

Suponga que el mensaje de caracteres se encuentra en la siguiente memoria interna al desplegador:

```
type memoria is array (integer range <>, integer range <>) of std_logic;

signal mensaje: memoria (1 to 10, 7 downto 0);
```

Para este problema:

a) Diseñe la máquina de estados de la unidad de control

b) Defina en VHDL la sección del circuito que despliega el mensaje en los 5 leds, la frecuencia en que se desplegarán columnas en los LEDs está dada por el reloj externo.

```
entity desplegador is
Port ( clk : in std_logic;
        leds : out std_logic_vector(1 to 5));
end desplegador;
```

©Editorial Digital Tecnológico de Monterrey

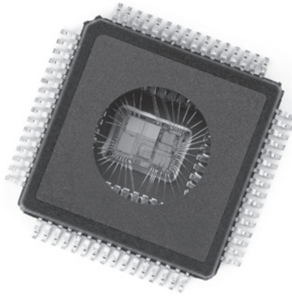
```
architecture Behavioral of desplegador is
type memoria is array (integer range <>, integer range <>) of std_logic;
signal mensaje: memoria (1 to 10, 7 downto 0);
Component decodled is
Port ( caracter: in std_logic_vector(7 downto 0);
config: out std_logic_vector (1 to 15));
end component;
begin

end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 11

El uso de memorias permite un diseño muy eficiente en aplicaciones que deben procesar datos. Los FPGAs cada vez incorporan en sus diseños internos mayor cantidad de espacios de memoria facilitando aplicaciones que manejan periféricos o que tienen datos fijos.

Este capítulo facilita el diseño de las aplicaciones que se proponen en el siguiente dedicado a los periféricos.

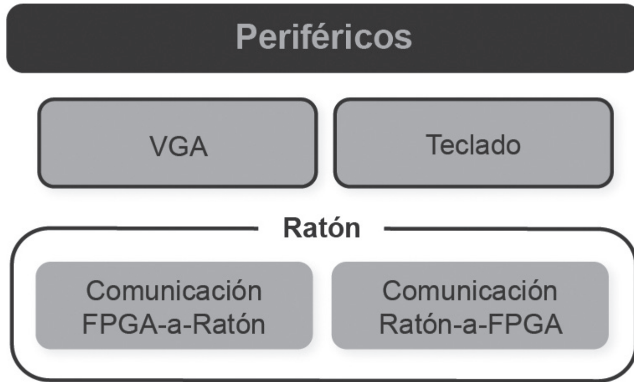


©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 12. Periféricos



©Editorial Digital Tecnológico de Monterrey

12.1 VGA

La interfaz VGA (Video Graphics Array) esta basada en cinco salidas: Red (R), Green (G), Blue (B), Horizontal Syncronization (HS) y Vertical Sync (VS), las cuales se explican en este capítulo. Un dato importante a mencionar es que se debe trabajar con un reloj de 25 MHz (Recordando que el Spartan-3 cuenta con un reloj de 50 MHz).

El VGA tiene tres salidas de color para cada **pixel** contenido en su resolución de 640 x 480 pixeles. A cada pixel se le aplicará la combinación de estos tres colores (**RGB**) y dependiendo de la intensidad de cada color, se desplegará un color final en ese pixel. Por ejemplo, si el color azul está al máximo mientras los otros dos están apagados (RGB<= "001"), el color desplegado en la pantalla será un azul fuerte.

El haz del monitor se encarga de colorear dichos pixeles recorriendo la pantalla horizontalmente coloreando cada línea por cada pulso de reloj. Cuando una línea ha sido complemente coloreada (hsync = 800), una señal (hsync) le indica que debe pasar al inicio de la siguiente línea horizontal, aumentando así con un 1 la segunda señal (vsync++) y regresando la primera señal a su valor inicial (hsync = 0). Así mismo, cuando se ha terminado de colorear la última línea (donde vsync = 521), la segunda señal (vsync) regresa a su valor inicial (vsync = 0). Por lo tanto, con la ayuda de estas dos señales (contadores), hsync y vsync, se podrá manejar de una manera efectiva las salidas HS y VS.

En el siguiente fragmento de código se aprecia lo anteriormente explicado:

```
process(clk25)
 begin
  if(clk25='1' and clk'event) then
     if(vsync = 521) then
       vcont<=(others=>'0');
      elsif(hsync=800) then
       hsync<=(others=>'0');
       vsync<=vsync+1;
      else
       hsync<=hsync+1;
       end if;
     end if;
end process;
```

©Editorial Digital Tecnológico de Monterrey

En la siguiente tabla y diagrama de tiempo, tomados ambos de la página 26 del manual *Spartan-3 Starter Kit Board User Guide*, podemos apreciar datos de vital importancia donde explica cuando deben estar en HIGH y LOW las salidas HS y VS.

Symbol	**Parameter**	**Vertical Sync**			**Horizontal Sync**	
		Time	**Clocks**	**Lines**	**Time**	**Clocks**
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48

Tabla 12.1 Tiempos VGA

©Editorial Digital Tecnológico de Monterrey

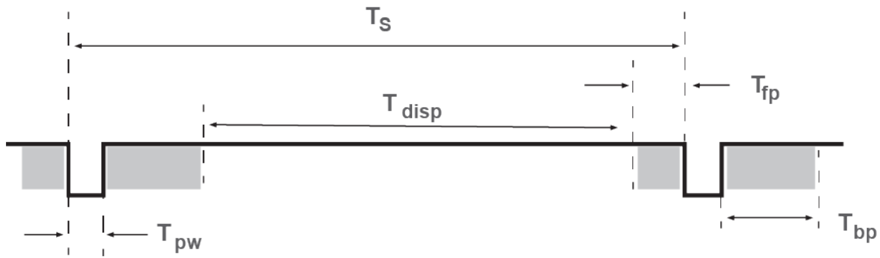


Figura 12.1 Control de tiempo en VGA

En el diagrama de tiempo, se pueden apreciar zonas grises, denominadas Back Porch (Tbp) y Front Porch (Tfp), siendo Tbp el intervalo previo y Tfp el intervalo posterior a los tiempos de sincronización del pulso. Cabe destacar que durante estos intervalos no se puede desplegar información.

Por otro lado, podemos apreciar un dato de gran relevancia, Pulse Width (Tpw), tanto en la sincronización vertical como en la horizontal, claro está, con diferentes valores.

Tomando en cuenta los datos obtenidos de Tpw, podemos expresar tanto HS en función de hsync como VS en función de vsync de una manera muy sencilla, la cual se explica a continuación.

Cada vez que nuestro contador hsync se encuentre dentro de un rango que contenga el valor de Tpw horizontal, HS deberá ser LOW, en cualquier otro caso HIGH.

Por ejemplo: (hsync>0 and hsync<97)

De manera similar, cada vez que nuestro contador vsync se encuentre dentro de un rango que contenga el valor de Tpw vertical, VS deberá ser LOW, en cualquier otro caso HIGH.

Por ejemplo: (vsync>0 and vsync<3)

Todo esto, claro, de una manera sincrónica con el reloj de 25 MHz.

©Editorial Digital Tecnológico de Monterrey

Ahora que ya sabemos la manera en como se manejan las salidas RGB, HS y VS, podemos desplegar desde colores hasta imágenes en la pantalla y para entender de mejor manera la forma de manipular la ubicación de dicho color o imagen observemos la siguiente figura:

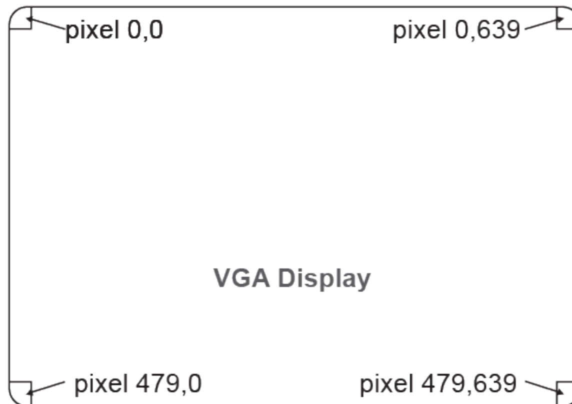


Figura 12.2 Número de pixeles en 'x' y 'y' empleados como coordenadas

Podemos observar en la figura que se puede manipular la ubicación del color o imagen en la pantalla mediante un plano coordenado (x, y). Para esto, necesitamos tener definido tanto 'x' como 'y' (siendo estas asincrónicas al reloj de 25 MHz) y contar con las señales ya mencionadas en los párrafos anteriores.

El valor que 'x' debe tomar, es la resta de hsync con Tpw y Tbp, mientras que 'y' es la resta de vsync con Tpw y Tbp, claro está, con sus respectivos valores tomados de la tabla.

Es importante recordar que el área de trabajo del **VGA** es de 640 x 480 pixeles, por lo tanto, todo lo que este fuera de ese espacio debe estar coloreado (RGB="000") de negro.

En la siguiente tabla se muestran las diferentes combinaciones RGB y los colores que resultan de cada una:

©Editorial Digital Tecnológico de Monterrey

R	G	B	Color
0	0	0	Negro
0	0	1	Azul
0	1	0	Verde
0	1	1	Celeste
1	0	0	Rojo
1	0	1	Magenta
1	1	0	Amarillo
1	1	1	Blanco

Tabla 12.2 Combinaciones RGB

©Editorial Digital Tecnológico de Monterrey

12.2 Teclado

La tarjeta en la que está montado el Spartan-3 cuenta con un puerto PS/2 para periféricos al cual puede conectarse tanto un teclado como un **ratón**. Este tipo de puerto tiene un estándar de comunicación y conexión. Un pin corresponde a la recepción/transmisión de un dato (transmisión por parte del periférico, recepción para el fpga). Otro pin debe recibir una señal de +5 V que el fpga transmitirá y que el teclado usará como fuente de energía. En esta sección se profundiza en el circuito manejador del teclado.

El teclado PS/2 cuenta con un circuito que genera un código en forma serial por cada tecla oprimida, lo que se le denomina un ***scan code***. Un *scan code* es el código de 8 *bits* asociado a cada letra, es decir, si el usuario presiona la letra L, el *scan code* que se genera es 4B (hexadecimal). El teclado emite una señal de reloj, lo cual facilita la lectura del *scan code*, ya que la sincronía del ***scan code*** es la de esta señal de reloj. Los circuitos que se comunican con el teclado pueden utilizar esta señal de reloj para detectar la secuencia que se genera por el teclado.

Para diseñar un circuito controlador que procese lo que el teclado genera y se obtengan uno por uno los *bits* de un *scan code* correspondiente a una tecla presionada, es necesario hacer las siguientes consideraciones.

Desde el momento en que el teclado se energiza, su circuito comienza a transmitir una secuencia de 11 *bits* que al terminar comienza con una nueva secuencia. En la transmisión serial, un *bit* es una señal de 1 o 0 que tiene la duración de un período de reloj (de su propio reloj). La secuencia que genera el teclado consta de un *bit* de start (0 lógico) que puede desecharse, el *scan code* de 8 *bits*, un *bit* de paridad que puede desecharse y un *bit* de stop (1 lógico) que también puede desecharse.

El circuito controlador del teclado puede diseñarse con una máquina de estados sincronizada con el reloj que envía el teclado. La máquina puede pasar por 11 estados que incluyan el reconocimiento del *bit* de start, el *scan code*, el *bit* de paridad y el *bit* de stop

©Editorial Digital Tecnológico de Monterrey

donde del segundo al noveno estado debe almacenarse el *bit* leído en un registro serial para formar el *scan code*, o bien mediante un contador almacenar cada uno de los *bits* necesarios.

Una manera efectiva para reconocer la tecla que fue presionada es checar el valor que tiene el registro donde se cargó serialmente el *scan code*, y dependiendo del valor almacenado es la tecla oprimida. Cabe destacar que, al dejar de presionar una tecla, por ejemplo <Enter>, la secuencia hexadecimal es la siguiente: 5A F0 5A, donde a F0 se le denomina break code.

Recomendaciones: El VHDL puede manejar constantes en hexadecimal. La notación requiere iniciar la constante con x y a continuación la constante entre comillas dobles, por ejemplo "01001011" es x "4B".

©Editorial Digital Tecnológico de Monterrey

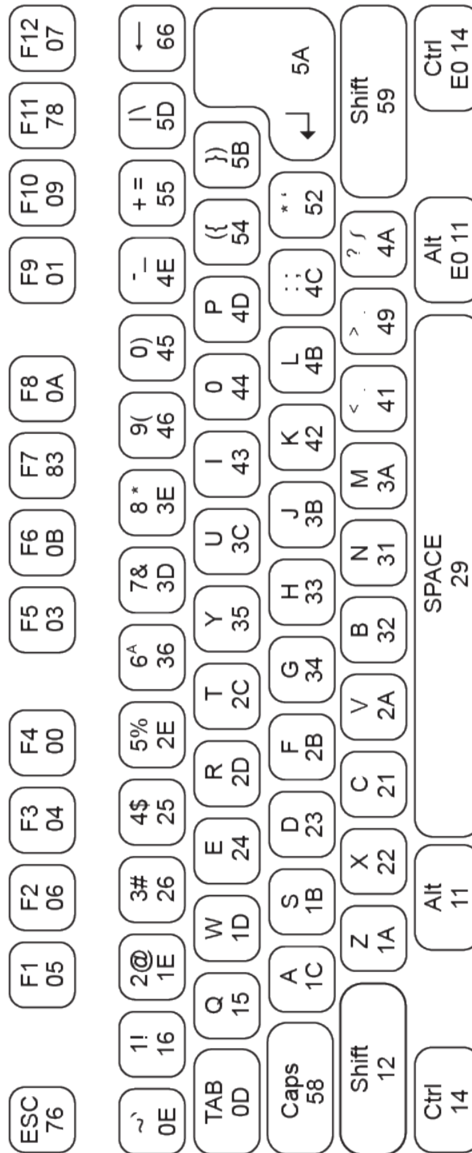


Figura 12.3 Scan codes hexadecimales del teclado

©Editorial Digital Tecnológico de Monterrey

En el circuito de la **Figura 12.6** se muestra la manera en como recibir el código hexadecimal proveniente del teclado.

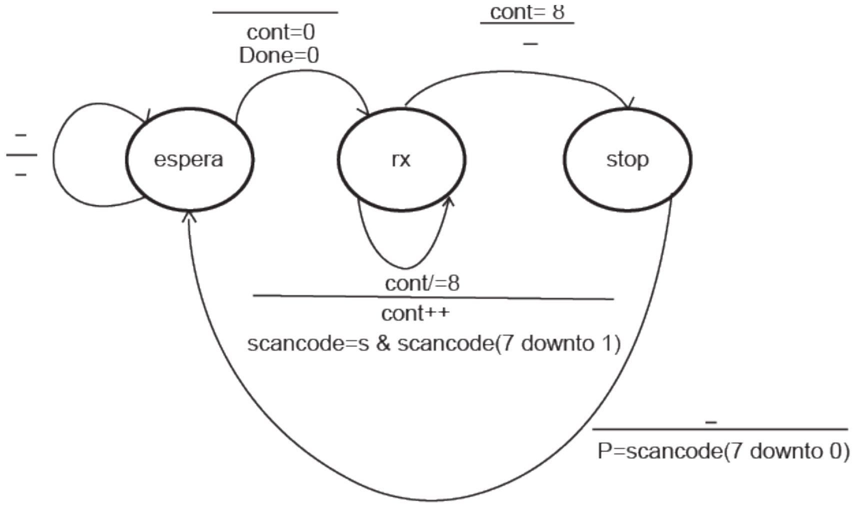


Figura 12.4 Autómata de manejor de teclado

©Editorial Digital Tecnológico de Monterrey

Código:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Teclado is
  port( ps2dato: in std_logic;
              ps2clk: in std_logic;
              code: out std_logic_vector(7 downto 0));
end Teclado;


architecture Behavioral of Teclado is
type state_type is (espera, rx, stop);
signal state: state_type;
signal scancode: std_logic_vector(7 downto 0);
signal cont: std_logic_vector(3 downto 0);
begin
  process(ps2clk)
  begin
       if (ps2clk'event and ps2clk= '1') then
              case state is
                    when espera =>
                         state<= rx; cont<="0000";

                    when rx =>
                         if cont=8 then
                              state<=stop;
                         else
                              state<=rx;
                              cont<=cont+1;
```

©Editorial Digital Tecnológico de Monterrey

```
                              scancode<=ps2dato & scancode(7 downto 1);
                             end if;
                       when stop =>
                             state<=espera;
                             code<=scancode(7 downto 0);
               end case;
        end if;
   end process;
end Behavioral;

Pines en el Spartan-3 del teclado y ratón.
NET "ps2clock"  LOC = "M16"  ;
NET "ps2dato"  LOC = "M15"  ;
```

©Editorial Digital Tecnológico de Monterrey

12.3 Ratón

El ratón se comunica con el FPGA a través de comunicación serial bidireccional, es decir, tanto el FPGA como el ratón transmiten datos y se inicia con el FPGA mandándole un paquete con un byte de comando para que el ratón se active, de lo contrario el ratón no mandará información después de ser encendido. En este caso el FPGA debe mandar el comando F4 hexadecimal para inicializar la actividad en el ratón habilitando en él el modo stream. De esta manera, este comenzará a mandar 3 paquetes con la información sobre los botones presionados y su movimiento al FPGA, por lo tanto, se debe elaborar un sistema bidireccional que transmita y reciba información con el uso de operaciones de triple estado.

El ratón envía la información al FPGA de manera serial a través de la línea de datos y en sincronía con su reloj, el cual debe ser filtrado para eliminar el ruido y lograr que el FPGA reciba correctamente los datos. Cada paquete que el ratón envía contiene 11 *bits*: 1 *bit* de start, 8 *bits* de datos, 1 *bit* de paridad y 1 *bit* de *stop*. Las acciones de los botones y algunas banderas se encuentran en el primer *byte* de datos, el movimiento en el eje x en el segundo *byte* y el movimiento en el eje y en el tercer *byte*.

Bit	Byte 1	Byte 2	Byte 3
7	Overflow en Y	X7	Y7
6	Overflow en X	X6	Y6
5	Bit de signo en Y	X5	Y5
4	Bit de signo en X	X4	Y4
3	1	X3	Y3
2	Botón central	X2	Y2
1	Botón derecho	X1	Y1
0	Botón izquierdo	X0	Y0

Tabla 12.3 Formato de los 3 bytes enviados por el ratón

©Editorial Digital Tecnológico de Monterrey

Cada *byte* de movimiento se define como 9 *bits* en complementos a 2, donde el noveno *bit* de cada *byte* de movimiento se encuentra en el primer *byte*. Por ende, el rango de movimiento se encuentra entre -255 a +255, donde si el noveno *bit* es 0 significa que es positivo y si es 1 es negativo.

El ratón cuenta con diferentes modos de operación. El más común es el modo stream, en el cual el ratón envía sus datos de movimiento y la actividad en los botones. Durante todo el proceso, el FPGA puede mandar comandos al ratón para establecer su modo de operación y, posteriormente, este mandar un *acknowledgment* (FA) al FPGA.

12.3.1 Comunicación FPGA a ratón

Para comenzar a recibir la actividad del ratón es necesario que el FPGA primero establezca comunicación con él mediante unos pasos específicos, y después enviarle el comando F4 para que así el ratón entre en modo stream y comience a enviar sus 3 *bytes*.

Ahora, como el FPGA desea enviar información, primero se deben poner las líneas de reloj y datos en un estado de "Petición de envío", y para esto se debe detener toda comunicación primero, poniendo en 0 la línea de reloj por al menos 100 microsegundos. Posteriormente, se hace la petición de envío poniendo la línea de datos en 0 y la línea de reloj en alta impedancia. Después de que el ratón ha detectado el estado de "Petición de envío", este comenzará a generar una señal de reloj para recibir los 8 *bits* de datos y un *bit* de stop. Una vez que se ha recibido el *bit* de stop, el ratón dará la señal de haber recibido el *byte* poniendo en 0 la línea de datos y generando un último pulso de reloj. Para este momento el FPGA deberá haber puesto la línea de datos en alta impedancia.

©Editorial Digital Tecnológico de Monterrey

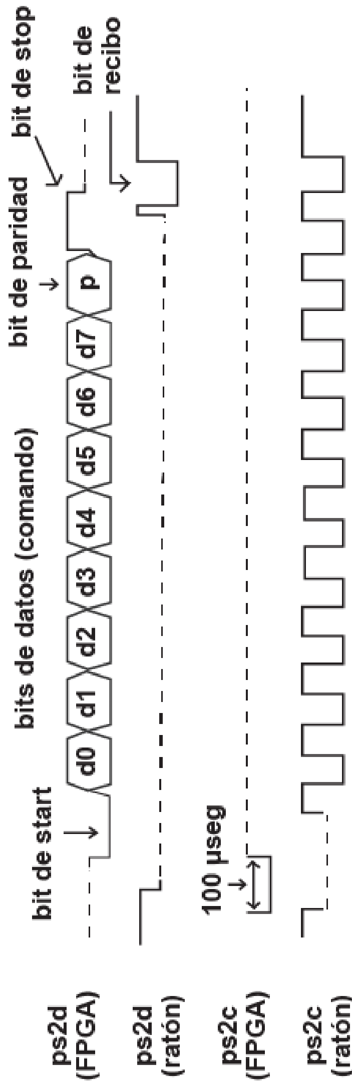


Figura 12.5 Diagrama de tiempo que muestra la inicialización del ratón

Por lo tanto, para establecer comunicación entre el FPGA y el ratón se deben seguir detalladamente los pasos presentados en el gráfico.

©Editorial Digital Tecnológico de Monterrey

Figura 12.6

12.3.2 Comunicación ratón a FPGA

Cada vez que se recibe un *byte* del ratón o del teclado se reciben 11 *bits*:

- 1 *bit* de start (siempre 0 lógico).
- 8 *bits* de datos, se recibe primero el menos significativo.
- 1 *bit* de paridad.
- 1 *bit* de stop (siempre 1 lógico).

Estos *bits* son leídos por la transición negativa de la señal de reloj del puerto ps/2 y su frecuencia tiene un rango de entre 10 kHz y 16.7 kHz. Para obtener correctamente los tres *bytes* enviados por el ratón se filtro la señal de reloj para eliminar el ruido.

©Editorial Digital Tecnológico de Monterrey

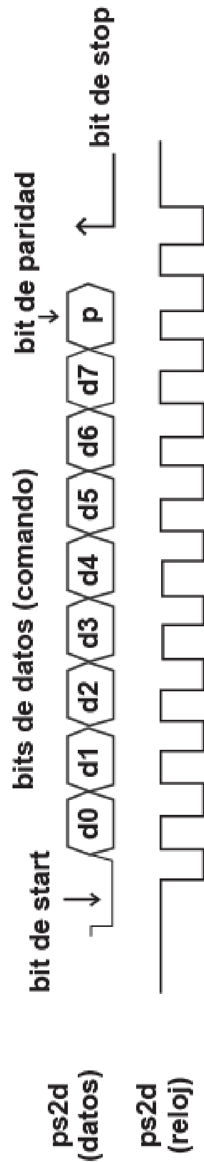


Figura 12.7 Diagrama de tiempo que muestra la recepción de datos a través del puerto ps/2

©Editorial Digital Tecnológico de Monterrey

12.3.3 Diseño de transmisión-recepción del ratón

En el siguiente código se muestra lo mencionado anteriormente, así como un correcto funcionamiento sobre la transmisión y recepción de datos a través del puerto ps/2 usando el ratón.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ps2raton_txrx is
port (
  clk: in std_logic;
  ps2d, ps2c: inout std_logic;
               xraton,yraton: out std_logic_vector(8 downto 0);
               mrl: out std_logic_vector(2 downto 0);
               recibocompletado: out std_logic

  );
end ps2raton_txrx;

architecture Behavioral of ps2raton_txtx is
  type statetype is (idletx, peticion, start, bytetx, stop,idlerx,bytesrx,done);
  signal state: statetype;
           signal bitsAenviar,bitsArecibir,contadortx,contadorbyte : integer;
           signal datotx: std_logic_vector(8 downto 0);
           signal datorx1,datorx2,datorx3: std_logic_vector(10 downto 0);
           signal byte1,byte2,byte3: std_logic_vector(7 downto 0);
  signal paridad,ps2c_filtrado,ps2c_filtrado,ps2c_t2,ps2c_t1: std_logic;

begin

          process(clk)
          begin
                  if rising_edge(clk) then
                             ps2c_t1<= ps2c;
                             ps2c_t2<=ps2c_t1;
                     end if;
          end process;

          ps2c_filtrado<= ps2c_t2 and (not ps2c_t1); --Transicion negativa del reloj filtrado.

  paridad <= not ('1' xor '1' xor '1' xor '1' xor
          '0' xor '1' xor '0' xor '0');

process(clk,state,contadortx,bitsAenviar,bitsArecibir,datorx1,datorx2,
                                 datorx3,datotx,contadorbyte,paridad,ps2c,filtrado)

  begin

                    if rising_edge(clk) then
                               case state is
                when idletx=>
```

©Editorial Digital Tecnológico de Monterrey

(Continuación)

```
                  elsif contadorbyte=2 then

                                          datorx2 <= ps2d & datorx2(10 downto 1);
                                          elsif contadorbyte=3 then
                                          datorx3<= ps2d & datorx3(10 downto 1);
                                          end if;
                                          bitsArecibir <= 9;
                                          state <= bytesrnx;
                                     else

                                          state<=idlerx;

                                     end if;

when bytesrx =>
 if ps2c_filtrado='1' then

                                    if contadorbyte=1 then
    datorx1 <= ps2d & datorx1(10 downto 1);
                                    elsif contadorbyte=2 then
    datorx2 <= ps2d & datorx2(10 downto 1);
                                    elsif contadorbyte=3 then
                                    datorx3<= ps2d & datorx3(10 downto 1);
                                    end if;
    if bitsArecibir = 0 then
       state <=done;
    else
       bitsArecibir <= bitsArecibir - 1;
                                              state<=bytesrx;
    end if;
  end if;
when done=>
  if contadorbyte=3 then
                                    contadorbyte<=1;
                                    recibocompletado<=´'1';
                            else
                                    contadorbyte<=contadorbyte+1;
                            end if;
                            state <= idlerx;

                       end case;

        end if;

  end process;
  byte1<=datorx1(8 downto 1);
  byte2<=datorx2(8 downto 1);
  byte3<=datorx3(8 downto 1);
  xraton<=byte1(4) & byte2;
  yraton<=byte1(5) & byte3;
  mrl<= byte1(2 downto 0);

end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Código ejemplo 1

En el siguiente ejemplo se usa como componente el código ps2raton_txrx para comprobar el funcionamiento del ratón y manipular los LEDs del FPGA, ya sea presionando el botón derecho, izquierdo o central del ratón o mediante el movimiento en x o en y.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity raton_ej is
 port(
  clik: in std_logic;
  ps2d, ps2c: inout std_logic;
                leds: out std_logic_vector(7 downto 0)
                );
end raton_ej;

architecture Behavioral of raton_ej is

signal xraton,yraton: std_logic_vector(8 downto 0);
signal mrl: std_logic_vector(2 downto 0);
signal recibocompletado: std_logic;
signal px,py: unsigned(9 downto 0);
signal xleds,yleds: std_logic_vector(3 downto 0);

begin

  raton: entity work.ps2raton_txrx(Behavioral)

port map(clk => clk, ps2d, ps2c => ps2c,
         xraton => xraton, yraton => yraton, mrl => mrl,
         recibocompletado => recibocompletado);

process(clk)

                       begin
                               if rising_edge(clk) then
                                         if recibocompletado='0' then
                                                  px<= px;
                                                  py<= py;
```

©Editorial Digital Tecnológico de Monterrey

```
(Continuación)
                elsif mrl="001" then
                          px<=(others=>'0');
                                                py<=(others=>'0');
                elsif mrl="010" then
                                                px<=(others=>'1');
                                                py<=(others=>'1');
                                      elsif mrl="100" then
                                                px<=(others=>'0');
                                                py<=(others=>'1');
                                      else
                                                px<= px + unsigned(xraton(8) & xraton);
                                                py<= py + unsigned(yraton(8) & yraton);
                                      end if;
                             end if;
               end process;

process(px,py)
          begin
          case px(9 downto 8) is
                   when "00"=> xleds <="1000";
                   when "01"=> xleds <="0100";
                   when "10"=> xleds <="0010";
                   when "11"=> xleds <="0001";
                   when others=>
         end case;
         case py(9 downto 8) is
                   when "00"=> yleds <="1000";
                   when "01"=> yleds <="0100";
                   when "10"=> yleds <="0010";
                   when "11"=> yleds <="0001";
                   when others=>
        end case;
end process;

leds<= xleds&yelds;

end Behavioral;
```

Código ejemplo 2

En el siguiente ejemplo se hace uso del ratón y de la interfaz VGA.

Este circuito también usa como componente el código ps2raton_txrx y consiste en controlar un cuadro con el propio ratón, ya sea presionando sus botones para cambiar su color o desplazando el ratón para de igual manera mover el cuadro.

©Editorial Digital Tecnológico de Monterrey

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity raton_vga is
        port( clk : in std_logic;
                        ps2c,ps2d : inout std_logic;
                        rgb : out std_logic_vector(2 downto 0);
                        hs_out : out std_logic;
                        vs_out : out std_logic;
                        );

end raton_vga;

architecture Behavioral of raton_vga is
signal clk25 : std_logic;
signal hc, x : integer;
signal vc, y : integer;
signal hinf : integer:= 200;
signal hsup: integer:=251;
signal vinf: integer:=200;
signal vsup: integer:=233;
signal xraton,yraton: std_logic_vector(8 downto 0);
signal mrl: std_logic_vector(2 downto 0);
signal recibocompletado: std_logic;
signal px,py: std_logic_vector(9 downto 0);
signal color: std_logic_vector(2 downto 0);


begin
  raton: entity work.ps2raton_txrx(Behavioral)
     port map(clk => clk, ps2d => ps2d, ps2c => ps2c,
        xraton => xraton, yraton => yraton, mrl => mrl,
        recibocompletado => recibocompletado);

        process(clk)
        begin
                if(clk = ‘1’ and clk’event) then
                        clk25<= not clk25;
                end if;
        end process;

        process(clk)
        being
                if rising_edge(clk) then
                        if recibocompletado=’0’ then
                                px<= px;
                                py<= py;
                        elsif mrl=”001” then
                                color<=color+1;
                        elisf mrl=”010” then
                                color<= color-1;
                        elisf px>= 593 then
                                px<=”1001010000”;
                        elisf px<=65 then
```

©Editorial Digital Tecnológico de Monterrey

```
                    px<="0001000010";
          elsif px>=440 then
                    px<="0110110111";
          elisf py<= 50 then
                    py <="0000110011";
          else
                    px<= px + (xraton(8) & xraton);
                    py<= py - (yraton(8) & yraton);
          end if;
        end if;
end process;

process (clk25)
begin
         if clk25'event and clk25 = '1' then
                    if(hc>=(px-10) and hc<=px and vc>=(py-10) and vc<=py ) then
                           rgb<="000";
                    else
                           rgb<="000";
                    end if;
         end process;

         process (clk25)
         begin
                  if clk25'event and clk25 = '1' then
                             if(hc=799)then
                                       hc<=0;
                             else
                                       hc<=hc + 1;
                                       if hc>=hinf and hc<=hsup then
                                          x<=x+1;
                                       else
                                         x<=0;
                                       end if;
                            end if;

                            if(vc=520) then
                                    vc<=0;
                            elsif(hc=799) then
                                     vc<= vc + 1;
                                     if vc>=vinf and vc<=vsup then
                                        y<=y+1;
                                       else
                                        y<=0;
                                       end if;
                           end if;
         end if;
end process;

hs_out <= '0' when hc > 659 and hc <= 755 else '1';
vs_out <= '0' when vc > = 493 and vc < 494 else '0';
end Behavioral;
```

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 12

Desarrolle y pruebe en el laboratorio los siguientes problemas.

1. Implemente un circuito que:

a) haga una prueba de color en un monitor de computadora.

b) dibuje líneas o figuras geométricas.

2. Una aplicación interesante que le hemos dado a los FPGAs es el manejo de dispositivos periféricos.

Este problema propone conectar un teclado PS/2 y un monitor para configurar un sistema que "dibuje" sobre el monitor utilizando aproximadamente 6 teclas del teclado.

La idea es la siguiente:

Se tendría un "cursor" de cualquier forma y tamaño, o incluso transparente, que puede iniciar al centro o en una esquina del área del monitor a cubrir. Este cursor hará las veces de "lápiz" para trazar líneas en el monitor.

Cuatro de las teclas se usarían como flechas con las que se trazarían líneas de la siguiente forma:

Tecla1: flecha hacia arriba, al oprimir esta tecla se traza una línea vertical hacia arriba.

Tecla 2: flecha hacia abajo, al oprimir esta tecla se traza una línea vertical hacia abajo.

Tecla 3: flecha hacia derecha, al oprimir esta tecla se traza una línea horizontal hacia la derecha.

©Editorial Digital Tecnológico de Monterrey

Tecla 4: flecha hacia izquierda, al oprimir esta tecla se traza una línea horizontal hacia la izquierda.

Oprimiendo la tecla 1 y 3 al mismo tiempo se trazaría una línea con una inclinación de 45 grados hacia arriba a la derecha. También podrían combinarse las teclas 1 y 4, 2 y 3, 2 y 4. Otra combinación sería inválida, podría o no darse prioridad a alguna tecla.

Tecla 5: hace que la punta del lápiz se vuelva transparente, teniendo un efecto de mover el cursor sin dejar rastro. Si se vuelve a oprimir esta tecla el lápiz vuelve a tomar color.

Tecla 6 o varias teclas. Utilizarlas para que el lápiz cambie de color.

Un ejemplo de trazos que podrían lograrse se muestra a continuación:

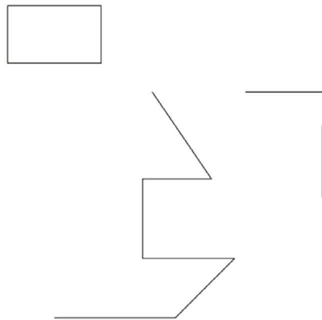


Figura 12.8

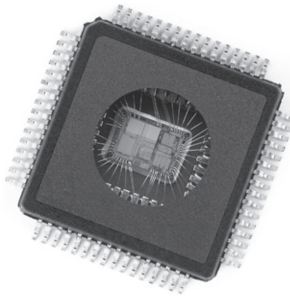
Se observan figuras separadas porque se usó lápiz "transparente" para desplazarse. Podría utilizarse toda la pantalla o solo un fragmento, por ejemplo, una cuarta parte.

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 12

El manejo de periféricos a través de un circuito dedicado logra, sin duda, mucho mejor tiempo de respuesta que una interfaz programada en un microprocesador o microcontrolador. En este capítulo no solo se explicó la operación de tres dispositivos periféricos estándares: monitor VGA, **teclado PS/2** y ratón, sino que se mostró la manera de modelar sus interfaces en VHDL para implementar sus circuitos en lógica configurable.

El estudio de este capítulo abre la posibilidad de desarrollar un sinnúmero de proyectos interesantes como lo son los juegos de video que requieren interfaces gráficas y comunicación a través de teclado y video, así como también la posibilidad de diseñar una computadora completa.

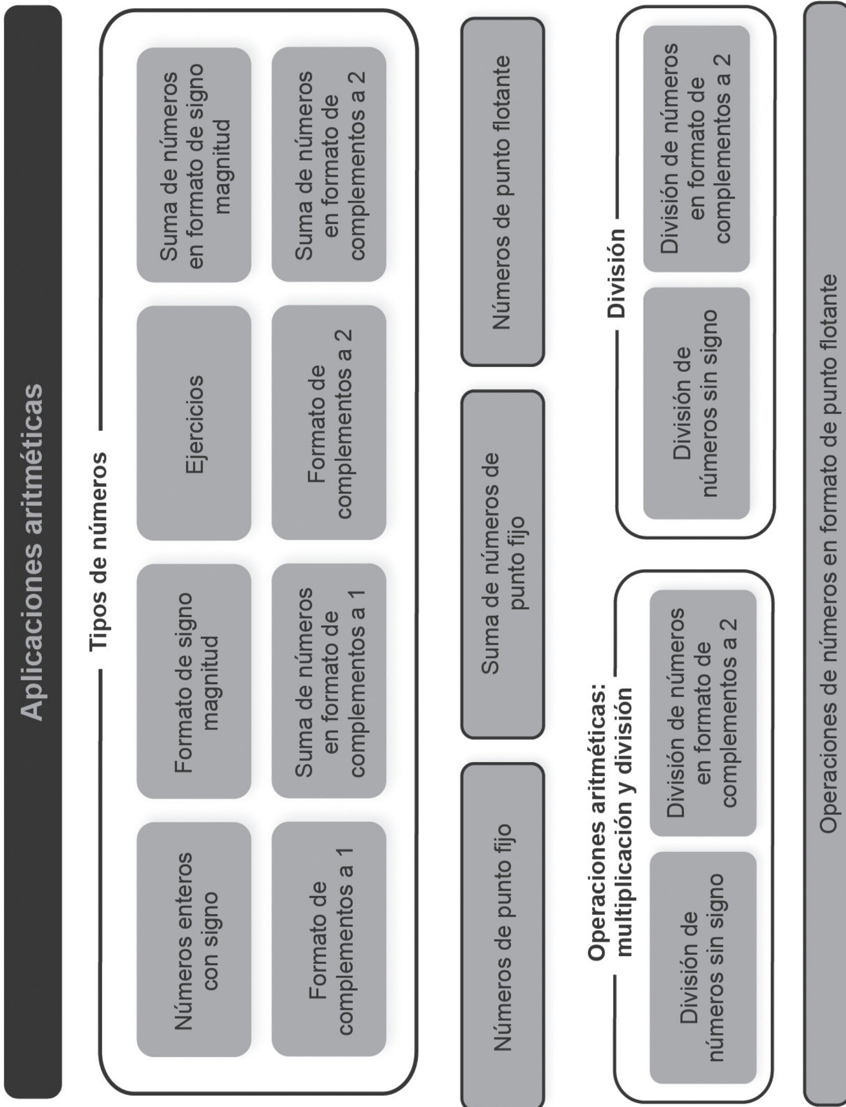


©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Capítulo 13. Aplicaciones aritméticas



©Editorial Digital Tecnológico de Monterrey

Es muy notoria la diferencia en el tiempo de respuesta de circuitos que realizan operaciones aritméticas repetidas si se implementan en una arquitectura específica, o si se implementan en un microcontrolador. Una arquitectura específica permite operaciones en paralelo y registros del tamaño adecuado. Se eliminan transferencias entre registros que sean innecesarias y por supuesto los accesos a memoria para recuperar instrucciones. El *overhead* es cero.

Las operaciones aritméticas ocurren entre números. La manera en que se representan los números puede incidir en la eficiencia de la arquitectura de un circuito que requiere calcular operaciones. En la primera sección de este capítulo se revisarán diferentes formatos para representar números, luego se revisarán las operaciones aritméticas de **multiplicación** y **división** y algunas aplicaciones.

©Editorial Digital Tecnológico de Monterrey

13.1 Tipos de números

Hay tres grandes **tipos de números**:

- Enteros
- BCD
- Números de punto fijo
- Números de punto flotante

En las siguientes secciones se revisarán los **números enteros**, de punto fijo y de punto flotante. Los BCD ya se estudiaron a profundidad en el capítulo 4.

13.1.1 Números enteros con signo

Los números enteros pueden tener signo o no. Si no tienen signo entonces se expresan solamente en binario con su magnitud con alguna longitud determinada. Si tienen signo, el número se requiere representar siguiendo las reglas de algún formato.

Existen algunos formatos, sin embargo, es posible crear reglas de representación e inventar cualquier formato. A continuación, se presentan los más comunes.

13.1.2 Formato de signo magnitud

Este es un formato muy simple, consiste en representar un número en n *bits*. El signo se representa en la posición n−1 y la magnitud en los restantes n−1 *bits*, de la posición 0 a la posición n−2.

El número positivo de mayor magnitud es +2**n−1** y el negativo de mayor magnitud es −2**n−1**.

©Editorial Digital Tecnológico de Monterrey

Figura 13.1 Formato signo magnitud

13.1.3 Ejercicios

1. Dado un formato signo y magnitud, de 8 *bits*, sume: 00011111 + 10001111

2. Dado un formato signo y magnitud, de 8 *bits*, sume: 00011111 + 10111111

3. En este mismo formato: ¿Cuál es el número mayor que se puede representar?

4. ¿Cuál es el número más negativo que se puede representar?

5. Represente el 0 en este formato.

6. Diseñe un circuito combinacional que realice la suma de dos números enteros de 8 *bits* que se encuentran en formato de signo, magnitud. Utilice, como bloques, sumadores de dos números de 8 *bits* y restadores de dos números de 8 *bits*. Estos bloques cuentan con un *carry out* (final). También, a manera de bloque, utilice *multiplexer*s de 2**n** : n. Genere la bandera de *overflow*.

©Editorial Digital Tecnológico de Monterrey

7. Compare la complejidad de este circuito con respecto a sumar números que se encuentren en **formato de complementos a 2** (si es necesario, repase en clase el formato de complementos a 2).

13.1.4 Suma de números en formato de signo magnitud

Como la suma es la operación más popular, es probable que sea la operación que más se repetiría en algún circuito promedio de los que realizan cálculos intensivos. Para evaluar qué tan bueno es este formato se observará el circuito que se requiere para sumar dos números. Este circuito ya se había presentado en el capítulo 4.

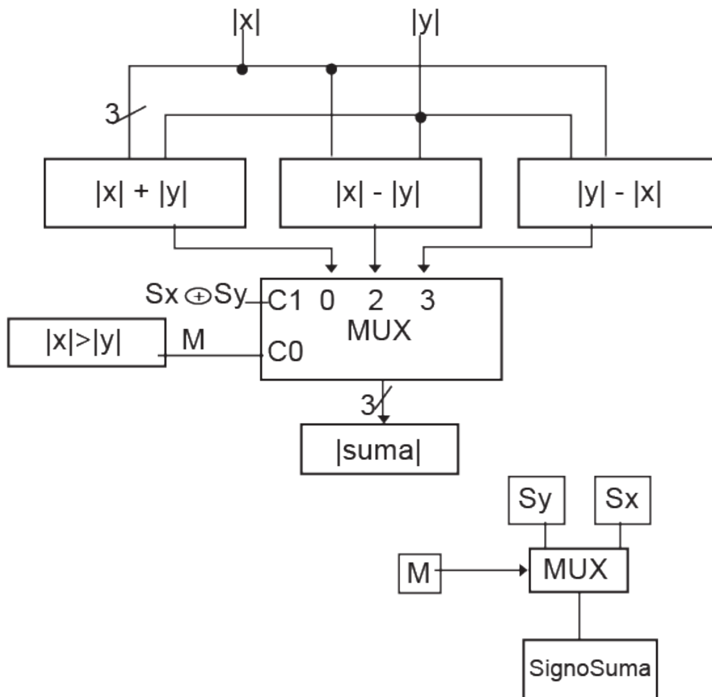


Figura 13.2 Circuito de un sumador de dos números en formato de signo magnitud

©Editorial Digital Tecnológico de Monterrey

El retraso de un circuito depende de su número de niveles, así que el retraso de este sumador corresponde al de un circuito de 4 niveles, dos niveles (circuito and-or) de la suma o resta y dos niveles (circuito and-or) del *multiplexer.* Además de tener un retraso de 4 niveles, este circuito requiere de muchos elementos.

13.1.5 Formato complementos a 1

Para una palabra de tamaño n *bits*:

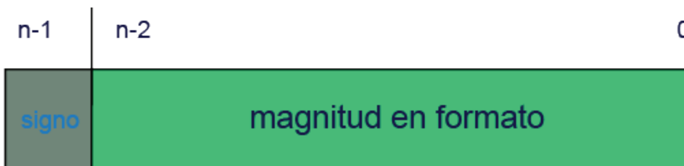


Figura 13.3 Formato de complementos a 1

¿Cómo se representa un número X en complemento a 1?

Para números positivos, el formato tiene la misma que la representación en signo magnitud, el signo es 0 y la magnitud se deja igual, así que:

Si $X>=0$: $0\&|X|$ ajustado a un total de n *bits*.

Para números negativos, la regla es:

Si $X<0$: $2n-1-|X|$

Siguiendo esta regla, el signo siempre queda como 1 y la magnitud queda "negada", es decir, los ceros cambian por unos y los unos por ceros.

©Editorial Digital Tecnológico de Monterrey

Para un formato de 8 *bits*:

¿Cuál es el número mayor positivo que se puede representar?

2**7**−1 = 127

¿Cuál es el número más negativo que se puede representar?

−(2**7**−1) = −127

¿Y para n *bits*?

El mayor positivo será 2**n−1**−1 y el más negativo será −(2n−1−1).

13.1.6 Suma de números en formato de complementos a 1

Cuando se suman dos números en formato de **complementos a 1**, alguno de los números podría ser negativo y estar representado como 2**n**−1−|X|. Si al sumar dos números, uno positivo y otro negativo, el resultado fuera positivo, habría que eliminar 2n−1 de la respuesta. Esto se logra restando (2n−1) lo que implica es ignorar el 1 que se desborda (2n) y sumar 1.

©Editorial Digital Tecnológico de Monterrey

Así que el circuito que se requiere es:

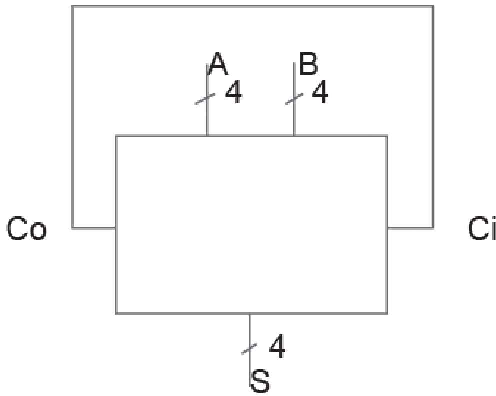


Figura 13.4 Circuito sumador de números en complementos a 1

Aunque la densidad del circuito no cambia con respecto a un sumador simple de números sin signo, el retraso del circuito es el de uno de cuatro niveles, ya que al retroalimentar la suma del *carry* out se tiene el efecto de un circuito de cuatro niveles, dos niveles de la suma (circuito and-or) y de nuevo dos niveles de la suma (circuito and-or).

Responde la siguiente actividad.

Si se suman directamente X+Y y el resultado obtenido no genera un acarreo y el *bit* más significativo es 1, ¿qué se puede esperar del resultado?

©Editorial Digital Tecnológico de Monterrey

13.1.7 Formato de complementos a 2

Si los números se representan en n *bits*:

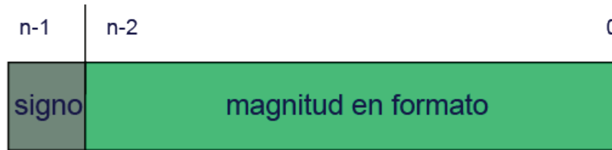


Figura 13.5 Formato de complementos a 2

¿Cómo se representa un número X en complemento a 2?

Si X>=0: 0&|X| ajustado a un total de n bits.

Si X<0: $2^n - |X|$ =1&|X|* ajustado a un total de n *bits*, donde |X|* es el complemento a 2 de la magnitud del número X. Se consigue este complemento recorriendo el número de derecha a izquierda, después de ocurrir el primer uno (el cual permanece igual), los siguientes *bits* se invierten, ceros por unos y unos por ceros.

Para un formato de 8 *bits*:

¿Cuál es el número mayor positivo que se puede representar?

$2^7 - 1 = 127$

¿Cuál es el número más negativo que se puede representar?

$-2^7 = -128$

¿Y para n *bits*?

El mayor positivo será $2^{n-1} - 1$ y el más negativo será -2^{n-1}.

©Editorial Digital Tecnológico de Monterrey

13.1.8 Suma de números en formato de complementos a 2

Si X, Y representan dos números en formato de complemento a 2.

Caso 1: x>=0, Y>=0
Es posible sumar directamente X + Y ya que sus signos son 0.

Caso 2: X>=0, Y <0, |X|>|Y|
Resultado esperado: +[|X|>|Y|]
Al sumar directamente X+Y, se realiza la operación:
$|X|+2^n+ |X|-|Y|$

Corrección necesaria: Restar 2^n para que quede solamente +[|X|-|Y|]. Esta corrección implica solamente ignorar el 1 que resulta como *carry out.*

Caso 3: X>=0, Y<0, |X|<|Y|
Resultado esperado: 2^n - [|Y|-|X|]

Al sumar directamente X+Y, se realiza la operación:
$|X|+2^n-|Y| = 2^n[|Y|-|X|]$

Corrección necesaria: Ninguna

Figura 13.6

Responde la siguiente actividad

Si se suman directamente X+Y y X<0 y Y<0, ¿qué corrección require el resultado para que quede correctamente representado en complemento a 2?

©Editorial Digital Tecnológico de Monterrey

1. En un sistema digital, los números N (que contienen signo) se representan en 5 *bits* con un formato polarizado. Los números polarizados (Np) tienen la siguiente forma:

Si N>=0, Np= 24 + N

Si N<0, Np= 24 – | N |,

es decir, Np= 24 + N

Se está analizando la manera de realizar operaciones aritméticas de números en este formato conservando su formato en la operación.

a) Indique cuál es el número de mayor magnitud positivo que se puede representar.

b) Indique cuál es el número de mayor magnitud negativo que se puede representar.

c) Indique las correcciones que tendría que hacer al proceso de suma para que al sumar dos números A y B en este formato, el resultado quede ya en este formato. Ejemplifique su respuesta.

2. Se está diseñando una computadora en la que los números enteros (N) se representan en 5 *bits* con un formato de complementos a la base disminuida en 3. Es así como en esta computadora que los números en este formato (llamados Nf) tienen la siguiente forma:

Si N>=0, Nf= N

Si N<0, Nf= 24 – 3 - /N/

©Editorial Digital Tecnológico de Monterrey

Se está analizando la manera de realizar operaciones aritméticas de números en este formato de tal manera que el resultado quede en este formato. El resultado se almacenaría en 5 *bits*.

Indique qué número está representado en 01100.

3. En cierto tanque hay sensores para observar el nivel de líquido contenido. Hay un nivel deseado que se denominará cero, el nivel más bajo es -3 y el más alto es +4. Diseñe un formato para que con el menor número de *bits* posible represente este rango (de -3 a +4).

©Editorial Digital Tecnológico de Monterrey

13.2 Números de punto fijo

Un número de punto fijo tiene una parte entera y una parte fraccional, por lo que tiene una cantidad fija de *bits* a la derecha del punto. Por ejemplo, en un formato de 7 *bits*, con dos *bits* de fracción, los números serían:

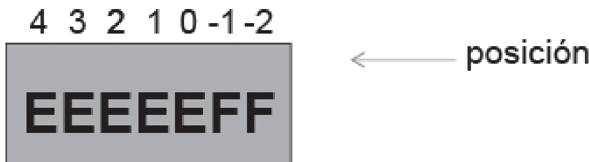


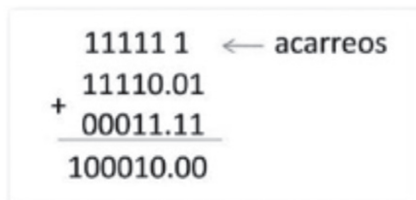
Figura 13.7 Formato de punto fijo con dos *bits* de fracción

Donde E son los *bits* de la parte entera y F de la fraccional. El punto guarda un lugar, pero no es necesario representarlo, ya que siempre tiene la misma posición.

13.2.1 Suma de números de punto fijo

La suma de números de punto fijo en un sistema posicional, ocurre como la suma de números enteros, se realiza de la posición menos significativa a la más significativa.

Por ejemplo, la suma:



Que se realiza de la posición -2 a la 4, se realiza evidentemente sin tocar el punto.

©Editorial Digital Tecnológico de Monterrey

```
  11111 1    ← acarreos
  11110 01
+ 00011 11
 ---------
 100010 00
```

El punto se conserva en la misma posición, así que la suma de números de punto fijo requiere sumadores para números enteros.

Si se capturan números de punto fijo, deben acomodarse en un formato y rellenar con ceros las posiciones de *bits* fraccionales que no se hayan completado.

Si un número de punto fijo tiene signo, puede seguirse cualquier formato de representación para números enteros. Se reitera que la única restricción que tiene el formato es que el punto implícito se conserve en la misma posición, es decir, siempre el número de *bits* fraccionales.

©Editorial Digital Tecnológico de Monterrey

13.3 Números de punto flotante

Los números de punto flotante son números expresados en notación científica bajo el siguiente formato: F*2**E**, donde F es una fracción y E es un exponente. Las características del formato que se expondrán a continuación son las que constituyen un estándar para la IEEE.

Las características son:

La fracción debe estar normalizada, esto es: ≤ | F | < 1

Tanto la fracción como el exponente se representan en complementos a 2.

El punto de la fracción es implícito, la posición más significativa de la fracción es el signo de la fracción.

La fracción estará representada en complementos a 2.

La fracción debe estar normalizada.

Para que una fracción esté normalizada.

Cuando es positiva debe ser mayor o igual a 0.1

Cuando es negativa debe ser mayor o igual a 1.0

El punto es implícito

El bit de signo y su bit de la derecha deben ser diferentes, la fracción debe iniciar con:

01... donde 0 representa el signo

10... donde 1 representa el signo

Para normalizar se hacen corrimientos a la izquierda (multiplicar por una potencia de 2) y cada vez que se haga un corrimiento se decrementa el exponente en una unidad para que el número conserve su valor.

©Editorial Digital Tecnológico de Monterrey

Para que observe fracciones positivas normalizadas de 4 *bits*:

Binaria	Decimal
0.111	7/8
0.110	6/8
0.101	5/8
0.100	4/8=1/2
0.011	No está normalizada

Tabla 13.1

Fracciones negativas normalizadas de 4 *bits*:

Binaria	Decimal
1.000	−1
1.001	−7/8
1.010	−6/8
1.011	−5/8
1.100	No está normalizada

Tabla 13.2

F = 0.101 E = 0101 N = 5/8 × 25

F = 1.011 E = 1011 N = –5/8 × 2–5

F = 1.000 E = 1000 N = –1 × 2–8

Sin normalizar: F = 0.0101 E = 0011

©Editorial Digital Tecnológico de Monterrey

N = 5/16 × 23 = 5/2

Normalizada: F = 0.101 E = 0010

N = 5/8 × 22 = 5/2

Sin normalizar: F = 1.11011 E = 1100

N = −5/32 × 2-4= −5 × 2-9

(haciendo un corrimiento de F a la izquierda) F = 1.1011 E = 1011

N = −5/16 × 2-5= −5 × 2-9

Normalizada: F = 1.011 E = 1010

N = −5/8 × 2-6= −5 × 2-9

La representación del cero es una excepción para la mantisa, según el estándar, el cero se representa como:

F = 0.000, E = 1000, es decir 0.000 × 2-8

Se podría esperar que el formato se conceptualice de la siguiente manera:

Figura 13.8 Formato de punto flotante

Suponga que en un sistema digital los números de punto flotante se representan de la siguiente manera:

Fracción (mantisa): 4 *bits* en formato de complementos a 2.

Exponente: 4 *bits* en formato de complementos a 2.

©Editorial Digital Tecnológico de Monterrey

a) ¿Cuál es el mayor número positivo que se puede representar?

b) ¿Cuál es el número negativo de mayor magnitud que se puede representar?

c) ¿Cuál es el menor número positivo que se puede representar?

d) ¿Cuál es el número negativo de menor magnitud que se puede representar?

e) Represente el número 12.5 en este formato, explique qué pasa al representar este número e indique cuál es la mayor mantisa que puede representar.

©Editorial Digital Tecnológico de Monterrey

13.4 Operaciones aritméticas: multiplicación y división

13.4.1 Multiplicación

Se estudiará esta operación con números enteros sin signo, luego se revisará el procedimiento a seguir con números con signo y con números representados en complementos a 2.

13.4.2 Multiplicación de números enteros sin signo

Hay dos formas básicas para multiplicar dos números A, B.

a) Sumar consigo mismo A, B veces.

b) Seguir el procedimiento que se utiliza cuando se multiplica a mano en el sistema decimal.

La primera es una forma sencilla pero que consume muchos ciclos de reloj, la segunda es una forma eficiente que puede traducirse a un circuito combinacional.

a) Sumar consigo mismo A, B veces

©Editorial Digital Tecnológico de Monterrey

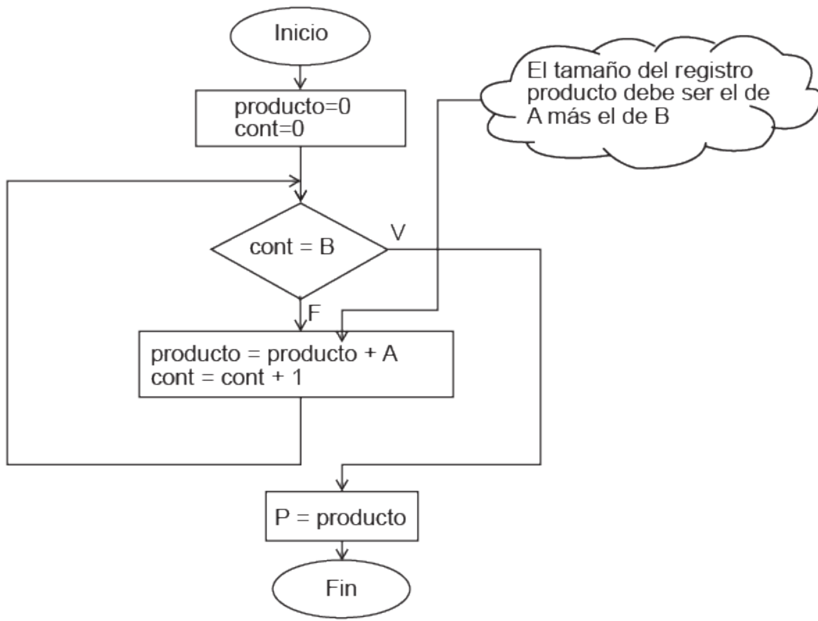


Figura 13.9 Algoritmo de multiplicación simple

La arquitectura que logra este algoritmo es la siguiente:

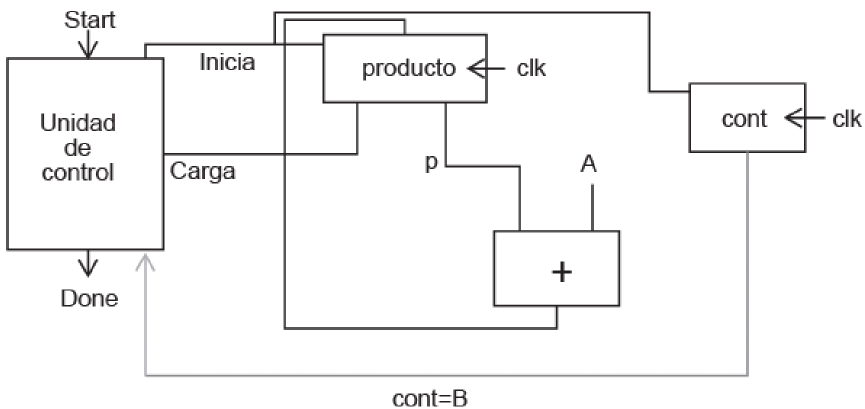


Figura 13.10 Diseño esquemático de multiplicador simple

©Editorial Digital Tecnológico de Monterrey

El diseño de la unidad de control es:

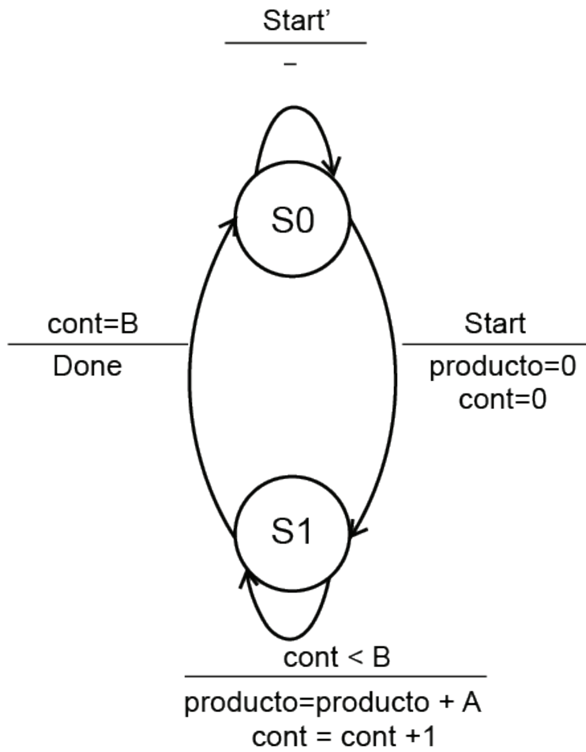


Figura 13.11 Autómata de multiplicador simple

©Editorial Digital Tecnológico de Monterrey

El código correspondiente es:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity multsuma is
    Port(start,clk : in std_logic;
         A,B: in std_logic_vector(3 downto 0);
                     P: out std_logic_vector(7 downto 0);
                     done: out std_logic);
end multsuma;

architecture Behavioral of multsuma is
     signal producto: std_logic_vector(7 downto 0);
     signal cont: std_logic_vector(3 downto 0);
     signal state: integer: = 0;

begin

     process(clk)
     begin
          if clk = '1' and clk'event then
               case state is
                    when 0 =>
                         if start = '0' then
                              state <= 0;

                         else
                              done <= '0';
                              producto <= (others => '0');
                              cont <= "0000";
                              state <= 1;
                         end if;
```

©Editorial Digital Tecnológico de Monterrey

```
                when 1 =>
                     if cont < B then
                          producto <= producto + A;
                          cont <= cont + 1;
                          state <= 1;
                     elsif cont = B then
                          done <= '1';
                          P <= producto;
                          state <= 0;
                     end if;
                when others =>
           end case;
      end if;
 end process;

end Behavioral;
```

b) El procedimiento que se utiliza cuando se multiplica a mano es el procedimiento óptimo de multiplicación

La multiplicación resuelta por medio del proceso visto en el segmento anterior requiere |Y| sumas, así que, si Y es un número de m *bits*, el máximo número de sumas requeridas sería 2**m**–1.

Existe un procedimiento mucho mejor, que solo requiere m sumas, que es el proceso que se sigue tradicionalmente para multiplicar dos números X, Y que se ilustra a continuación con una multiplicación en el sistema decimal:

```
   225
x   32
  ----
   450
  675
  ----
  7200
```

La explicación de este procedimiento, generalizado para X·Y en cualquier base β se detalla a continuación.

©Editorial Digital Tecnológico de Monterrey

Recordando que en los sistemas posicionales los números se pueden expresar como en un polinomio, el número Y se expresa como:

$$Y = y_n y_{n-1} y_{n-2} \dots y_2 y_1 y_{0\beta}$$

$$= y_n\beta^n + y_{n-1}\beta^{n-1} + y_{n-2}\beta^{n-2} + \dots + y_2\beta^2 + y_1\beta^1 + y_0\beta^0.$$

Por la propiedad conmutativa de la suma es posible expresar el número Y a partir del dígito de la posición menos significativa, de tal manera que la multiplicación resulta:

$$X \cdot Y = X \cdot (y_0\beta^0 + y_1\beta^1 + y_2\beta^2 + \dots + y_{n-2}\beta^{n-2} + y_{n-1}\beta^{n-1} + y_n\beta^n).$$

Por otra parte, la multiplicación

yiβi = yi00…00 ya que la forma de lograr que quede en la posición i es concatenando por la derecha i ceros.

Regresando a la multiplicación X·Y, las operaciones que se realizan son:

```
      X
     ·Y
      X·y0
    X·y1
   X·y2
   .........
   .........
 ____________
    Producto
```

©Editorial Digital Tecnológico de Monterrey

Nótese que el cero de la posición menos significativa se está dejando oculto, pero sí se realiza el posicionamiento hacia la izquierda de X·yi, ya que el numero está desplazado hacia la izquierda.

En particular, en base 2, yi solo puede ser 0 o 1, así que, o se requiere sumar 0 o X·2i lo que cual equivale a sumar consecutivamente X recorridos en i posiciones. Esto ocurre m veces, el número de *bits* de Y, desde i=0 hasta m-1.

Ejemplo: En este ejemplo se multiplica un número X=10011011 de 8 *bits* por un número Y= 1011 de 4 *bits*. El producto requiere a lo más 12 *bits*.

```
     10011011
x        1011
-------------
     10011011
     1001101
    00000000
   10011011
-------------
 011010101001
```

Pensando en el *hardware* que se requiere para efectuar la multiplicación por este procedimiento, la suma se puede efectuar cada dos elementos utilizando un acumulador que inicie en 0. De tal modo que la operación quede de la siguiente manera:

Un ejemplo de una multiplicación presentando sumas parciales, la primera de ellas que suma con ceros el primer operando, es la siguiente: (X es de 8 *bits*, Y de 4 *bits* y el producto de 12 *bits*).

©Editorial Digital Tecnológico de Monterrey

```
      X
    + Y
       0
-----------------
+        X•y0
        Suma1
-----------------
        X•y2
        Suma2
-----------------
+     X•y3
        Suma3
-----------------
+  ......
   ......
-----------------
   Producto
```

Multiplicación secuencial bajo este procedimiento.

El diseño esquemático del circuito secuencial que realiza la multiplicación es:

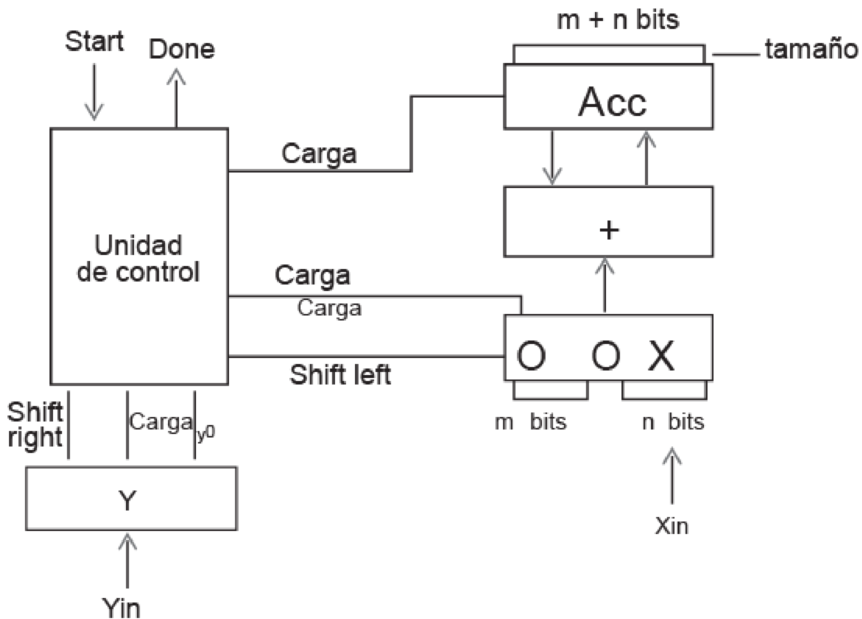


Figura 13.12 Diseño esquemático de multiplicador secuencial

©Editorial Digital Tecnológico de Monterrey

Diseño de la unidad de control

Primero se presenta un diseño para un multiplicador Y de 4 *bits*, que utiliza un estado para mostrar la multiplicación con cada *bit* de Y.

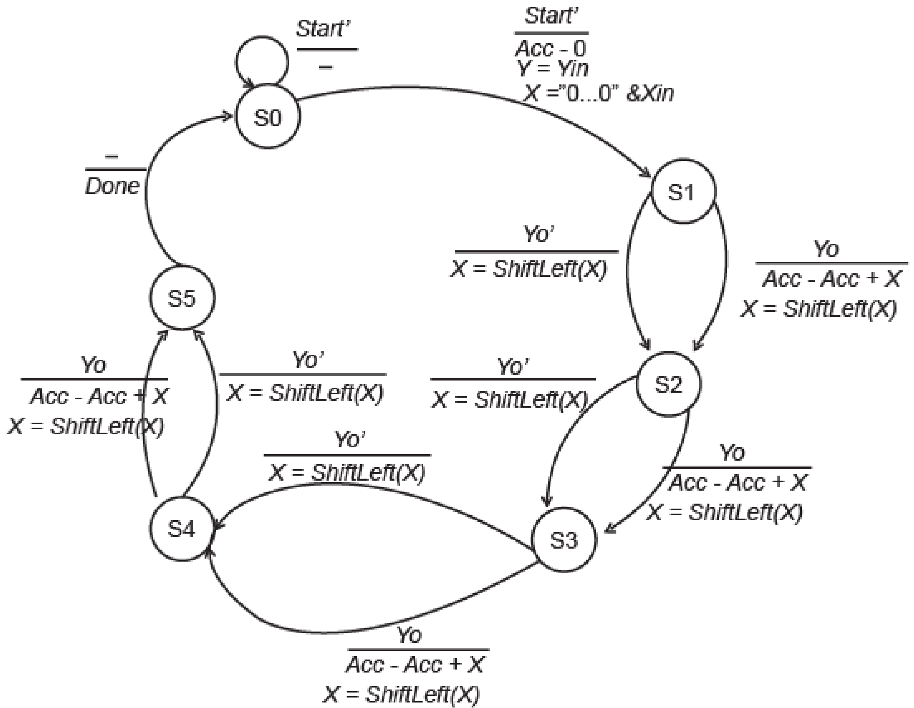


Figura 13.13 Autómata de multiplicador secuencial

Sin embargo, diseñar de esta manera la unidad de control resulta tedioso, ya que si, por ejemplo, Y fuera de 16 *bits*, se requerirían 18 estados. Entonces es posible utilizar un contador de apoyo que lleve la cuenta de la cantidad de *bits* de Y, de tal manera que el diseño esquemático resulta:

©Editorial Digital Tecnológico de Monterrey

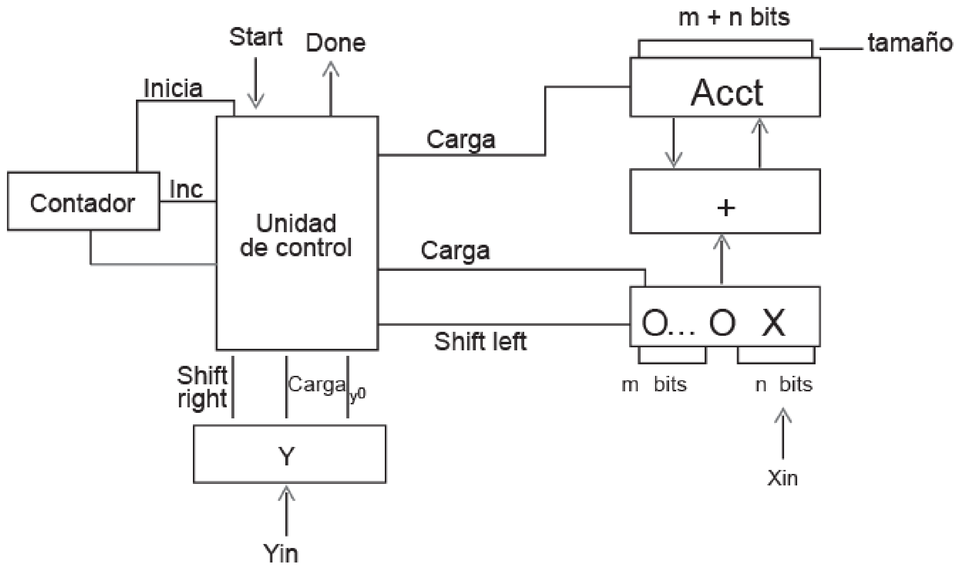


Figura 13.14 Diseño esquemático de multiplicador secuencial

Y el diseño de la unidad de control se reduce a dos estados:

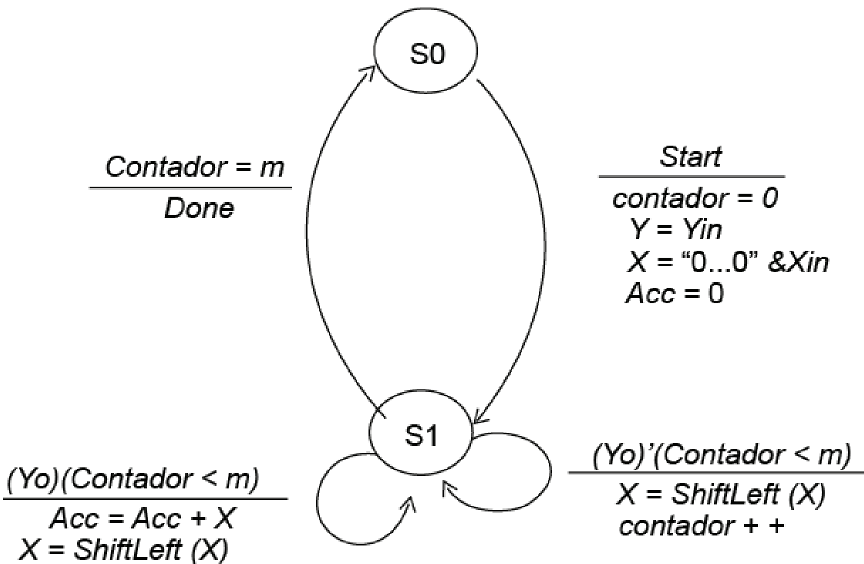


Figura 13.15 Autómata de multiplicador secuencial

©Editorial Digital Tecnológico de Monterrey

Aunque todavía es posible reducir el diseño esquemático, porque como al registro que aloja a Y se le introducen ceros, cuando Y llegue a 0 termina la multiplicación. De hecho, si desde el inicio Y es 0 la multiplicación no se realiza y el resultado es 0. Con esta consideración el contador no es necesario.

El diseño esquemático resulta:

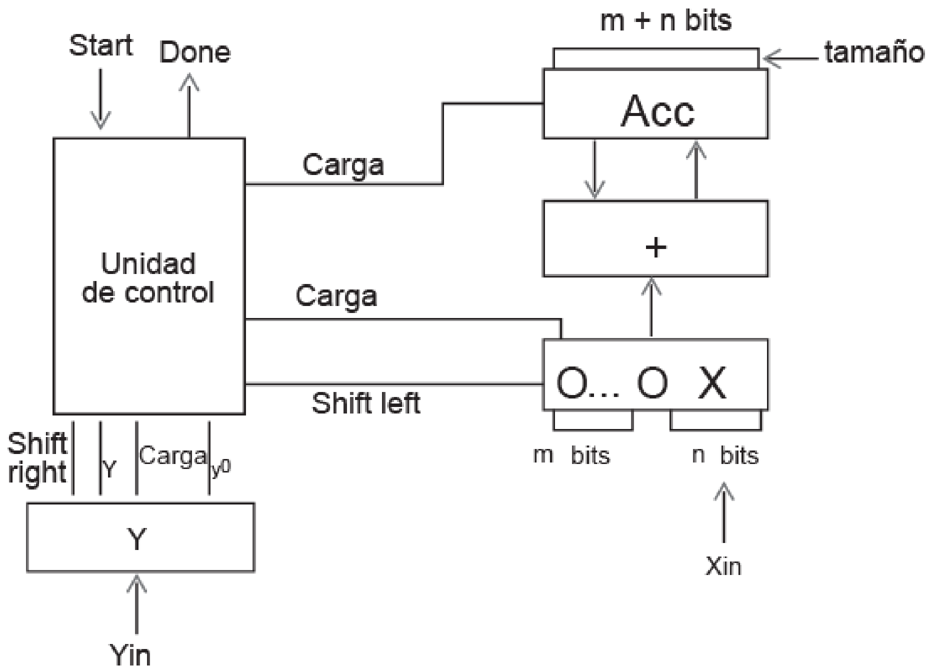


Figura 13.16 Esquemático de multiplicador secuencial

©Editorial Digital Tecnológico de Monterrey

Y el diseño de la unidad de control es:

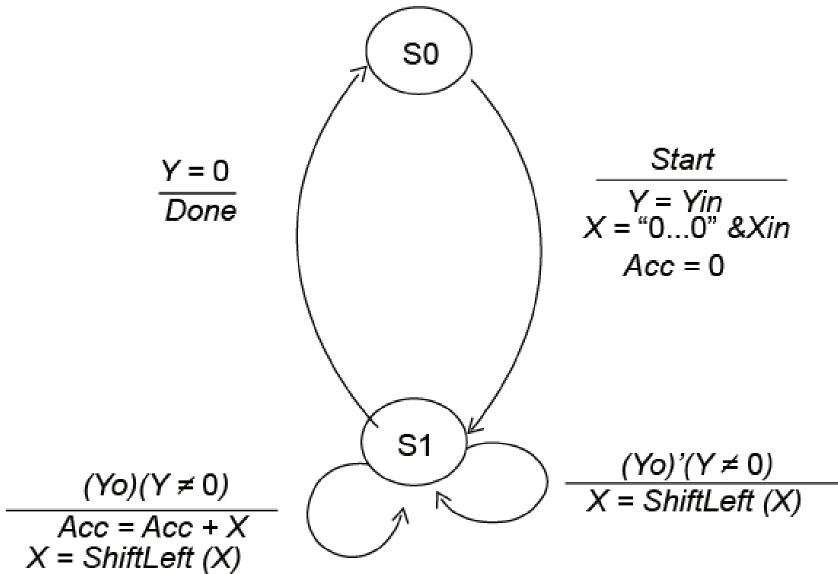


Figura 13.17 Autómata de multiplicador secuencial

a) Efectúe (a mano) las siguientes multiplicaciones de números binarios (dejando la respuesta en binario):

1001*1000=

101*100=

111*10000=

b) Generalice: X*2n= (utilice el operador & de VHDL)

c) Si X es un número binario almacenado en un registro, ¿qué operación, en *hardware*, habría que hacer sobre el registro para obtener X*2n?

©Editorial Digital Tecnológico de Monterrey

d) Efectúe (a mano) las siguientes multiplicaciones de números binarios (dejando la respuesta en binario):

1001*1010=

101*101=

111*10101=

Nota: en la operación X*Y=Z, X es el multiplicando, Y es el multiplicador, Z es el producto.

e) Para un multiplicador de 4 *bits*, ¿cuántas sumas debe realizar para obtener el resultado?

Multiplicación combinacional bajo este procedimiento

Con estos circuitos secuenciales se ha reproducido el procedimiento que se realiza para multiplicar a mano; sin embargo, un circuito que multiplique puede ser diseñado bajo un enfoque combinacional. Bajo esta perspectiva, la multiplicación se realiza por completo en una pasada, solo sufre los retrasos de la propagación de los acarreos. Este circuito es el mejor por ser el más rápido.

Si se considera un número X de 4 *bits* y un número Y también de 4 *bits*, la multiplicación se expresa como:

©Editorial Digital Tecnológico de Monterrey

				X_3	X_2	X_1	X_0
				Y_3	Y_2	Y_1	Y_0
				X_3Y_0	X_2Y_0	X_1Y_0	X_0Y_0
			X_3Y_1	X_2Y_1	X_1Y_1	X_0Y_1	
			C_{12}	C_{11}	C_{10}		
		C_{13}	S_{13}	S_{12}	S_{11}	S_{10}	
		X_3Y_2	X_2Y_2	X_1Y_2	X_0Y_2		
		C_{22}	C_{21}	C_{20}			
	C_{23}	S_{23}	S_{22}	S_{21}	S_{20}		
	X_3Y_3	X_2Y_3	X_1Y_3	X_0Y_3			
	C_{32}	C_{31}	C_{30}				
C_{33}	S_{33}	S_{32}	S_{31}	S_{30}			
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0

Las Sij representan las sumas parciales, y las Cij sus acarreos.

Para mostrar más claramente al circuito, aquí se presenta su diseño esquemático basado en compuertas, *Full Adders* (FA) y *Half Adders* (HA):

©Editorial Digital Tecnológico de Monterrey

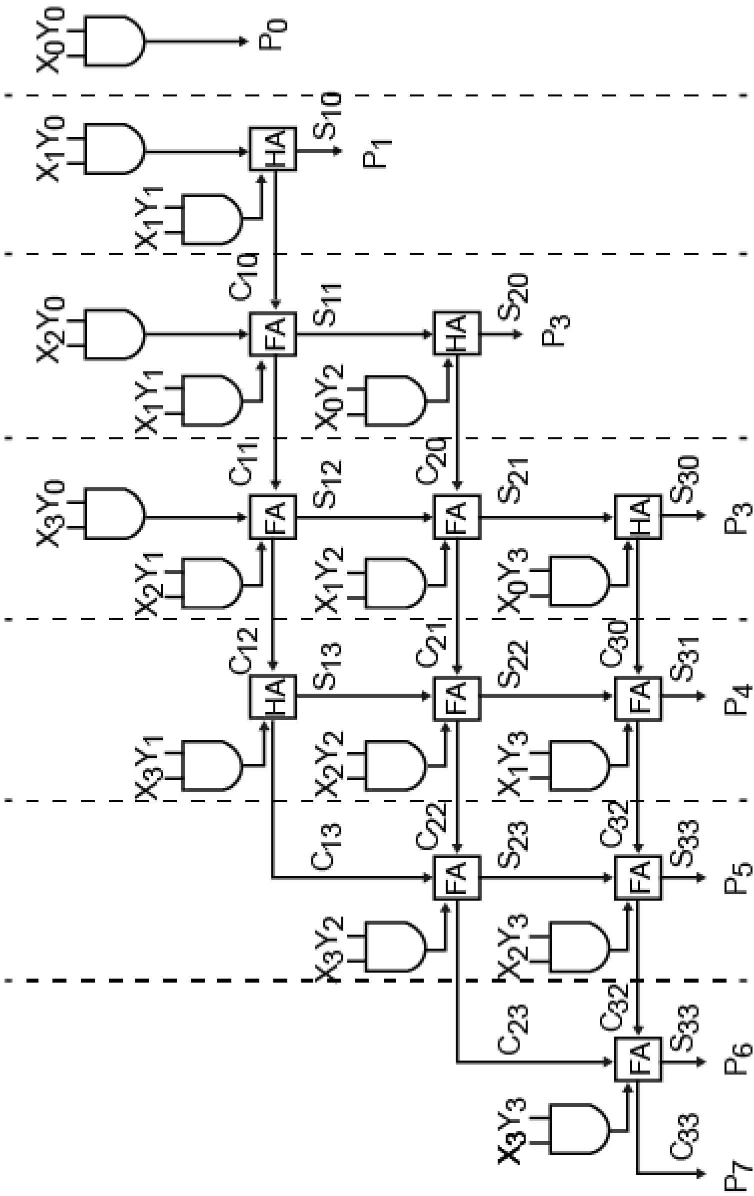


Figura 13.18 Esquemático de multiplicación combinacional

©Editorial Digital Tecnológico de Monterrey

Como ya se mencionó, con cualquiera de los métodos que se utilice para multiplicar es posible verificar que al multiplicar X por Y, si X tiene una extensión de n *bits* y Y de m *bits*, la longitud requerida para la salida es m+n *bits*.

Afortunadamente en VHDL es posible utilizar el operador * para expresar una multiplicación. El único requisito es que el producto tenga el tamaño de la suma de los tamaños de los operandos, no es necesario que los operandos sean de igual longitud. Una multiplicación queda definida como:

```
Producto <= A*B;
```

El sistema de desarrollo ya cuenta con el *software* necesario para expandir el circuito combinacional correspondiente, como el que se muestra en la gráfica anterior.

Multiplicación de números enteros con signo en formato de complementos a 2

Longitud de palabra requerida para almacenar el producto. Como ambos operandos tienen signo, para un multiplicando de longitud m y un multiplicador de longitud n el tamaño que requiere el producto es: m + n − 1.

Para calcular el número de operaciones óptimo partiendo de dos operandos que se encuentran en complementos a 2, partiremos de:

©Editorial Digital Tecnológico de Monterrey

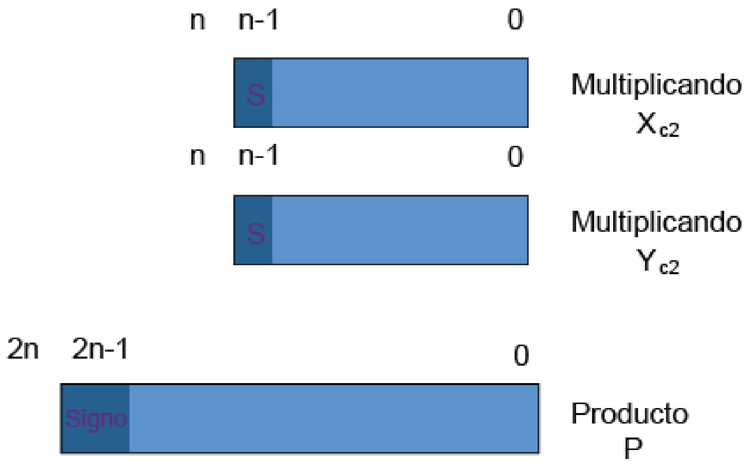


Figura 13.19 Multiplicación en formato de complementos a 2

El análisis matemático de las multiplicaciones de números representados en complementos a 2 se basará en casos.

– Caso 1: X>=0, Y>=0

Ya que los signos son cero y la magnitud está inalterada la multiplicación se realiza directamente como si fueran números sin signo.

```
Caso 1: X>=0, Y<0

Alternativa 1: multiplicación de las magnitudes:
Se requiere:

     - Complementar Yc2
     - Realizar multiplicación
     - Complementar producto

Alternativa 2: multiplicación directa en complementos
¿Qué ocurre si multiplicamos sin complementar Yc2?
```

©Editorial Digital Tecnológico de Monterrey

Caso 2: X>=0, Y<0

Alternativa 1: multiplicación directa en complementos
¿Qué ocurre su multiplicamos sin complementar Y?

Multiplicamos: $X_{c2} * Y_{c2} = |X| * (2^n - |Y|)$
$= 2^n|X| - |X||Y|$

y buscamos: $2^{2n} - |X||Y|$
Corrección necesaria: $2^{2n} - X||Y| =$
$2^{2n} + 2^n|X| - 2^n|X| - |X||Y|$

$2^{2n} - |X||Y| = X_{c2} * Y_{c2} + (2^{2n} - 2^n|X|)$
$= X_{c2} * Y_{c2} + 2^{2n}X_{c2}$

Alternativa 2:

- Complementar Y_{c2}
- Realizar multiplicación
- Complementar producto
- Implica dos operaciones extra

Alternativa3:
$2^{2n} - |X||X| = X_{c2} * Y_{c2} + 2^{2n} - 2^n|X|$
$= X_{c2} * Y_{c2} + 2^{2n}X_{c2}$

- Implica una operación extra

©Editorial Digital Tecnológico de Monterrey

Caso 3: X<0, Y>0

Alternativa 1: multiplicación de las magnitudes:
Se requiere:

- Complementar X_{c2}
- Realizar multiplicación
- Complementar producto

Alternativa 2:Multipliación directa en complementos
¿Qué ocurre si multiplicamos sin complementar Y?

Multiplicamos: $(2^n-|X|)*|Y| = 2^n|Y|-|X||Y|$
y buscamos: $2^{2n}-|X||Y|$
Corrección necesaria: $2^{2n}-|X||Y|=$
$2^{2n} + 2^n |Y|-2^n |Y|-|X||Y|$
$2^{2n} -|X||Y| = X_{c2}*Y_{c2} + 2^{2n}-2^n|Y|$

Implica una operación extra y en *hardware* implica restar Y desplazada n posiciones.

Otra posible corrección:
Multiplicamos: $(2^n-|X|)*|Y| = 2^n|Y|-|X||Y|$
y buscamos: $2^{2n}-|X||Y|$
Corrección necesaria: $2^{2n}-|X||Y|=$
$2^{2n} + 2^{2n} |Y|-2^{2n} |Y|-|X||Y|$
$2^{2n} -|X||Y| = X_{c2}*Y_{c2} + 2^{2n}-2^{2n}|Y|$

Operación que no requiere corrección por estar fuera del tamaño del producto

Conclusión:

Alternativa 1:
- Complementar X_{c2}
- Realizar multiplicación
- Complementar producto
  - Implica dos operaciones extra

Alternativa 2:
2^{2n}- Otra posible corrección. $|X||Y| = X_{c2}*Y_{c2} + 2^{2n}-2^{2n}|Y|$
- no implica operaciones extra

©Editorial Digital Tecnológico de Monterrey

Caso 4: X<0, Y<0

Alternativa 1: multiplicación de las magnitudes:
Se requiere:
- Complementar X
- Complementar Y
- Realizar multiplicación

Alternativa 2:Multiplicación directa en complementos
¿Qué ocurre si multiplicamos sin complementar X ni Y?

Multiplicamos: $(2^n-|X|)*(2^n-|Y|)$
$= 2^{2n}-2^n|X|-2^n|Y| + |X||Y|$
y buscamos: $|X||Y|$
Corrección necesaria: $|X||Y| =$
$= 2^{2n}-2^{2n}-2^n |X|+ 2^n |X|-2^n|Y|+ |X||Y|$
$|X||Y| = X_{c2}**Y_{c2}*-2^{2n} + 2^n|X| + 2^n|Y|$
implica dos operaciones extra
Otra posible corrección
Multiplicamos: $(2^{2n}-|X|)*(2^n-|Y|)$
$= 2^{3n} -2^n |X|-2^{2n}|Y| + |X||Y|$
y buscamos: /X//Y/
Corrección necesaria: /X//Y/=
$2^{3n}-2^{3n}-2^n-|X| + 2^n |X|-2^{2n} |Y| + 2^{2n} |Y|+ |X||Y|$
$|X||Y| = X_{c2}*Y_{c2}* + 2^n|X|-2^{3n} + 2^{2n}|Y|$
solo se requiere una corrección que no sale del formato

Alternativa 1:
- Complementar X
- Complementar Y
- Realizar multiplicación
- Implica dos operaciones extra

Alternativa 2:
- $|X||Y| = X_{c2}*Y_{c2}* + 2^n|X|-2^{3n} + 2^{2n}|Y|$
- no implica operaciones extra

©Editorial Digital Tecnológico de Monterrey

```
Resumen

- Caso 1: X>=0, Y>=0
  Misma solución que números sin signo.

- Caso 2: X>=0, Y<0
  2^2n-|X||Y| = Xc2*Yc2 + |X|

- Caso 3: X<0, Y>=0
  2^2n-|X||Y| = Xc2*Yc2 no implica operaciones extra

- Caso 4: X<0, Y<0
  |X||Y| = Xc2*Yc2 + 2^n|X|
```

Responde la siguiente actividad.

1. Los siguientes números se encuentran en formato de complementos a 2, multiplíquelos conservando su formato y obtenga la respuesta en el mismo formato en 8 *bits*.

0111*0101=

1001*0100=

1111*1101=

2. Se tienen números de 4 *bits* en formato de complementos a 2. A y B son números en este formato. Diseñe en VHDL el cálculo de su producto y déjelo en P. P debe quedar en 8 *bits* en formato de complementos a 2.

3. En un sistema digital se manejan números de 12 *bits* en punto fijo sin signo. 8 *bits* corresponden a la parte entera y a 4 a la fraccional, el punto está implícito. Los números en este formato son: Nf=EEEEEEEEFFFF, donde E es la parte entera y F la fraccional.

©Editorial Digital Tecnológico de Monterrey

Para este formato:

a) Diseñe en VHDL el siguiente cálculo: Pf= Af*Pf tanto Af como Pf están en este formato y deben conservarlo. No se preocupe por el desborde, Pf debe tener un tamaño suficientemente grande para soportar la operación. Considere que Af solo usa cuatro de los enteros y los cuatro *bits* más significativos son cero.

b) Diseño en VHDL el siguiente cálculo: Pf= Af+Pf tanto Af como Pf están en este formato y deben conservarlo. No se preocupe por el desborde, Pf debe tener un tamaño suficientemente grante para soportar la operación. Considere que Af solo usa cuatro de los enteros y los cuatro *bits* más signficativos son cero.

13.5 División

13.5.1 División de números sin signo

A diferencia de las operaciones de suma, resta y multiplicación, la división es una operación secuencial.

Revisemos un poco la división en el sistema decimal:

```
                   362   ← cociente
divisor → 15 | 5435   ← dividendo
               -45
                 93
                -90
                  35
                 -30
                   5   ← residuo
```

©Editorial Digital Tecnológico de Monterrey

Ahora revisemos la división binaria:

```
           1111
1100 |10110100
      -1100
       10101
       -1100
        10010
        -1100
         01100
         -1100
          0000
```

Podemos observar que la división se realiza de manera similar.

Se sugiere el siguiente esquemático para conseguir el cociente y residuo de la división X/Y:

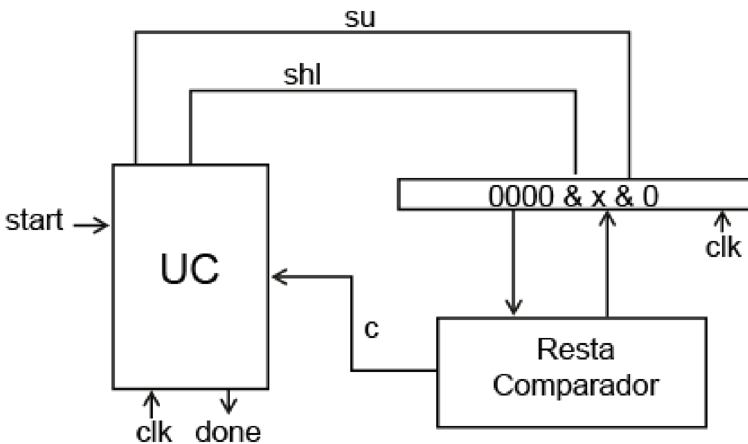


Figura 13.20 Esquemático de divisor

La manera en cómo funciona la unidad de control al dividir 10111110 entre 1001 se describe de la siguiente manera:

©Editorial Digital Tecnológico de Monterrey

Tomemos a M como el número de *bits* del dividendo (8) y a N como el número de *bits* del divisor (4). Además, tomemos la señal C como el *bit* que se agregará en cada *shift left* efectuado y un contador que definirá que se ha obtenido el residuo y cociente correctamente.

En el registro lo primero que se puede observar es que se introduce el dividendo concatenado con N ceros. Posteriormente se están comparando constantemente los 5 *bits* más significativos del registro con 01001, el cual es el divisor con un cero concatenado, hasta que el contador sea igual a M.

Ahora, mientras que los 5 *bits* más significativos del registro sean menores a 01001, la unidad de control continuará enviando la señal de realizar un *shift left* concatenando C con valor de cero. Sin embargo, si se llega a encontrar que los 5 *bits* más significativos del registro son igual o mayor a 01001, entonces la unidad de control se encargará de mandar la señal de cargar el resultado de la resta (su) en los 5 *bits* más significativos del registro y a la vez se concatenará C con un valor de 1.

Finalmente, se almacena en el registro el residuo y el cociente de la división cuando el contador es igual a M. Cabe destacar que el dividendo y el cociente tienen M *bits*, mientras que el divisor y el residuo N *bits*.

©Editorial Digital Tecnológico de Monterrey

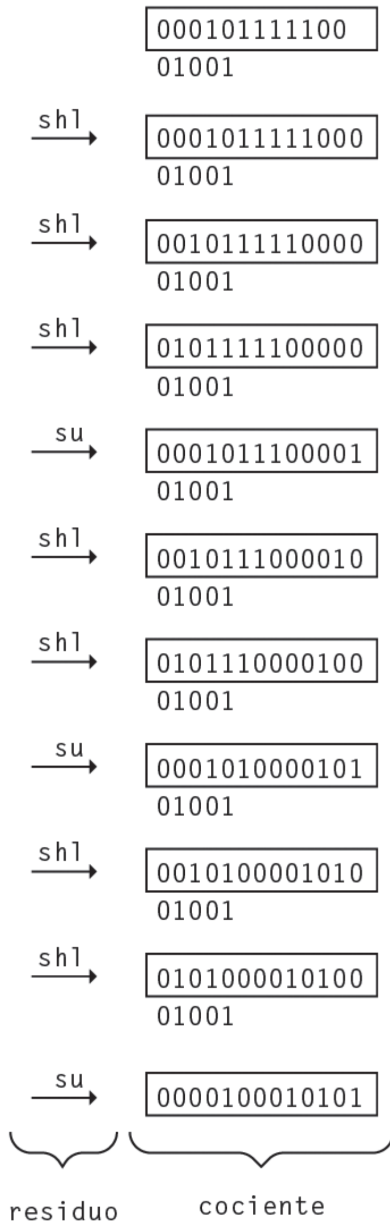


Figura 13.21

©Editorial Digital Tecnológico de Monterrey

Enseguida se muestra paso a paso como efectúa la unidad de control la división descrita anteriormente:

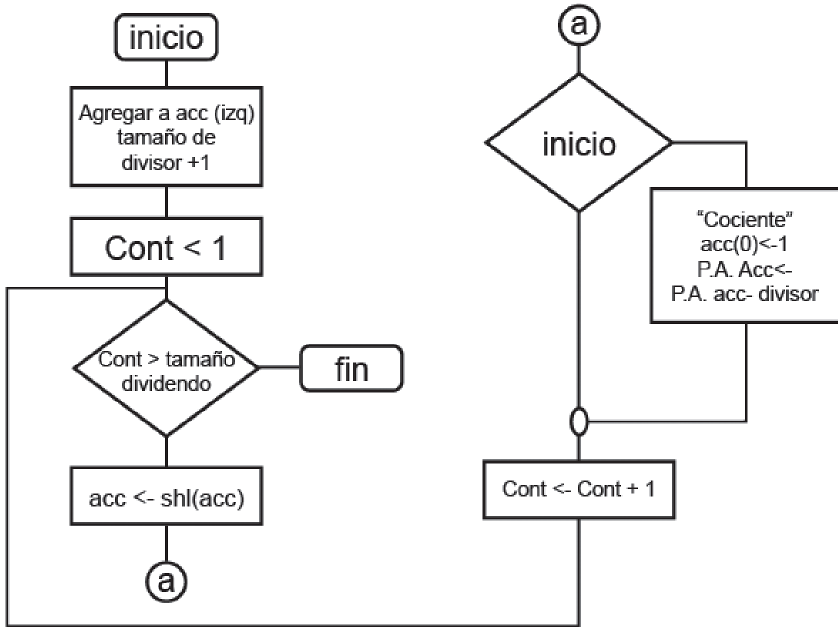


Figura 13.22 Algoritmo de division

©Editorial Digital Tecnológico de Monterrey

Una manera de representar la división mediante una máquina de estados es la siguiente:

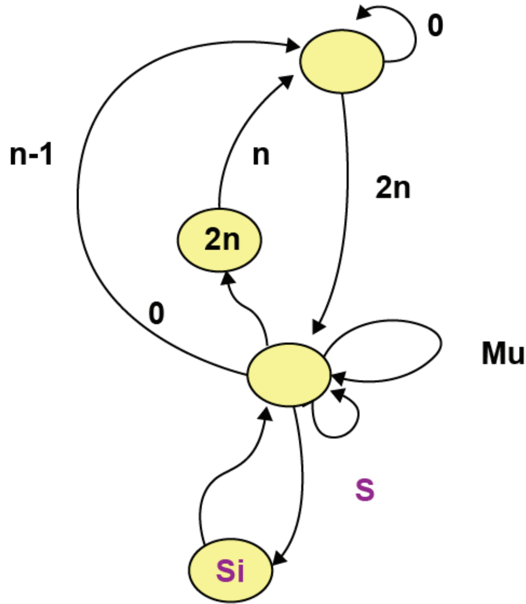


Figura 13.23 Autómata de división

©Editorial Digital Tecnológico de Monterrey

La división codificada en VHDL es:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity div2 is
    Port(x: in std_logic_vector(7 downto 0);
         y: in std_logic_vector(3 downto 0);
         acc: inout std_logic_vector(12 downto 0);
                   start: in std_logic;
                   clk: in std_logic;
                   done: out std_logic);
end div2;

architecture Behavioral of div2 is
type STATE_TYPE is (S0, S1, S2, S3);

attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of STATE_TYPE: type is "00 01 10
11";

signal state: STATE_TYPE:=S0;
signal k, c: std_logic;
signal count:std_logic_vector(2 downto 0);
begin
process(start, clk)
begin
  if clk = '1' and clk'event
        then
          case state is
           when S0 => if start = '1' then
                              state <= S1;
                              acc <= "0000"&x&'0';
                              count <= "000";
```

©Editorial Digital Tecnológico de Monterrey

```
                              done <= '0';
                              else state<=S0;
                        end if;
            when S1 => if k = '0' and c = '0' then
                              acc<=acc(11 downto 0)&'0';
                              count <= count + 1;
                              state <= S1;
                              elsif k = '0' and c = '1' then
                        acc <= (acc(12 downto 8)-('0'&y))&acc(7 downto 1)&'1';
                              state <= S2;
                              elsif k = '1' and c = '0' then
                              state <= S0;
                              done <= '1';
                              elsif k = '1' and c = '1' then
                        acc< = (acc(12 downto 8)-('0'&y))&acc(7 downto 1)&'1';
                              state <= S3;
                        end if;
            when S2 => acc <=a cc(11 downto 0)&'0';
                               count <= count+1;
                        state <= S1;
                   when S3 => done <= '1';
                        state <= S0;
end case;
  end if;
  end process;
  c <= '1' when (acc(12 downto 8) >= ('0'&y)) else '0';
  k <= '1' when (count="111") else '0';
end Behavioral;
```

Este estilo de diseño es fácil de codificar y es muy seguro en la etapa de implementación del circuito.

Responde la siguiente actividad.

1. Efectúe (a mano) las siguientes divisiones de números binarios con signo, en representación de complementos a 2 y divisor de cuatro *bits.*

00001001 / 0011 =

01010000 / 0100 =

1111 / 0001 =

©Editorial Digital Tecnológico de Monterrey

2. Se tienen números de cuatro *bits* en formato de complementos a 2.

a) Indique cuál es la respuesta correcta para la siguiente división: 0101/1100

Cociente=

Residuo =

b) ¿Cuál sería la respuesta si divide directamente, es decir, si deja el dividendo en complementos a 2?

c) ¿Será posible implementar la corrección algebraica que se debe hacer para que la respuesta sea correcta en caso de dividir directamente en complementos a 2?

3. El siguiente propone resolver la multiplicación de una manera ineficiente. Consiste en realizar la operación A*B sumando A consigo misma B veces. Diseñe el circuito, detalle el procedimiento a seguir, la arquitectura y el diseño de la unidad de control.

4. Este problema propone realizar una división con un método muy ineficiente, consiste en realizar restas sucesivas. Si se va a realizar la operación X/Y restando Y de X todas las veces que sea posible dejando un resultado positivo de la resta. La cuenta del número de restas que sean posibles constituye el cociente, la última resta antes de tener un resultado negativo es el residuo.

©Editorial Digital Tecnológico de Monterrey

Diseñe el circuito, detalle el procedimiento a seguir, la arquitectura y el diseño de la unidad de control.

5. En este problema se le pide calcular la raíz cuadrada de un número N restando los números impares de N a partir de 1, y contando el número de restas que sea posible antes de que ocurra un resultado negativo.

Diseñe el circuito, detalle el procedimiento a seguir, la arquitectura y el diseño de la unidad de control.

6. Diseñe un circuito que resuelva X!, la entrada X es un número de 4 *bits* sin signo.

Diseñe el circuito, detalle el procedimiento a seguir, la arquitectura y el diseño de la unidad de control. Recuerde que X!=1*2*3*…*X, o bien, X!=(X-1)!*X

7. Es posible el cálculo de funciones trascendentales (no polinomios) utilizando expansiones de Taylor. Por ejemplo, para calcular ex es posible aproximar este cálculo a través del siguiente cálculo:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Se utilizará esta fórmula para el cálculo de la exponencial de números x positivos, sin signo, de 4 *bits*.

©Editorial Digital Tecnológico de Monterrey

División de números en formato de complementos a 2

La naturaleza de la división provoca que no sea posible dividir directamente en formato de complementos, no hay manera de deshacer el error con operaciones simples. Por este motivo, es necesario realizar la división con números positivos, así que el procedimiento requiere pasar los números a dividir a formato de signo magnitud, realizar la división con las magnitudes y calcular por separado el signo resultante. Si el cociente se requiere en formato de complementos a 2, entonces habrá que pasar el resultado de formato de signo magnitud a formato de complementos a 2.

©Editorial Digital Tecnológico de Monterrey

13.6 Operaciones de números en formato de punto flotante

La implementación de las operaciones en punto flotante queda fuera del alcance de este libro, solamente se mencionará su forma y se comentará su realización.

Curiosamente la suma y la resta son las operaciones más complejas en punto flotante.

Si se desean sumar dos números $F_1*2E_1 + F_2*2E_2$ el procedimiento a seguir requiere que uno de los números de menor exponente ajuste su fracción hasta que tenga el exponente del número mayor. Una vez hecho este ajuste las fracciones se pueden sumar, y una vez sumadas de nuevo hay que normalizar el número resultante. El procedimiento de la resta es el mismo que el de la suma.

Para multiplicar dos números en punto flotante $F_1*2E_1 * F_2*2E_2$ el procedimiento es más sencillo, la operación a realizar es: $(F_1* F_2)*2$ (E1+E2). Luego de hacer este cálculo el resultado se normaliza.

Para dividir dos números en punto flotante F_1*2E_1 / F_2*2E_2 el procedimiento también es sencillo, la operación a realizar es: $(F_1/F_2)*2$ (E1−E2). Luego de hacer este cálculo el resultado se normaliza.

El detalle que hay que cuidar en los circuitos de estas operaciones es la determinación de si hay o no desborde en el resultado (*overflow*).

Ejercicios

En un sistema digital se cuenta con números de punto flotante, con exponente en complementos a 2 de 4 *bits* y fracción en complementos a 2 de 6 *bits*.

©Editorial Digital Tecnológico de Monterrey

Calcule el resultado de las siguientes operaciones de números en este formato:

1) 0011 011100 + 0111 010100

2) 0011 011100 + 0100 101100

3) 0100 010000 / 1100 010000

4) 0100 011000 * 0110 010000

5) 1100 011000 * 1101 010000

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Actividad integradora del capítulo 13

A continuación, se muestran tres aplicaciones en tres ramas diferentes. En cada una se ofrece una breve introducción y un problema, del cual se le requiere definir una solución (solo el diseño de la unidad de control).

1. Aplicación de control.

Introducción

Esta gráfica modela un sistema de control general:

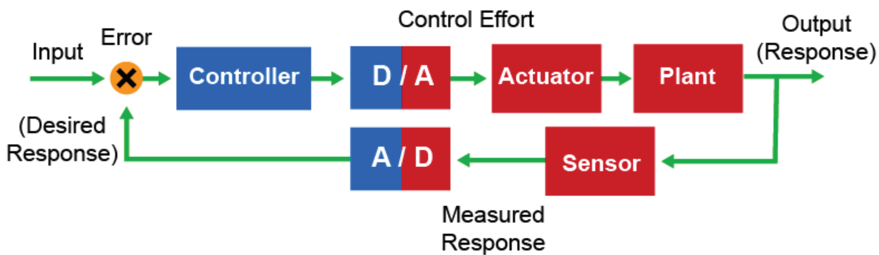


Figura 13.24

Un lazo de control consiste de tres partes:

1. Medias que provienen de sensores o transductores que están conectados al proceso.

2. Un criterio de decisión para producir una señal de control

3.Las acciones que se efectúan por los dispositivos actuadores, tales como válvulas, brazos, palancas.

©Editorial Digital Tecnológico de Monterrey

A medida que el controlador lee los sensores, resta la meida del punto de control, o set point para determinar el error. Utilice el error para calcular la corrección de la medida y toma acciones a través de los actuadores para corregir el error. Por ejemplo, suponga que un tanque esutilizado para suministrar agua a varios lugares de una planta y es necesario manatener el nivel de agua constante. Es posible utilizar un sensor para medir el nivel del agua y enviar acciones de control a una válvula para que permita la entrada de agua hasta llegar al nivel deseado.

Algunos sistemas de control utilizan controladores en red o en cascada, en configuración maestro-esclavo (los controles esclavos utilizan las señales generadas por los controladores maestros). Esta es una configuración frecuente en motores.

Los controles acoplados y en cascada son comunes en procesos químicos, ventilación, calefacción, aire acondicionado, y otros.

Es válido tomar medidas en intervalos discretos de tiempo. El siguiente es un modelo de un controlador.

$$\mathrm{Output}_{n+1} = \mathrm{Output}_{n} + K_p e_n + K_d(e_n - e_{n-1})$$

Problema:

oKd = 2

oKp = 5

o= 0.50 segundos, significa que cada 0.5 seg. Se lee la medida del adc, puede suponer que cuenta con un reloj con ese período.

oSalida deseada = 2.0 equivalente a 11110 en el adc (convertidor análogodigital)

Suponga que se desea controlar un brazo de robot que realiza un trabajo de ensamble muy preciso. Sin pretender detallar el sistema, se realiza una medida con un convertidor análogo digital de 5 *bits*. Se requiere calcular una salida (Output n+1) también de 5

©Editorial Digital Tecnológico de Monterrey

bits. Calcule el error restando la medida que realice con el cad (adc en inglés) con la medida ideal que debería ocurrir. La salida alimentaría a un convertidor digital análogo para realizar una acción de control. Utilice los valores proporcionados para las constantes Kd y Kp. Diseñe una arquitectura esquemática para este problema y la máquina de estados necesaria para la unidad de control. Las entradas son: la señal de reloj y la medida del adc, la salida es output n+1. Puede suponer que tanto el dac como el adc responden en un tiempo mucho menor a 0.5 seg.

2. Aplicación de procesamiento digital de señales.

Introducción

En procesamiento de señales, la función de un filtro es remover partes que no se desean de una señal, tales como el ruido, o bien extraer partes útiles de la señal, tales como componentes que tienen cierta frecuencia.

El siguiente diagrama de bloques muestra esta idea básica.

Figura 13.25

Un filtro digital utiliza un modelo numérico para realizar cálculos de valores de la señal que se han muestreado utilizando un convertidor análogo digital (ADC por sus siglas en inglés). La salida se convierte en una señal utilizando un convertidor digital análogo (DAC por sus siglas en inglés).

En un filtro digital, una señal es representada por una secuencia de números, no es una señal continua.

©Editorial Digital Tecnológico de Monterrey

El siguiente diagrama muestra la acción de un filtro.

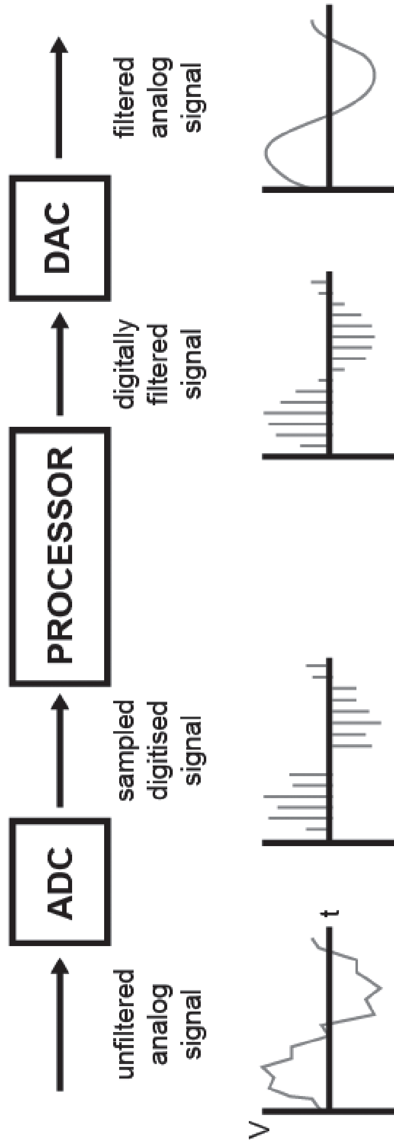


Figura 13.26

©Editorial Digital Tecnológico de Monterrey

Orden de un filtro digital

El orden de un filtro digital está determinado por el número de entradas previas utilizadas para calcular la salida presente.

Ejemplo, (orden 0): $y_n = x_n$

Ejemplo, (orden 0): $y_n = Kx_n$

Ejemplo, (orden 1): $y_n = x_{n-1}$

Ejemplo, (orden 1): $y_n = x_n - x_{n-1}$

Ejemplo, (orden 1): $y_n = (x_n + x_{n-1}) / 2$

Ejemplo, (orden 2): $y_n = (x_n + x_{n-1} + x_{n-2}) / 3$

Ejemplo, (orden 2): $y_n = (x_n - x_{n-2}) / 2$

Problema:

Suponga que desea filtrar una señal de audio. La señal (medida como un voltaje) la recibe a través de un adc de 5 *bits* (el mismo del problema anterior). Simplificando el problema, aplique un filtro que calcule una señal yn de la siguiente manera: $\mathbf{y_n = (x_n - x_{n-2}) / 2}$**.**

La entrada a su circuito es $\mathbf{x_n}$ y la salida $\mathbf{y_n}$ calculada directamente de la medida que proporciona el adc, sin hacer ningún escalamiento. Las medidas se realizan cada 0.2 segundos y justo cuenta con una señal de reloj con ese periodo. Diseñe una arquitectura esquemática para este problema y la máquina de estados necesaria para la unidad de control.

3. Reconocimiento de patrones.

Un algoritmo simple y razonablemente efectivo para ubicar si dos imágenes digitalizadas coinciden es el siguiente:

©Editorial Digital Tecnológico de Monterrey

Por simplificar el algoritmo suponga que la imagen consiste de un vector (en vez de matriz) de 200*200= 40000 números enteros que varían de 0 a 255 (cada número corresponde a la información de color de un píxel). Dada la información de dos imágenes, una representada por el vector X y la otra por el vector Y, indicar si corresponden a la misma fuente. El algoritmo consiste en conseguir el producto punto de ambos vectores normalizados, si la sumatoria obtenida por el producto punto vector es cercana a 1 entonces se acepta que coinciden.

La cantidad de operaciones serían:

40000

Σ $(x_i - x)$ $(y_i - y)$ x representa el promedio de todos los valores de x_i

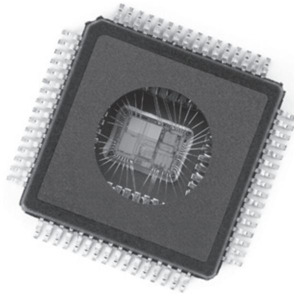
i=0 y representa el promedio de todos los valores de y_i

Diseñe una unidad de control que lleve a la construcción de un circuito que pueda ser incluido en un dispositivo móvil como complemento de seguridad (como clave de acceso) (sería aplicable a huellas, rostro, firma, iris, etc, con el dispositivo adecuado para captar la imagen).

©Editorial Digital Tecnológico de Monterrey

Conclusión del capítulo 13

Las aplicaciones que justifican el diseño de una arquitectura son aquéllas que requieren el cálculo de operaciones aritméticas en forma repetida. En los problemas propuestos se muestra un conjunto de problemas que constituyen desarrollos típicos que requieren arquitecturas a la medida, si es que requieren tiempos de respuesta mínimos.



©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Glosario (General)

A

ABEL (Advanced Boolean Expression Language)

Lenguaje inicialmente creado en 1983 para programar PLD (*Programmable Logic Device*) por Data I/O Corporation, en Redmond, Washington. Es un lenguaje descriptor de *hardware* de más bajo nivel que VHDL y Verilog. Los circuitos se describen en álgebra booleana y tablas de verdad.

Acarreo (*carry*)

Es el *bit* que excede a una posición y que se conecta como entrada al siguiente módulo en una operación aritmética binaria.

Álgebra booleana

El álgebra de la lógica consiste en establecer relaciones entre variables booleanas a través de operadores lógicos.

Astable

Es un circuito multivibrador que no tiene ningún estado estable, oscila entre un nivel y otro, lo que significa que posee dos estados "quasi-estables" entre los que conmuta, permaneciendo en cada uno de ellos un tiempo determinado. El tiempo que transcurre entre un estado y el siguiente es denominado periodo. Por el patrón que sigue la repetición de su conmutación pertenece al tipo de señales periódicas (que repiten un patrón de oscilación en el tiempo).

©Editorial Digital Tecnológico de Monterrey

Autómata (máquina de estados)

Es un circuito con un comportamiento automático. Un circuito de este tipo realiza una serie de pasos repetitivos, siempre respondiendo de la misma manera ante una secuencia de estímulos en las entradas.

B

Bit de paridad impar

Es 1 cuando la cantidad de 1 en un número es impar, de otra forma es 0.

Bit de paridad par

Es 1 cuando la cantidad de 1 en un número es par, de otra forma es 0.

Bus (vector) de datos

Conjunto de señales que tienen una relación entre sí. Todas forman parte de un todo, por lo cual reciben el mismo nombre y se distinguen por un índice.

C

Circuitos aritméticos

Circuitos lógicos que reciben como entrada buses de datos que representan números binarios y cuya salida es el resultado en binario de alguna operación aritmética o de comparación. Generalmente la salida se encuentra en el mismo formato que los números de entrada.

Circuito combinacional

Circuito lógico cuya salida depende únicamente de las entradas y sus relaciones.

©Editorial Digital Tecnológico de Monterrey

Circuito en cascada

Circuito lógico formado por módulos. Cada módulo produce una fracción de la salida directa del circuito y salidas que se conectan como entrada a un siguiente módulo.

Circuito lógico

Circuito en el que las entradas son señales lógicas que se relacionan a través de compuertas lógicas, produciendo salidas que a su vez son señales lógicas.

Complementos a 2 de un número

Diferencia entre 2n y el número, donde el número tiene n *bits*.

Compuertas lógicas

Circuitos básicos formados por transistores que realizan operaciones lógicas básicas o compuestas: and, or, not, xor, xnor, nand, nor.

Concatenación

Operador de VHDL que sirve para unir bajo un nombre a un conjunto de vectores o porciones de vectores. No tiene una compuerta equivalente, ya que no constituye a una operación lógica.

Conexión maestro-esclavo

Circuito en el que las señales de control de un circuito esclavo son proporcionadas por el circuito maestro.

Contador

Registro cuyas salidas cambian de una manera periódica siguiendo un orden predeterminado.

©Editorial Digital Tecnológico de Monterrey

D

Dato serial asincrónico

Bit que ingresa a un componente secuencial sin la sincronía de una señal de reloj.

Dato serial sincrónico

Bit que ingresa a un componente secuencial con la sincronía de una señal de reloj común al circuito al que ingresa.

Decodificador

Circuito con m entradas de control y 2m salidas. La salida que corresponda a la combinación que se encuentre presente en las entradas de control es uno, el resto es cero.

Diagramas de estados

Resume el comportamiento automático de un circuito secuencial.

Dígito BCD (Binary Coded Decimal)

Es un dígito de 0 a 9 expresado en binario.

Diseño funcional

En VHDL es posible expresar el diseño de un circuito utilizando los operadores aritméticos del álgebra: +, -, *. También es posible definir funcionalmente una operación de comparación por medio de sus operadores: mayor, mayor o igual, etc. La expresión funcional representa al circuito combinacional que produce la salida que corresponda. Los FPGAs cuentan con los módulos aritméticos requeridos para configurar un circuito aritmético.

División

Esta operación es secuencial, requiere una señal de reloj para ser resuelta.

©Editorial Digital Tecnológico de Monterrey

E

Ecuación booleana

Ver función booleana.

Encoder

Circuito que cuenta con 2n entradas, detecta la entrada que se encuentre en 1 (una entrada correcta solo tiene un 1 y el resto de las entradas es 0) y la salida es la conversión a binario de la posición de la entrada que está en 1.

Estados

Se construyen con *flip flops* independientes o bien, usando un mayor nivel de integración, con registros.

F

Flip flop

Unidad básica de memoria sincrónica, circuito que almacena un *bit* en la transición de una señal que tiene altas y bajas.

Formato de complementos a 1

Para números positivos, el formato tiene la misma que la representación en signo magnitud, el signo es 0 y la magnitud se deja igual, así que:

Si X>=0: 0&|X| ajustado a un total de n *bits*.

Para números negativos, la regla es:

Si X<0: 2n-1-|X|

Siguiendo esta regla, el signo siempre queda como 1 y la magnitud queda "negada", es decir, los ceros cambian por unos y los unos por ceros.

©Editorial Digital Tecnológico de Monterrey

Formato de complementos a 2

Si X>=0: 0&|X| ajustado a un total de n *bits*.

Si X<0: 2n-|X| =1&|X|* ajustado a un total de n *bits*, donde |X|* es el complemento a 2 de la magnitud del número X. Se consigue este complemento recorriendo el número de derecha a izquierda, después de ocurrir el primer uno (el cual permanece igual), los siguientes *bits* se invierten, ceros por unos y unos por ceros.

Formato de signo magnitud

Este es un formato muy simple, consiste en representar un número en n *bits*. El signo se representa en la posición n-1 y la magnitud en los restantes n-1 *bits*, de la posición 0 a la posición n-2.

FPGA Field Programmable Gate Array

Tipo de PLD (Programmable Logic Device) basado en la interconexión de células llamadas CLB. En un FPGA se configura un circuito lógico. Los principales fabricantes de tarjetas de desarrollo para FPGAs que cuentan tanto de un FPGA como de periféricos e interfaces para entrada y salida son:

Altera – FPGA Manufacturer (Cyclone, Arria, Statix)

Lattice – FPGA Manufacturer (XP, Mach, SC, EC, ECP)

Xilinx – FPGA Manufacturer (Spartan & Virtex)

Avnet (US) – A wide range of Spartan 3, Virtex 2, Virtex 2 Pro, Virtex 4 and Virtex 5 boards

Braemac (Australia) – Australian Altera/Terasic Distributor

BurchED (Australia) – Xilinx FPGA Video Guides

Digilent Inc (US) – Tarjetas de Desarrollo de Xilinix

©Editorial Digital Tecnológico de Monterrey

KNJN LLC (US) – Tarjetas de Desarrollo diversas

Trenz Electronics (Germany) – Tarjetas de Desarrollo de Xilinix

Xess (US) – Tarjetas de Desarrollo de Xilinix

Frecuencia

Representa la cantidad de ciclos que ocurren por unidad de tiempo. La frecuencia de una señal periódica se mide en hertz.

1 Herz = (un ciclo) /segundo

Función booleana

Es la relación de variables booleanas (señales) por medio de operadores lógicos.

I

IEEE Institute of Electrical and Electronics Institute

Asociación con base en E.U.A. y capítulos en todo el mundo. Produce revistas especializadas, organiza congresos y es reponsable de fijar estándares.

L

Latch

Unidad básica de memoria, circuito que almacena un *bit*.

M

Máquinas de estados

Se conocen por los apellidos de sus autores: máquina de Mealy y máquina de Moore, la diferencia depende de las entradas de las funciones que emiten las salidas.

©Editorial Digital Tecnológico de Monterrey

Máquina de Mealy

Las salidas Z se generan por un circuito combinacional cuyas entradas son Si y Xij.

Máquina de Moore

Las salidas solo dependen del estado, así que se generan a través de un circuito combinacional cuya entrada solo es Si.

Medio sumador o Half Adder (HA)

Circuito que recibe dos *bits* como entrada y produce un *bit* de su suma aritmética en binario y un acarreo, emulando la suma que ocurre al sumar dos guarismos en un sistema posicional. Constituye el módulo que suma los *bits* menos significativos dentro de un circuito sumador en cascada.

Memoria

Dispositivo para almacenar datos. Se acceda a través de una dirección.

Monoestable

Es un circuito multivibrador cuya función consiste en que, al recibir una excitación exterior, cambia de estado y se mantiene en este durante un periodo que viene determinado por una constante de tiempo. Transcurrido dicho periodo, la salida del monoestable vuelve a su estado original, por tanto, tiene un estado estable (de aquí su nombre) y un estado casi estable.

Multiplexer

Ver definición de Selector.

Multiplicación

Es una operación que puede ser resuelta en forma secuencial o combinacional, la óptima es esta última. Los FPGAs cuentan con los recursos para expandir esta operación en forma combinacional.

©Editorial Digital Tecnológico de Monterrey

N

Nivel comportamiento o funcional

En este nivel se describe un circuito a partir de su funcionamiento, en VHDL existen algunas estructuras básicas que lo permiten; además, facilitan el uso de condiciones que describen la respuesta del circuito a sus entradas.

Niveles de descripción

Existen tres diferentes niveles de descripción de la arquitectura de un circuito en VHDL: nivel estructural, nivel ecuaciones y nivel comportamiento o funcional.

Niveles de un circuito

Es el número más largo de compuertas interconectadas, siendo la salida de una, una entrada a la siguiente, en un circuito lógico.

Nivel ecuaciones

Consiste en describir un circuito a partir de las expresiones booleanas que lo conforman.

Nivel estructural

Consiste en describir un circuito a partir de las conexiones de entrada y salida de sus compuertas o bien de las conexiones de sus componentes básicos. Es la descripción más parecida a la conexión que se realizaría si se cableara el circuito.

Número BCD

Es un número compuesto por dígitos BCD. Aunque está expresado dígito a dígito en binario, es un número en base 10.

©Editorial Digital Tecnológico de Monterrey

Números enteros

Los números enteros pueden tener signo o no. Si no tienen signo entonces se expresan solo en binario con su magnitud con alguna longitud determinada. Si tienen signo, el número se representa siguiendo las reglas de algún formato.

Números enteros con signo

Número entero al que se le dedica un *bit* para indicar su signo. Hay diversos formatos para representarlos: signo magnitud, complementos a 1, complementos a 2.

O

One shot

Circuito monoestable que genera un estado que perdura un tiempo determinado. El que se usa en el capítulo, perdura un ciclo de la señal de reloj que le ingrese.

Operadores lógicos

Son los operadores de la lógica: and, or, not. Existen algunos operadores compuestos: nand, nor, xor.

P

Periodo

Equivale a la siguiente relación:

Periodo = 1/frecuencia que tiene como unidades: segundos/ciclo

Pixel

Unidad mínima a colorear en un monitor. El haz del monitor se encarga de colorear pixeles recorriendo la pantalla horizontalmente, coloreando todos los pixeles de cada línea por cada

©Editorial Digital Tecnológico de Monterrey

pulso de reloj. Cuando una línea ha sido complemente coloreada (hsync=800), una señal (hsync) le indica que debe pasar al inicio de la siguiente línea horizontal, aumentando así con un uno la segunda señal (vsync++) y regresando la primera señal a su valor inicial (hsync = 0). Asimismo, cuando se ha terminado de colorear la última línea (donde vsync = 521), la segunda señal (vsync) regresa a su valor inicial (vsync = 0). Por lo tanto, con la ayuda de estas dos señales (contadores), hsync y vsync, se podrán manejar de una manera efectiva las salidas HS y VS.

Process

Estructura de control en VHDL que permite definir un circuito en nivel funcional a través de estructuras condicionales.

R

RAM (*Random Access Memory*)

Memoria de lectura y escritura.

Ratón

Interfaz que se comunica a través de comunicación serial bidireccional, es decir tanto el FPGA como el ratón transmiten datos, donde el FPGA inicia mandándole un paquete con un *byte* de comando para que el ratón se active, de lo contrario el ratón no mandará información después de ser encendido.

Rebote

Ruido transitorio que produce un botón al hacer contacto. Generalmente el rebote tiene una frecuencia menor a 50 MHz. La señal de reloj que se utilice para conectarse al *one shot* debe tener una frecuencia menor que el ruido, porque de otro modo el rebote deja de ser ruido y se convierte en una señal periódica.

©Editorial Digital Tecnológico de Monterrey

RGB

El VGA tiene tres salidas de color para cada pixel contenido en su resolución de 640 } 480 pixeles. A cada pixel se le aplicará la combinación de estos tres colores (RGB) y dependiendo de la intensidad de cada color se desplegará un color final en ese pixel. Por ejemplo, si el color azul está al máximo mientras los otros dos están apagados (RGB<= “001”), el color desplegado en la pantalla será un azul fuerte.

Registro

Un registro es un componente electrónico que permite almacenar un conjunto de *bits*, o bien, presentar en su salida una secuencia determinada, como es el caso de los contadores o de los registros internos de las máquinas de estados.

Registro de corrimiento (*Shift register*)

Registro cuyas salidas cambian de una manera periódica, recorriéndose a la derecha o a la izquierda de acuerdo a una señal de control.

Reloj

Señal que se utiliza para sincronizar circuitos que deben funcionar ya sea en la transición positiva o negativa de esta señal.

Retraso

En VHDL es posible definir el tiempo en que la salida de una compuerta lógica o un circuito produce una salida válida. Esto solo es posible en la simulación de un componente.

Retraso de un circuito

El retraso de un circuito depende de su número de niveles. El retraso es igual a la suma del retraso de cada uno de sus niveles.

©Editorial Digital Tecnológico de Monterrey

Restador serial

Circuito que calcula una resta procesando la operación en forma secuencial por posición, solo utilizando un restador completo.

ROM (*Read Only Memory*)

Memoria solo de lectura.

Rotación

Función que provoca que un *bit* que se desplaza por un extremo de un registro ingrese por el otro.

Ruido eléctrico

Es una señal astable de alta frecuencia producida al hacer contacto un circuito electromecánico, entre otras causas.

S

Scan code

Es el código de 8 *bits* asociado a cada letra, es decir, si el usuario presiona la letra L, el *scan code* que se genera es 4B (hexadecimal).

Selector

Circuito con m entradas de control, 2m entradas, y una salida. La salida es igual a lo que se encuentre en la entrada que corresponda a la combinación que se encuentre presente en las entradas de control.

Segundero BCD

Circuito que hace un conteo de segundos generando la salida con dígitos BCD.

©Editorial Digital Tecnológico de Monterrey

Señal de control asincrónica

Es una entrada que se utiliza para que, al ser el registro, realice la función que corresponda.

Señal de control sincrónica

Es una entrada que se utiliza para que, al ser detectada junto a una transición positiva o negativa (según sea el caso) de la señal de reloj, el registro realice la función que corresponda.

Señal lógica

Representa a una variable booleana. Tiene dos medidas de voltaje: el cero lógico tiene un valor de alrededor de 0 volts, generalmente se interpreta como cero hasta 1.5 volts y el 1 lógico que es una medida alrededor de 5 volts, que generalmente varía de 3.5 a 6 volts.

Señal de reloj

Es una señal astable que fluctúa y tiene dos estados 0 y 1 lógico (con los niveles de voltaje y corriente que soporte el circuito al que se va a conectar).

Señal de un botón

La señal que proviene de botones genera un 1 lógico mientras el botón se encuentre oprimido y, generalmente, tanto el cambio de 0 a 1 como de 1 a 0, tiene transitorios que son cambios de alta frecuencia de 1 a 0 y a 1 por un lapso corto hasta que finalmente la señal se estabiliza.

Sistema numérico binario

Es el sistema numérico cuya base es 2. Es un sistema posicional. En este sistema los guarismos se llaman *bits* (*binary digit*).

SRAM (Static RAM)

Memoria electrónica volátil organizada en bloques de tamaño fijo.

©Editorial Digital Tecnológico de Monterrey

Sumador BCD

Circuito cuya entrada son dos números BCD y su salida es un número BCD.

Sumador completo o Full Adder (FA)

Circuito que recibe tres *bits* como entrada, dos *bits* de la misma posición que corresponden a dos números binarios y un acarreo y produce un *bit* de su suma aritmética en binario y un acarreo, emulando la suma que ocurre al sumar dos guarismos en un sistema posicional. Constituye el módulo que suma los *bits* menos significativos dentro de un circuito sumador en cascada.

T

Teclado PS/2

Cuenta con un circuito que genera un código en forma serial por cada tecla oprimida.

Timer 555

Circuito que produce una señal periódica a la que es posible regular su período.

Tipos de números

Enteros, BCD, números de punto fijo, números de punto flotante.

U

Unidad de control

Máquina de estados que produce señales de control que habilitan funciones de registros u otros componentes secuenciales.

©Editorial Digital Tecnológico de Monterrey

V

Variables booleanas

Con una variable booleana se modela un evento, fenómeno o dispositivo que solo tiene dos estados. En un circuito lógico, una señal representa a una variable booleana.

Vector de datos

Ver definición de Bus (vector) de datos.

Verilog (*Verify logic*)

Fue creado en 1985 en Automated Integrated Design Systems. Cadence, la cual tiene ahora todos los derechos sobre los simuladores lógicos de Verilog. Es un lenguaje descriptor de *hardware* con una sintaxis similar a C.

VGA (*Video Graphics Array*)

Es la interfaz basada en cinco salidas: Red(R), Green (G), Blue (B), que en conjunto se denominan RGB, Horizontal Syncronization (HS) y Vertical sync (VS).

VHDL

Acrónimo de VHSIC *Hardware* Description Language, VHSIC = Very High Speed Integrated Circuit. Es un lenguaje descriptor de *hardware* con una sintaxis similar a ADA.

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

©Editorial Digital Tecnológico de Monterrey

Bibliografía

Ashenden, P. J. (1998). *The Student's Guide to VHDL*. San Francisco, E.U.A.: Morgan Kaufmann Publishers, Inc.

Bergeron, J. (2000). *Writing Testbenches: Functional Verification of HDL Models*, E.U.A.: Springer.

Bhasker, J. (1999). *VHDL Primer.* (3a Edición) Prentice Hall, Upper Saddle River E.U.A.

Coelho, D. (1989). *The VHDL Handbook*. Estados Unidos: Kluwer Academic Publishers.

Dewey, A. (1997). *Analysis and Design of Digital Systems*. E.U.A.: PWS Publishing Company, New York.

Doulos. (2012). *A Brief History of VHDL*. Obtenido de http://www.doulos.com/knowhow/vhdl_designers_guide/a_brief_history_of_vhdl/

Gajski D. y Khun R. (Diciembre 1983). *Introduction: New VLSI Tools*. IEEE Computer, Vol. 16, No. 12, pp. 11–14.

IEEE (1989). *IEEE Standard VHDL Language Reference Manual*. IEEE Std 1076–1987. Institute of Electrical and Electronics Engineers.

IEEE (1994). *IEEE Standard VHDL Language Reference Manual.* Institute of Electrical and Electronics Engineers.

Lipsett, R., Schaefer, C., Ussery, C. (1989). *VHDL: Hardware Description and Design*. Springer

©Editorial Digital Tecnológico de Monterrey

Mano, M., Kime, C. (2001). *Logic and Computer Design Fundamentals*. (2º ed.). Prentice Hall, Upper Saddle River: Estados Unidos.

Navabi, Z. (1992). *VHDL: Analysis and Modeling of Digital Systems*. Estados Unidos: Mc Graw Hill.

Pardo, F., Boluda, J. A. (2000). *VHDL Lenguaje para síntesis y modelado de circuitos*. Ed. Alfaomega: España.

Pellerin, D., Taylor, D. (1997). *VHDL Made Easy!* Prentice Hall, Upper Saddle River.

Perry, D. L. (1993) *VHDL* (2a. ed.). E.U.A.: McGraw Hill.

Qualis Design Corporation (2000). *VHDL quick reference card*. Obtenido en: http://www.eda.org/rassp/vhdl/guidelines/vhdlqrc.pdf.

Rico, R., Marcos, S. (1998). *Simulación de Arquitecturas de Computadores*. México: Servicio de Publicaciones UAH.

Roth, C. H. (1998). *Digital System Design using VHDL*. PWS Publishing Company, New York.

Sandstrom, J. (Octubre, 1995). *Comparing Verilog to VHDL Syntactically and Semantically. Integrated System Design (EE Times)*. Obtenido en: http://www.sandstrom.org/systemde.htm.

Sandstrom, J. (2000). *Qualis Design Corporation*. Obtenido en: VHDL quick reference card, http://www.eda.org/rassp/vhdl/guidelines/vhdlqrc.pdf.

S/A. (2000) *Qualis Design Corporation*. Obtenido en: VHDL quick reference card, http://www.eda.org/rassp/vhdl/guidelines/vhdlqrc.pdf.

©Editorial Digital Tecnológico de Monterrey

Terés, T. et al. (1997). *VHDL: Lenguaje estándar de diseño electrónico.* Mc Graw Hill; Estados Unidos.

Toole, B. (28 de marzo 2011). *Ada Byron, Lady Lovelace. Biographies of Women Mathematicians*. Obtenido de: http://www.agnesscott.edu/lriddle/women/love.htm.

VHDL Reference Guide, Xilinx, Inc. (1999). Foundation Series. Obtenido de: http://toolbox.xilinx.com/docsan/

Xilinx, Inc. (1999). *VHDL Reference Guide, Xilinx, Inc*. Obtenido en línea: http://toolbox.xilinx.com/docsan/ (seleccionar Foundation Series).

Xilinx, Inc, (2012). *Silicon Devices*. Obtenido de: http://www.xilinx.com/

Yalamanchili, S. (2001). *Introductory VHDL: From Simulation to Synthesis*. E.U.A.: Prentice-Hall.

Yalamanchili, S. (2005). *VHDL Starter's Guide.* (2º ed.). E.U.A.: Prentice-Hall.

©Editorial Digital Tecnológico de Monterrey

Créditos

Sistemas digitales a través de diseños esquemáticos y VHDL / Norma Frida Roffe Samaniego

776 p.

1. VHDL (Lenguaje de descripción para hardware)
2. Diseño de sistemas—Procesamiento de datos

LC: TK7885.7 Dewey: 621.392”

Editorial Digital del Tecnológico de Monterrey

Gerardo Isaac Campos Flores. Director de Efectividad Institucional del Tecnológico de Monterrey

Alejandra González Barranco. Líder de Editorial Digital

Elizabeth López Corolla. Coordinadora editorial

María Fernanda Vergara Bernal. Correctora de estilo.

Innovación y diseño para la enseñanza y el aprendizaje.

Noemí Villarreal Rodríguez. Coordinación de proyectos institucionales y empresariales

Jesús Alejandro Rocha Gámez. Administración de proyecto

María Isabel Zendejas Morales. Diseño editorial

©Editorial Digital Tecnológico de Monterrey

Aviso legal

D.R.© Instituto Tecnológico y de Estudios Superiores de Monterrey, México. 2021.
Ave. Eugenio Garza Sada 2501 Sur Col. Tecnológico C.P. 64849 | Monterrey, Nuevo León | México.
Sistemas digitales a través de diseños esquemáticos y VHDL.
Se prohíbe la reproducción total o parcial de esta obra por cualquier medio sin previo y expreso consentimiento por escrito del Instituto Tecnológico y de Estudios Superiores de Monterrey.
ISBN Obra Independiente: 978-607-501-677-1
Primera edición: agosto 2021

©Editorial Digital Tecnológico de Monterrey

Amazon Media EU S.à.r.l.
Luxemburgo, Luxemburgo
30 de agosto de 2021
100 ejemplares

©Editorial Digital Tecnológico de Monterrey