PRACTICA: 1

CONTROL DE SEÑALES DE TRAFICO

Para recordar el diseño de circuitos secuenciales utilizaremos el diseño de un controlador de señales de tráfico, muy simple, para un cruce de dos calles.

Descripción textual

Un semáforo tienen dos luces: roja y verde. El semáforo de la calle principal está normalmente en verde, mientras que el semáforo de la calle secundaria está en rojo. Este es el funcionamiento por defecto (Figura 1). Cuando se detecta un coche en la calle secundaria se cambian las señales de tráfico. El semáforo de la calle secundaria cambia a verde y el semáforo de la calle primaria cambia a rojo. Cuando se produce este cambio se activa una señal en un visor y para volver al estado por defecto un operador debe activar la señal TIEMPO.

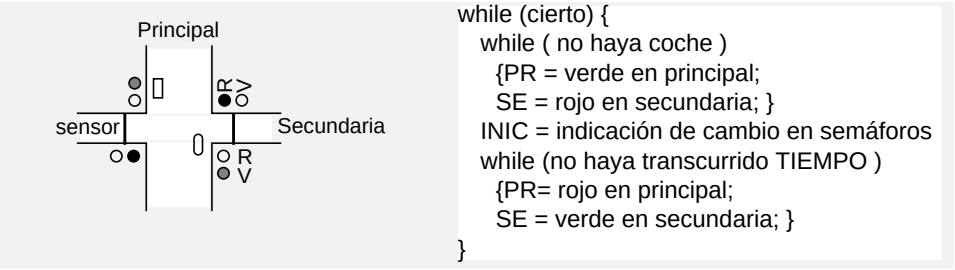


Figura 1 Cruce de calles con semáforos y pseudo código del controlador.

Diagrama de estados y transiciones

La funcionalidad del sistema se puede describir mediante un diagrama de estados y transiciones, el cual se muestra en la Figura 2. Los estados se identifican mediante los acrónimos PR y SE. Una transición se etiqueta de la siguiente forma: evento / acción(salida). Dentro de un nodo se muestra el valor de las señales de salida cuando dependen exclusivamente del estado. Los acrónimos CP y CS indican respectivamente calle primaria y calle secundaria. Los acrónimos V y R indican respectivamente verde y rojo. En el estado PR la calle principal tiene el semáforo en verde. Cuando llega un coche se produce la transición al estado SE. Además se activa la señal INIC y la señal TIEMPO se desactiva. En el estado SE se cambian las luces de los semáforos. El autómata permanece en el estado SE hasta que el operador activa la señal TIEMPO. Entonces, se produce la transición al estado PR.

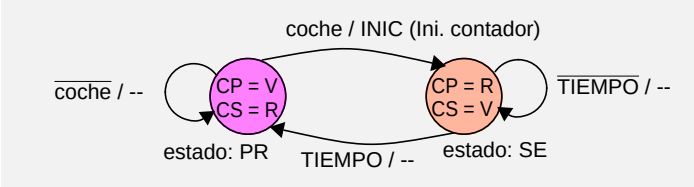


Figura 2 Diagrama de transiciones entre estados de un controlador de semáforos.

Notemos que la salida INIC se corresponde con un autómata de Mealy y las salidas CP y CS se corresponden con un autómata de Moore.

Diseño del autómata

Las transiciones entre estados también se pueden describir mediante una tabla de estados y salidas. Las entradas coche y TIEMPO, la salida INIC y las salidas CP y CS se representan mediante una variable booleana (1 bit, Figura 3).

En las filas se identifica el estado. La variable PR indica el funcionamiento por defecto. En el primer grupo de columnas se identifican las entradas coche y TIEMPO. En la segunda fila de este grupo de columnas se muestra la codificacion de las señales coche y TIEMPO (coche TIEMPO). En una casilla se indica el próximo estado y si se activa la señal INIC (p_estado INIC). En el siguiente grupo de columnas se indican las señales de salida para controlar los semaforos. Las señales CP y CS indican el control de los semáforos en la calle principal y en la calle secundaria respectivamente. Cuando una de estas señales es igual a uno el semáforo correspondiente está en verde. Por tanto, cada una de estas señales se utiliza para activar la luz verde y a la vez para desactivar la luz roja en el mismo semáforo. Mientras se está en el estado PE no se tiene en cuenta el valor de la señal TIEMPO. Por defecto está activada, valor uno, y cuando se activa la señal INIC se desactiva, valor cero. El operador, cuando estima oportuno, vuelve a acivar la señal TIEMPO para indicar que ha finalizado el periodo de semáforo en verde en la calle secundaria. Mientras se está en el estado SE, el operador desactivada la señal INIC y la señal coche no se tiene en cuenta.

casilla (p_estado INIC)		coche / TIEMPO 0 0	01	10	11	salidas CP	CS
estado	PE	PR, 0	PR, 0	SE, 1	SE, 1	1	0
	SE	SE, 0	PR, 0	SE, 0	PR, 0	0	1

Figura 3 Tabla de estados, transiciones y salidas.

Los nombres de los estados los representaremos también mediante una variable booleana, ya que sólo tenemos que representar dos estados. La variable booleana que representa el estado la denominamos EST y la codificación que utilizaremos es EST = 0 para representar el estado PE y EST = 1 para representar el estado SE. En la siguiente tabla se muestra la codificación.

casilla (p_estado INIC)		coche / TIEMPO 0 0	01	10	11	salidas CP	CS
EST	0	0, 0	0, 0	1, 1	1, 1	1	0
	1	1, 0	0, 0	1, 0	0, 0	0	1

En la Figura 4 se muestra un diagrama temporal del funcionamiento de los semáforos en el cruce de calles.

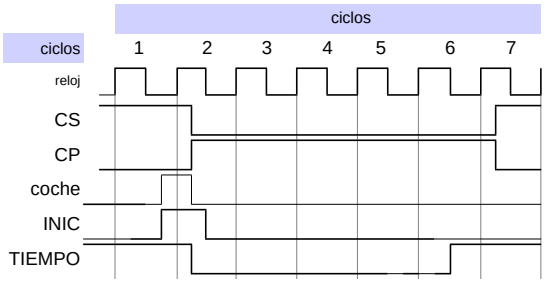


Figura 4 Diagrama temporal donde se observa la evolución de las señales.

De la tabla anterior se pueden extraer las tablas para diseñar el circuito. En una tabla se identifica el cambio de estado (P_EST), en otra tabla se indica la activación de la señal INIC y en la tercera tabla se identifican las señales de salida que dependen sólo del estado. Además estas tablas se ordenan como tablas de Karnaugh para ayudar a la simplificación de las expresiones boolenas.

P_EST		coche / TIEMPO 0 0	01	11	10
EST	0	0	0	1	1
	1	1	0	0	1

INIC		coche / TIEMPO 0 0	01	11	10
EST	0	0	0	1	1
	1	0	0	0	0

		sal CP
EST	0	1
	1	0

		sal CS
EST	0	0
	1	1

Simplificando obtenemos las siguientes expresiones lógicas.

$$P\_EST = \overline{EST} \bullet coche + EST \bullet \overline{TIEMPO}$$

$$INIC = \overline{\overline{EST}} \bullet coche$$

$$CP = \overline{\overline{EST}} \qquad CS = EST$$

En la Figura 5 se muestra el diagrama del autómata utilizando puertas y registros.

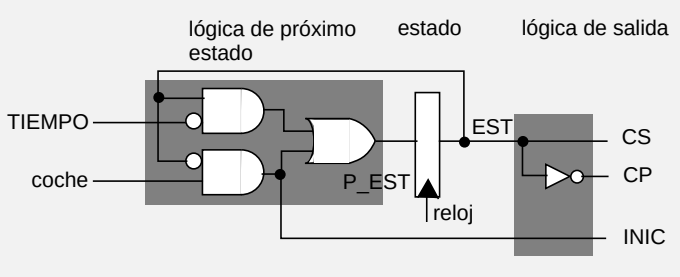


Figura 5 Automata para el control de semáforos.

Las luces de un semaforo se controlan mediante el circuito combinatorio que se muestra en la Figura 6.

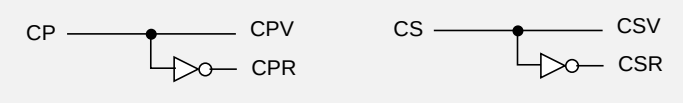


Figura 6 Control de las luces de los semáforos. La última letra de un acronimo indica verde (V) o rojo (R).

Especificación en VHDL

La interface del circuito en VHDL se declara de la siguiente forma.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MECtra is
 port (  coche: in     std_logic ;
         TIEMPO: in  std_logic;
         INIC: out     std_logic;
         CP, CS: out  std_logic);
end MECtra;
```

Para describir una máquina de estados en VHDL utilizaremos tres procesos (en el apéndice 3 se muestra un esqueleto de los procesos. Un proceso modela los registros de estado, otro proceso modela la lógica de próximo estado y el tercer proceso modela la lógica de salida.

Es útil dibujar un diagrama del sistema con cada proceso representado por una caja y explicitar todas las entradas y salidas de cada proceso (Figura 7). Si una señal es una salida en dos cajas, o si una señal no es una entrada a un proceso, algo no está correcto.

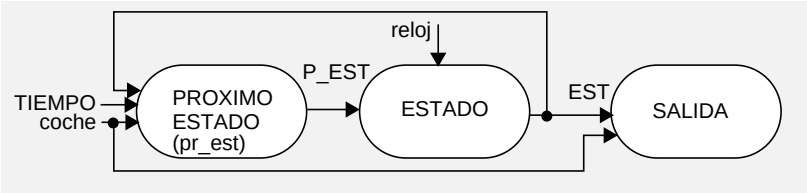


Figura 7 Procesos utilizados para modelar el autómata.

En la declaración de la arquitectura se declara como objeto señal EST, la cual modela el registro (salida) que almacena el estado. Adicionalmente se declara como objeto señal P_EST que modela la entrada del registro (próximo estado). Estos dos objetos se utilizarán para efectuar la comunicación entre procesos.

```
architecture comportamiento of MECtra is
type tipoestado is (PR, SE);
signal EST, P_EST: tipoestado;          -- especificación del registro
. . .                                   -- (salida) y de su entrada
end comportamiento;
```

En la lista de activación de los procesos se especifican las señales que determinan su ejecución.

```
estado: process (reloj) begin
pr_est: process (EST, coche, TIEMPO) begin
salida: process (EST, coche) begin
```

El proceso estado se activa en ambas transiciones de la señal de reloj. Por tanto, utilizaremos la función flanco ascendente para efectuar la distinción dentro del cuerpo del proceso.

```
function flanco_ascendente (signal reloj : std_logic)
        return boolean is
begin
        return (reloj = '1' and reloj'event);
end flanco_ascendente;
```

Cuando se detecta un flanco ascendente se asigna el valor de la señal P_EST a la señal EST en el proceso estado.

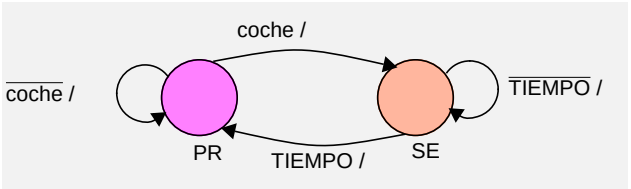
Para establecer un estado inicial al conectar el controlador de semáforos utilizaremos una señal de puesta a cero asíncrona (pcero). Esta señal hay que añadirla en la declaración previa de la interface (pcero:**in std_logic**;).

El proceso que modela el registro de estado es el siguiente.

```
estado: process (reloj, pcero) begin
        if pcero = '1' then EST <= PR;
        elsif flanco_ascendente (reloj) then
            EST <= P_EST;
        end if;
    end process;
```

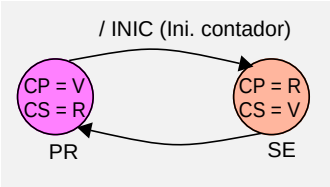
El proceso de la lógica de próximo estado es el siguiente.

```
pr_est: process (EST, coche, TIEMPO) begin
        P_EST <= EST;
        case EST is
            when PR => if coche = '1' then
                            P_EST <= SE;
                       end if;
            when SE => if TIEMPO = '1' then
                            P_EST <= PR;
                       end if;
            when others => P_EST <= PR;
        end case;
    end process;
```



El proceso de la lógica de salida es el siguiente.

```
salida: process (EST, coche) begin
        INIC <= '0';
        case EST is                          salidas
            when PR => CP <= '1';            autómata de Moore
                       CS <= '0';            autómata de Moore
                       if coche = '1' then
                          INIC <= '1';       autómata de Mealy
                       end if;
            when SE => CP <= '0';            autómata de Moore
                       CS <= '1';            autómata de Moore
            when others => CP<= '1';         autómata de Moore
                       CS <= '0';            autómata de Moore
        end case;
    end process;
```



ENTREGA. *En la entrega de la práctica indique los trozos de código de cada proceso que se ejecutan cuando el autómata está en el estado PR y llega un coche. Suponga que la señal de reloj es estable y tiene el valor 1. Suponga también que la señal pcero es estable y tiene el valor cero.*

El proceso de la lógica de salida también puede especificarse mediante sentencias concurrentes.

```
INIC <= '1' when (EST = PR and coche = '1') else '0';
CP <= '1' when EST = PR) else '0' ;
CS <= '1' when EST = SE) else '0' ;
```

Con la herramienta LogicWorks, construya el autómata utilizando la descripción VHDL. Compruebe el funcionamiento utilizando conmutadores y visores. Las señales pcero, TIEMPO y coche las activará manualmente mediante conmutadores.

ENTREGA. *En la entrega de la práctica muestra un diagrama temporal, extraído de la ventana de tiempos de LogicWorks, describiendo las transiciones que se observan.*

PARTICIONADO EN CAMINO DE DATOS Y CONTROLADOR

En este apartado se elimina la actuacion de un operador en el controlador de tráfico. La actuación manual de un operador se sustituye por un automatismo.

Descripción textual

Cuando un coche se aproxima al cruce por la calle secundaria se activa un sensor y determina que, durante un intervalo fijo de tiempo, el semáforo de la calle principal se ponga rojo y el semáforo de la calle secundaria se ponga verde. Una vez expira el intervalo de tiempo la calle principal vuelve a tener el semáforo en verde y la calle secundaria en rojo.

Diseño del sistema

Para establecer el intervalo fijo de tiempo utilizaremos la señal de reloj. Supondremos que la frecuencia de reloj del sistema es tal que el intervalo fijo de tiempo finaliza al cabo de 4 ciclos.

Podemos modificar el diagrama de estados del autómata y añadir estados para modelar todo el sistema. Esto es, modelar el intervalo de tiempo. Sin embargo, la funcionalidad del controlador de tráfico queda enmascarada por

estos estados adicionales que implementan un simple contador. Es más claro separar la función control de tráfico del contador de tiempo.

Una forma de efectuar la separación es utilizar dos autómatas que se comunican. Otra alternativa es pensar que se dispone de un elemento hardware que es un contador. Este elemento es utilizado frecuentemente en diseños y disponemos de su descripción. Por tanto, podemos describir un sistema secuencial en términos de un camino de datos y el controlador, el cual es un autómata de diseño específico. El camino de datos está construido a partir de elementos que han sido previamente diseñados.

En al Figura 8 se muestra un modelo general de sistema particionado siguiendo la idea descrita.

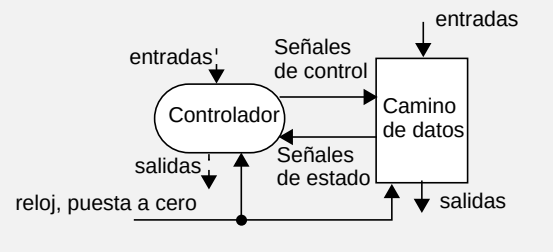


Figura 8 Esquema general de un sistema con camino de datos y controlador específico.

En general un camino de datos manipula y procesa datos. El controlador específico permite y determina el secuenciamiento de las operaciones en el camino de datos. Las señales de estado, provenientes del camino de datos, son utilizadas por el controlador como indicacion de eventos que se producen en el camino de datos. Las señales de control, provenientes del controlador específico, se utilizan en el camino de datos para establecer las operaciones que quieren realizarse.

En el diseño del controlador de tráfico el camino de datos es el contador y el controlador diseñado es el controlador específico (Figura 9). El controlador determina que se inicie la medida de un periodo fijo de tiempo. La señal coche es entrada del controlador. Comparando la Figura 8 y la Figura 9 observamos que la señal TIEMPO es una señal de estado y las señales INC, CP y CS son señales de control.

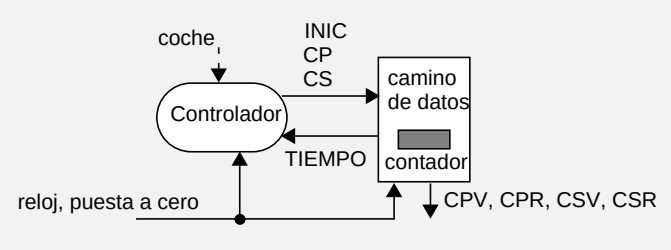


Figura 9 Camino de datos y controlador del autómata para control de semáforos.

Debemos diseñar un contador que empiece a contar cuando se activa la señal INIC (Figura 10). La salida (TIEMPO) se desactivará mientras está contando y se volverá a activar cuando hayan transcurrido los ciclos prefijados. La señal de entrada (INIC) se activara durante un periodo de la señal de reloj. Mientras el contador está contando no se tiene en cuenta ninguna activación de la señal INIC.

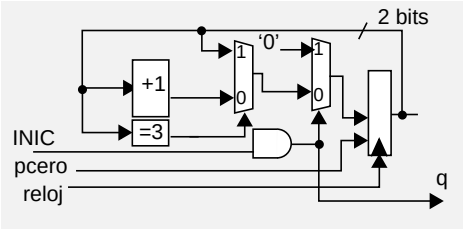


Figura 10 Contador.

En el diagrama temporal de la Figura 11 se muestra la evolución temporal cuando se detecta un coche en la calle secundaria.

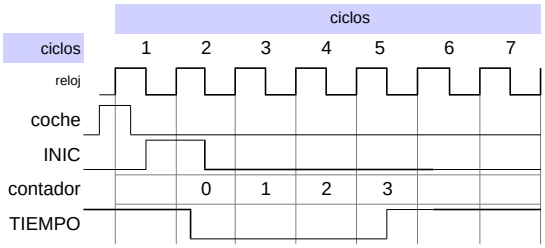


Figura 11 Diagrama temporal.

Utilice la descripción VHDL del contador especificado en el apéndice 4 para crear un módulo contador. Conecte esta módulo al autómata diseñado previamente. Compruebe el funcionamiento del sistema. Las señales pcero y coche las activará manualmente mediante conmutadores.

ENTREGA. *En la entrega de la práctica muestra un diagrama temporal, extraído de la ventana de tiempos de LogicWorks, describiendo las transiciones que se observan.*

ALMACENAMIENTO DE INFORMACIÓN

Una de las deficiencias del controlador de tráfico descrito es que la calle secundaria es prioritaria cuando el sensor detecta un coche. Se cambian los semáforos aunque hayan sido cambiados en el ciclo previo. Un diseño más equitativo es que el semáforo de la calle principal también esté en verde durante un periodo fijo de tiempo, aunque se haya detectado un coche en la calle secundaria.

ENTREGA. *En la entrega de la práctica muestra un diagrama temporal, extraído de la ventana de tiempos de LogicWorks, donde se observe el hecho que se acaba de describir.*

La señal que indica que hay un coche en la calle secundaria es un pulso cuya duración es menor que el intervalo de tiempo del contador. Por tanto, hay que recordar que hay un coche que está esperando. En caso contrario, la indicación de que ha llegado un coche pueder perderse.

Un dato que necesita ser recordado debe almacenarse en un registro. Almacenar un dato de esta forma es lo mismo que almacenar el estado en un autómata. El almacenamiento puede estar incluido en la propia descripción del controlador o disponer de un elemento de memorización en el camino de datos.

Descripción textual

El semáforo de la calle principal está normalmente en verde, mientras que el semáforo de la calle secundaria está en rojo. Este es el funcionamiento por defecto (Figura 1). Cuando se detecta un coche en la calle secundaria se cambian las señales de tráfico. El semáforo de la calle secundaria cambia a verde y el semáforo de la calle primaria cambia a rojo. Cuando se produce este cambio se activa un contador, que después de un periodo fijo de tiempo activa la señal TIEMPO y entonces los semáforos y vuelven al estado por defecto. Al volver al estado por defecto se vuelve a activar el contador. Mientras el contador no indique que ha transcurrido el tiempo prefijado no se cambian los semaforos aúnque se detecte un coche en la calle secuendaria.

Diagrama de estados y transiciones para controlar el camino de datos

En la Figura 12 se muestra el diagrama de transiciones entre estados. Del estado PR se efectúa una transición al estado SE cuando ha transcurrido el intervalo de tiempo prefijado y se detecta un coche en la calle secundaria. Entonces se vuelve a iniciar la medida de un intervalo de tiempo fijo (INIC). La transición del estado SE al estado

PR se produce cuando ha transcurrido el intervalo de tiempo prefijado. Se vuelve ha iniciar una medida de un intervalo de tiempo (INIC. Mientras no haya transcurrido el intervalo de tiempo o no se detecte un coche el sistema permanece en el estado PR.

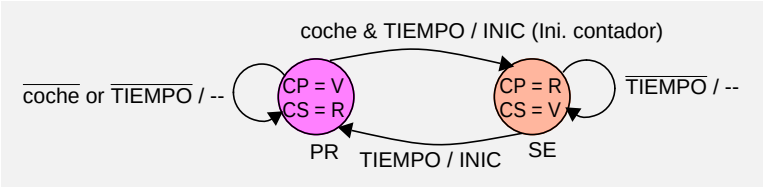


Figura 12 Diagrama de transiciones entre estados.

ENTREGA. *Construya la tabla de transición entre estados y diseñe el esquema con puertas y registros. Entregue las tablas y el esquema de puertas y registros.*

Especificación en VHDL incluyendo el registro

En las siguientes descripciones se muestra mediante una marca a la izquierda del código las sentencias añadidas. Para modelar el registro declaramos el objeto cohesp como señal. Además, declaramos el objeto llecoh también como señal para modelar la entrada del registro. El objeto llecoh se utiliza para indicar que acaba de llegar un coche. EL objeto cohesp indica que está esperando un coche en la calle secundaria .

Definicion de las señales utilizadas para especificar el registro y la entrada.

```
architecture comportamiento of MECTra is
type tipoestado is (PR, SE);
signal EST, P_EST: tipoestado;
signal llecoh, cohesp: std_logic;
. . .
end comportamiento;
```

La señal cohesp (coche esperando) se actualiza al mismo tiempo que la señal EST que representa al estado.

```
estado: process (reloj, pcero) begin
    if pcero = '1' then
        EST <= PR;
        cohesp <= '0';
    elsif flanco_ascendente (reloj) then
        EST <= P_EST;
        cohesp <= llecoh;          -- memoriza que ha llegado un
                                   -- coche en el ciclo previo o
                                   -- esperando
    end if;
end process;
```

Para actualizar el registro que identifica si hay un coche esperando utilizamos un proceso. Este proceso determina si ha llegado un coche, la calle principal tiene el semáforo en verde y no ha transcurrido el íntervalo fijo de tiempo. La señal cohesp está en la lista de activación del proceso. La señal llecoh se activa o desactiva en el siguiente proceso.

```
llegcoh: process (EST, coche, cohesp, TIEMPO) begin
    if (EST = PR and cohesp = '1' and TIEMPO = '1') then  -- ha transcurrido el intervalo de
                                                          -- tiempo y hay un coche esperando
        llecoh <= '0';
    elsif coche = '1' then
        llecoh <= '1';                                    -- indicación de que ha llegado
                                                          -- un coche
    else
        llecoh <= cohesp;
```

```
        end if;
    end process;
```

Para modelar la lógica de próximo estado y la lógica de salida utilizaremos un sólo proceso en lugar de los dos procesos utilizados en el diseño previo. En el apéndice 3 se muestra el esqueleto de los procesos cuando se efectúa el modelado con uno y dos procesos.

El proceso que modela la logica de próximo estado y la lógica de salida es el siguiente.

```
pr_est_sal: process (EST, cohesp, TIEMPO) begin
        INIC <= '0';
        case EST is
            when PR =>
                CP <= '1';
                CS <= '0';
                 if cohesp = '1' and TIEMPO = '1' then
                    INIC <= '1';
                    P_EST <= SE;
                else
                    P_EST <= PR;
                end if;
            when SE =>
                CP <= '0';
                CS <= '1';
                if TIEMPO = '1' then
                    INIC <= '1';
                    P_EST <= PR;
                else
                    P_EST <= SE;
                end if;
            when others => CP <= '1'; CS <= '0'; P_EST <= PR;
        end case;
    end process;
```

Utilice la descripción VHDL para crear un módulo. Conecte el módulo al contador encapsulado previamente. Compruebe el funcionamiento del sistema. Las señales pcero y coche las activara manualmente mediante conmutadores.

ENTREGA. *En la entrega de la práctica muestra un diagrama temporal, extraído de la ventana de tiempos de LogicWorks, describiendo las transiciones que se observan.*

Descripción del sistema utilizando un registro en el camino de datos

En la Figura 13 se muestra la conexión del controlador y el camino de datos. En el camino de datos se identifican el contador (cnt), el registro (reg) y la lógica para generar las señales de control de las luces de los semáforos.

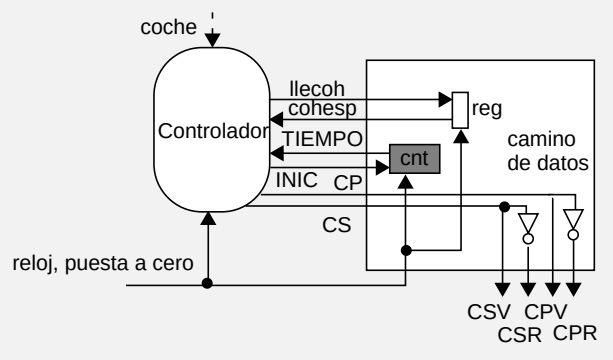


Figura 13 Camino de datos y controlador de un control de semáforos.

Especificación en VHDL del controlador del camino de datos

La interface del circuito en VHDL se declara de la siguiente forma.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MECta is
 port (  coche: in      std_logic ;
         TIEMPO: in  std_logic;
         cohesp: in     std_logic;
         llecoh: out    std_logic;
         INIC: out      std_logic;
         CP, CS: out  std_logic);
end MECtra;
```

En el apéndice 4- se muestra la especificación de un registro e VHDL. Encapsule la especificación para utilizarla posteriormente en el diseño del sistema.

Modifique la especificación VHDL del controlador que incluye un registro para implementar el controlador del camino de datos de la Figura 13. Conecte el controlador con el registro y el contador y compruebe el funcionamiento del sistema. Las señales pcero y coche las activara manualmente mediante conmutadores.

ENTREGA. *En la entrega de la práctica muestra un diagrama temporal, extraido de la ventana de tiempos de LogicWorks, describiendo las transiciones que se observan.*

Apéndice 1: Señales versus variables

Podemos decir que una señal representa un cable o algún tipo de conexión física en un diseño. Una señal toma valores en instantes específicos de tiempo en la simulación. Una variable toma el valor inmediatamente cuando es asignada. Seguidamente se utilizan dos ejemplos para mostrar la diferencia entre la asignación de variables y señales en el cuerpo de un proceso (Figura 14).

En el ejemplo de la izquierda de la Figura 14 el objeto v_a se declara como una variable. Entonces, la sentencia S2 utiliza el valor establecido en v_ar en la sentencia S1.

Ejemplo: variables	Ejemplo: señales
[p_var:] **process** (c, d) **is**	signal v_a: std_logic;
variable v_a: std_logic;	[p_señ] **process** (c, d) **is**
begin	**begin**
S1: v_a := c and b;	S1: s_a <= c and b;
S2: . . . := v_a . . .	S2: . . . := s_a . . .
S3: . . . <= v_a . . .	S3: . . . <= s_a . . .
end process [process_label];	**end process** [process_label];

Figura 14 Ejemplos de asignación de variables y señales.

En el ejemplo de la derecha de la Figura 14 el objeto s_a se declara como una señal. La sentencia S2 no utiliza el valor establecido en la sentencia S1. Utiliza el valor que la sentencia S1 ha establecido en la ejecución previa del proceso. Ello es debido a que una señal toma el valor asignado después de un retardo. Por defecto, este retardo es un valor denominado delta que es mayor que cero. Por tanto, como un proceso se ejecuta en tiempo cero el valor asignado a una señal no es observable en una sentencia que se especifica posteriormente en el cuerpo del proceso y la señal aparece en la parte derecha de una sentencia de asignación de señal. En otras palabras, se ignoran dependencias textuales entre señales. En la Figura 15 se ilustra de forma esquemática el proceso de actualización de señales en un proceso.

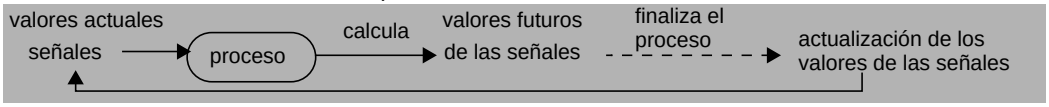


Figura 15 Actualización de señales en un proceso.

Seguidamente se muestran dos especificaciones en VHDL de un sumador de un bit utilizando el constructor proceso. La lista de activación del proceso incluye las señales a, b y cen ya que la lógica combinatoria debe responder a cambios en cualquiera de las señales de entrada. La diferencia entre ambas especificaciones se encuentra en los objetos p y g. En la especificación de la izquierda los objetos son variables mientras que en la especificación de la derecha son señales.

Sumador de 1 bit: variables	Sumador de 1 bit: señales
library ieee;	**library** ieee;
use ieee.std_logic_1164.**all**;	**use** ieee.std_logic_1164.**all**;
entity s1bitV **is**	**entity** s1bitS **is**
port (a, b, cen: **in std_logic**;	**port** (a, b, cen: **in std_logic**;
s, csal: **out std_logic**);	s, csal: **out std_logic**);
end s1bitV;	**end** s1bitS;
architecture beh **of** s1bitV **is**	**architecture** beh **of** s1bitS **is**
begin	**signal** p, g: **std_logic**;
proc_s1bitV: **process** (a, b, cen)	**begin**
variable p, g: **std_logic**;	proc_s1bitS: **process** (a, b, cen)
begin	**begin**
p := a **xor** b**;**	**p** <= a **xor** b**;**
g := a **and** b**;**	**g** <= a **and** b**;**

```
    s <= p xor cen;                    s <= p xor cen;
    csal <= g or (p and cen);          csal <= g or (p and cen);
end process proc_s1bitV;           end process proc_s1bitS;
end beh;                           end beh;
```

En La parte izquierda de la Figura 16 se muestra el sistema que se modela cuando se definen los objetos p y g como variables y en la parte de la derecha cuando se definen como señales. El reloj virtual que controla los registros es el ciclo de simulación (tiempo delta, Figura 15).

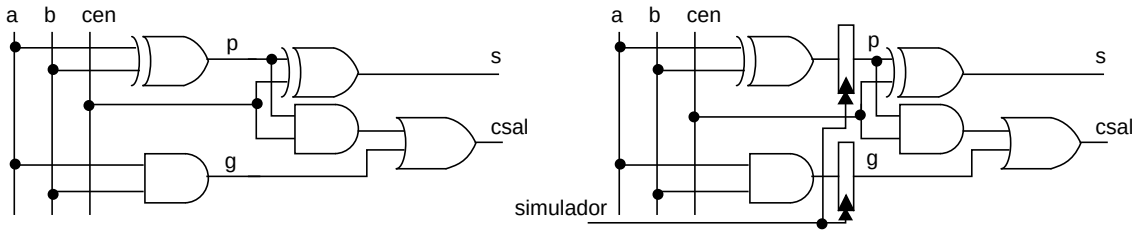


Figura 16 Esquemas de circuito sintetizados de una descripción VHDL.

Seguidamente se muestra el sumador de un bit cuando las salidas están sincronizadas por una señal de reloj. La especificación VHDL es la siguiente, donde se utiliza la llamada a una función denominada flanco_ascendente para determinar el flanco ascendente de la señal de reloj .

Sumador de 1 bit: variables	Sumador de 1 bit: señales
library ieee;	**library** ieee;
use ieee.std_logic_1164.**all**;	**use** ieee.std_logic_1164.**all**;
entity s1bitV **is**	**entity** s1bitS **is**
port (a, b, cen, reloj: **in std_logic**;	**port** (a, b, cen, reloj: **in std_logic**;
s, csal: **out std_logic**);	s, csal: **out std_logic**);
end s1bitV;	**end** s1bitS;
architecture beh **of** s1bitV **is**	**architecture** beh **of** s1bitS **is**
begin	**signal** p, g: **std_logic**;
proc_s1bitV: **process** (a, b, cen, reloj)	**begin**
variable p, g: **std_logic**;	proc_s1bitS: **process** (a, b, cen, reloj)
begin	**begin**
if (flanco_ascendente (reloj) **then**	**if** (flanco_ascendente (reloj) **then**
p := a **xor** b**;**	**p** <= a **xor** b**;**
g := a **and** b**;**	**g** <= a **and** b**;**
s <= p **xor** cen**;**	**s** <= p **xor** cen**;**
csal <= g **or** (p **and** cen)**;**	**csal** <= g **or** (p **and** cen)**;**
end if;	**end if;**
end process proc_s1bitV;	**end process** proc_s1bitS;
end beh;	**end** beh;

En la Figura 16 se muestran los sistemas que modelan las especificaciones previas. En la parte izquierda se muestra el sistema cuando p y g se definen como variables y en la parte derecha se muestra el sistema cuando p y g se definen como señales. Todos los registros actúan en el flanco ascendente de la señal de reloj.

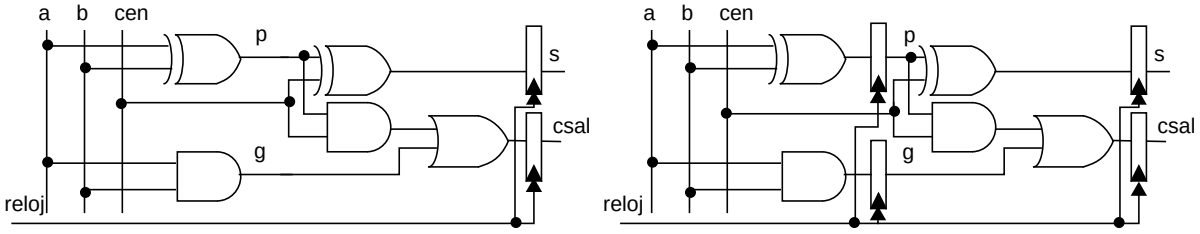


Figura 17 Esquemas de un sumador de un bit síncrono sintetizados de una descripción VHDL.

Apéndice 2: Circuitos secuenciales

Los circuitos digitales complejos tienen incluido el concepto de estado. En otras palabras, las salidas dependen de valores previos de las entradas y de los valores actuales de las entradas. El valor previo de las entradas se almacena implícitamente, de forma codificada, en lo que se denomina estado. Estos sistemas se denominan secuenciales en contraposición a los sistemas combinacionales.

En la Figura 18 se muestra un modelo genérico de un sistema secuencial. El estado del sistema se almacena en registros. La salida de los registros es el estado actual y la entrada de los registros es el estado futuro.

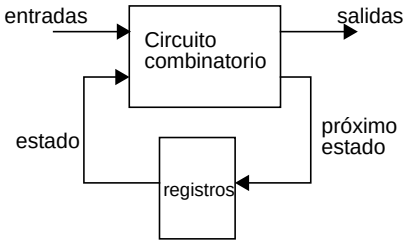


Figura 18 Modelo genérico de un sistema secuencial.

Los circuitos secuenciales se pueden dividir en dos clases básicas: síncronos y asíncronos. Un circuito sÍncrono actualiza el estado cuando cambia una señal de reloj. Un circuito asÍncrono cambia el estado en el mismo instante en que se modifica el estado futuro. Nosotros nos centraremos en los circuitos síncronos. Un circuito síncrono facilita la verificación y la comprobación del funcionamiento.

En un circuito combinatorio se puede producir lo que se denomina riesgo o condición de carrera. Esto es, las salidas puede mostrar valores espurios debido al retardos distintos en los caminos desde las entradas a las salidas. Un circuito secuencial debe diseñarse para que si tales riesgos se producen sean ignorados.

Para asegurar que un circuito secuencial es capaz de ignorar los riesgos que pueden producirse en un circuito combinatorio se utiliza un reloj para sincronizar los datos. Cuando la señal de reloj tiene el valor cero se ignora cualquier riesgo. Este comportamiento se implementa mediante puertas AND. Se efectua la operación AND de cada salida con la señal de reloj. En estas condiciones el sistema aún es susceptible a riesgos cuando la señal de reloj toma el valor uno. Por tanto, es usual utilizar registros que sólo muestrean la señal de entrada en cambios de la señal de reloj. Un cambio en el valor de la señal de reloj (flanco) tiene una duracion mucho menor que un semiperiodo de la señal de reloj. Por tanto, el dato sólo debe ser estable durante la duración del flanco. Por ello, los registros de estado de un circuito secuencial sÍncrono son registros que actúan en un flanco de la señal de reloj.

Circuito síncrono. Un circuito síncrono utiliza los registros como elementos de memorización y todos los registros están controlados por un único reloj.

El diagrama básico de un circuito síncrono se muestra en la Figura 19. El elemento de memorización (registro) conocido como registro de estado es un conjunto de registros, sincronizados por una única señal de reloj. La salida del registro es la señal de estado que representa el estado interno del sistema. La lógica próximo_estado es un circuito combinatorio que determina el próximo estado. La lógica salida es otro circuito combinatorio.

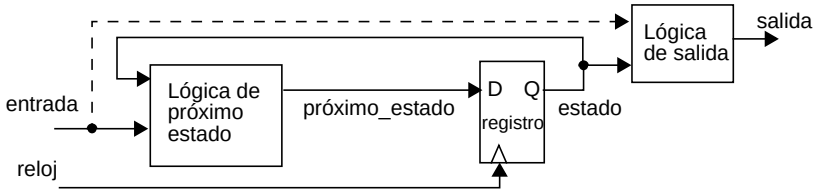


Figura 19 Modelo esquemático de un circuito secuencial síncrono.

En la Figura 19 las señales de salida dependen del estado y de las señales de entrada (línea de trazos). Este tipo de circuito síncrono se denomina autómata de Mealy. Si las señales de salida sólo dependen del estado el circuito síncrono se denomina autómata de Moore. Nosotros nos centraremos en estos últimos.

El funcionamiento del circuito es el siguiente.

- En el flanco ascendente de la señal de reloj el valor de la señal próximo_estado (D) se muestrea y se propaga a la salida (Q), siendo ahora el nuevo estado. El valor se almacena en el elemento de memorización (registro) y no se modifica durante todo el periodo de la señal de reloj. Representa el estado del sistema.
- Las lógicas combinatorias próximo_estado y salida determinan respectivamente el próximo estado y la salida.
- En el siguiente flanco ascendente de la señal de reloj se repite el proceso.

En la Figura 20 se muestra un diagrama temporal de las relaciones de dependencia entre los retardos de los componentes de un circuito secuencial síncrono. La magnitud de los retardos que se representa es un ejemplo.

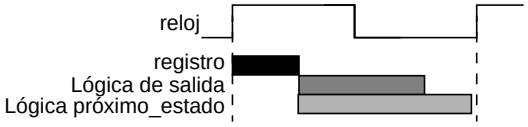


Figura 20 Diagrama temporal en un periodo de la señal de reloj de un circuito síncrono. Relaciones de dependencia entre los retardos.

Dado un cicuito síncrono podemos distinguir, de forma informal, varios tipos: a) circuito secuencial regular, b) circuito secuencial aleatorio y c) circuito secuencial combinado. La diferencia entre el primer tipo y el segundo es la complejidad de las transiciones entre estados y de las lógicas combinacionales. En el primer caso la representación binaria de los estados usualmente tiene una interpretación. Ejemplos del primer tipo son contadores y registros de desplazamiento. Los circuitos del segundo tipo se denominan máquinas de estados finitos. En el tercer tipo se incluyen los circuitos que consistente de elementos del primer tipo y del segundo tipo. En este tercer caso, la máquina de estados finitos se utiliza para controlar el circuito secuencial regular.

Diseños simples

Registro con permiso de escritura síncrono. En la tabla de la Figura 21 se representa el funcionamiento del circuito que se muestra en la parte derecha de la misma figura. El símbolo Q* indica el futuro valor de la salida y el símbolo Q indica el valor actual de la salida. La señal PE (permiso de escritura) sólo se tiene en cuenta en el flanco ascendente de la señal de reloj. Esto significa que la señal está sincronizada con la señal de reloj. En el flanco ascendente de la señal de reloj se muestrean las señales PE y D. Si PE = 0 se mantiene el valor en el elemento de memorización. En caso contrario, cuando PE = 1, el funcionamiento es el de un registro. Esto es, la entrada se propaga a la salida.

reloj	PE	Q*
0	-	Q
1	-	Q
flanco	0	Q
flanco	1	D

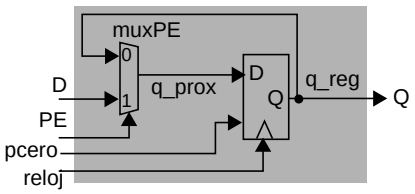


Figura 21 Registro con permiso de escritura síncrono.

En la parte derecha de la Figura 21 se ha mostrado el diagrama de un registro con permiso de escritura. Observemos que podemos asimilar elementos de este diagrama con elementos mostrados en la Figura 19. La lógica del próximo_estado es el multiplexor y no existe lógica para determinar la señal de salida, ya que es directamente la salida del registro.

En la Figura 22 se muestra una descripción VHDL del registro con permiso de escritura síncrono donde se distinguen tres partes. El proceso denominado reg modela al elemento de memorización denominado registro. El elemento mux se modela mediante una sentencia de asignación de señal de forma condicional[1]. La sentencia de asignación de señal modela la lógica de salida. La lógica de salida es un cable que conecta la salida del registro con el puerto de salida.

Registro con permiso de escritura síncrono

```
library IEEE;
use IEEE.std_logic_1164.all;

entity regen is
 port (   reloj, pcero, PE, D: in  std_logic;
          Q: out                   std_logic );
end regen;

architecture trespart of regen is
signal q_reg: std_logic;
signal q_prox: std_logic;
 begin
        --Estado
 reg: Process (reloj, pcero)
    begin
            if (pcero ='1') then
                q_reg <= '0 'after 2 ns;
            elsif (reloj'event and reloj = '1') then
                q_reg <= q_prox after 2 ns;
            end if ;
end process ;
        -- Lógica de próximo estado
                q_prox <= D after 5 ns when PE = '1' else
                              q_reg after 5 ns;
        -- Lógica de salida
                q <= q_reg;
end trespart;
```

Figura 22 Descripción VHDL de un registro con permiso de escritura síncrono distinguiendo los elementos combinatorios y secuenciales.

En la descripción VHDL de la Figura 22 el registro tiene un retardo de actualización de 2 ns y la lógica de próximo estado tiene un retardo de 5 ns.

La descripción VHDL de la Figura 22 sigue el modelo de la Figura 19. En este modelo se pueden identificar de forma clara los elementos combinatorios y los elementos secuenciales. Esto permite comprobar y verificar el funcionamiento de los componentes de forma aislada. En la Figura 22 se muestra una descripción VHDL del mismo tipo de registro con un sólo proceso. Cuando los circuitos son complejos este tipo de descripción, donde se entremezclan los distintos elementos, dificulta la comprobación y verificación.

1.Esta sentencia de asignación de señal de forma condicional puede sustituirse por un proceso.

Registro con permiso de escritura síncrono

```
library IEEE;
use IEEE.std_logic_1164.all;

entity regen is
 port (   reloj, pcero, PE, D: in  std_logic;
          Q: out                   std_logic );
end regen;

architecture beha of regen is
signal q_reg: std_logic;
signal q_prox: std_logic;
 begin
 regen: Process (reloj, pcero)
    begin
            if (pcero ='1') then
                Q <= '0 'after 2 ns;
            elsif (reloj'event and reloj = '1') then
                if PE = '1' then
                   Q <= D after 2 ns;
                end if ;
            end if ;
 end process ;
 end beha;
```

Figura 23 Descripción VHDL de un registro con permiso de escritura utilizando un único proceso.

Contador binario. Un contador binario pasa por una secuencia de estados cuya codificación en binario se puede interpretar como números naturales. Por ejemplo un contador binario de 2 bits repite indefinidamente la secuencia: 00, 01, 10, 11.

Un contador binario tiene un registro que almacena n bits y su salida se interpreta como un número natural codificado en base 2. El contador incrementa el contenido del registro en cada ciclo de la señal de reloj; contando desde 0 hasta 2^n - 1. La cuenta se repite indefinidamente.

En la Figura 24 se muestra un esquema de un contador binario. Comparando esta figura con la Figura 19 podemos identificar que la lógica próximo_estado es un incrementador, el cual calcula el nuevo valor del próximo estado. No existe lógica para determinar la señal de salida, ya que es directamente la salida del registro.

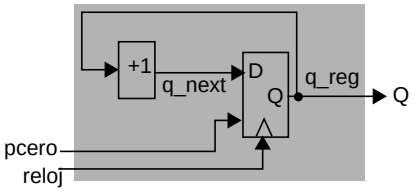


Figura 24 Contador binario.

En la Figura 25 se muestra una descripción VHDL del contador binario donde se distinguen tres partes. El proceso denominado reg modela al elemento de memorización denominado registro. La primera sentencia de asignación de señal después del proceso modela el incrementador unidad. La última sentencia de asignación de señal modela la lógica de salida. La lógica de salida es un cable que conecta la salida del registro con el puerto de salida.

Contador binario

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity contador is
 port (   reloj, pcero : in  std_logic;
          Q: out             std_logic_vector (3 downto 0) );
end contador;

architecture trespart of contador is
signal r_reg: std_logic_vector (3 downto 0);
signal r_prox: std_logic_vector (3 downto 0);
 begin
        --Estado
 reg: Process (reloj, pcero)
    begin
            if (pcero ='1') then
                r_reg <= (others => '0 ') after 2 ns;
            elsif (reloj'event and reloj = '1') then
                r_reg <= r_prox after 2 ns;
            end if ;
 end process ;
        -- Lógica de próximo estado
                r_prox <= r_reg +1 after 8 ns ;
        -- Lógica de salida
                q <= r_reg;
end trespart;
```

Figura 25 Descripción VHDL de un contador binario.

Contador módulo. En la Figura 26 se muestra el esquema de circuito de un contador módulo 7. La lógica próximo estado es un incrementador, un multiplexor y un comparador.

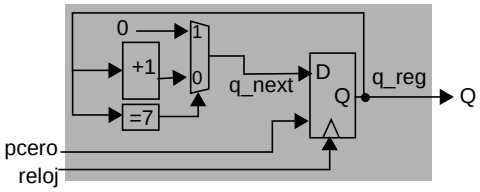


Figura 26 Contador módulo.

En la Figura 27 se muestra la descripción VHDL siguiendo el modelo de la Figura 19 donde se identifican tres partes: a) estado, b) lógica de próximo estado y c) lógica de salida. El límite del contador se describe mediante una constante. Para describir la lógica de próximo estado se utiliza un incrementador, un comparador y un multiplexor. La condición que se evalúa es el valor del contador (comparación).

Contador módulo

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity cntrmod is
 port (   reloj, pcero : in  std_logic;
          Q: out             std_logic_vector (3 downto 0) );
end cntrmod;

architecture trespart of cntrmod is
constant modu: integer := 7;
signal r_reg: std_logic_vector (3 downto 0);
signal r_prox: std_logic_vector (3 downto 0);
 begin
        --Estado
 reg:Process (reloj, pcero)
    begin
            if (pcero ='1') then
                r_reg <= (others => '0 ') after 2 ns;
            elsif (reloj'event and reloj = '1') then
                r_reg <= r_prox after 2 ns;
            end if ;
 end process ;
        -- Lógica de próximo estado
                r_prox <= (others => '0') after 10 ns when conv_integer( r_reg) = modu else
                            r_reg +1 after 10 ns ;
        -- Lógica de salida
                q <= r_reg;
 end trespart;
```

Figura 27 Descripción VHDL de un contador módulo.

Apéndice 3: Esquemas de descripción en VHDL de circuitos secuenciales

En este apéndice se presentan los esqueletos de una especificación VHDL de los autómatas de Moore y Mealy.

Autómata de Moore

- próximo estado = función (entradas, estado)
- salida = función (estado)

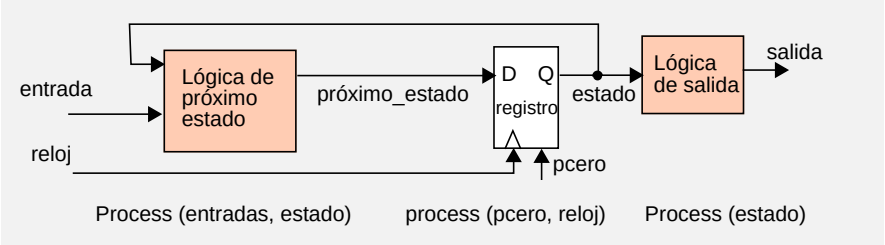


Figura 28 Esquema lógico de un autómata de Moore.

Autómata de Mealy

- próximo estado = función (entradas, estado)
- salida = función (estado, entradas)

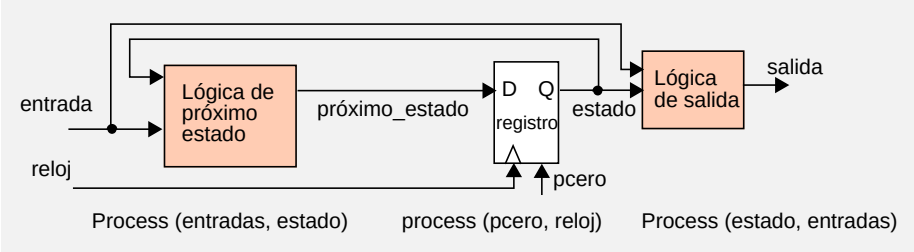


Figura 29 Esquema lógico de un autómata de Mealy.

La diferencia entre un autómata de Moore (Figura 28) y un autómata de Mealy (Figura 29) es que la lógica de salida, en el caso del autómata de Mealy, tiene como entradas las entradas del sistema.

Seguidamente se describen los esqueletos de dos alternativas para especificar en VHDL los autómatas de Moore y de Mealy. En el primer caso se utilizan tres procesos. Un proceso para cada uno de los elementos de las Figura 28 y Figura 29. Posteriormente se describe el esqueleto con dos procesos. En este caso la lógica de próximo estado y la lógica de salida se describen en el mismo proceso.

Especificación en VHDL utilizando 3 procesos.

Las posiciones de las diferencias se marcan con un trazo en la parte izquierda de la especificación.

Automáta de Moore

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity patternMoore is
    port (clk, reset: in STD_LOGIC;
        input: in STD_LOGIC;
        output: out STD_LOGIC);
end;

architecture synth of patternMoore is
                                    -- define the state type
```

Autómata de Mealy

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity patternMealy is
    port (clk, reset: in STD_LOGIC;
        input: in STD_LOGIC;
        output: out STD_LOGIC);
end;

architecture synth of patternMealy is
```

```
    type statetype is (S0, S1, . . . );
                                -- declare present state and next state
    signal state, nextstate: statetype;
begin

-- state register
    process (clk, reset)
    begin
        if reset  '1' then
                                --choose reset state
            state <= S0;
        elsif clk'event and clk  '1' then
                state <= nextstate;
        end if;
    end process;

-- next state logic
                                -- sensitive to both input and present state
    process (state, input)
    begin
        case state is
            when S0  => if (input = ...) then
                    nextstate <= S1;
                else
                    nextstate <= S0;
                end if;
            . . .
            when others => nextstate  ... ;
        end case;
    end process;

                                -- output logic - or can use concurrent statements
                                -- output logic: difference between Moore and Mealy is in the ouput function, Mealy depends on input
    process (state)
    begin
        if state = . . . then
            output < = . . .
        .elsif . . .
        ...
        end if ;

    end process ;
end;
```

```
    type statetype is (S0, S1, . . .);
    signal state, nextstate: statetype;
begin

-- state register
    process (clk, reset)
    begin
        if reset  '1' then

            state <= S0;
        elsif clk'event and clk  '1' then
            state <= nextstate;
        end if;
    end process;

-- next state logic
    process (state, input)
    begin
        case state is
            when S0 => if (input = ...)  then
                    nextstate <= S1;
                else
                    nextstate <= S0;
                end if;
            . . .
            when others => nextstate  ... ;
        end case;
    end process;

    process (state, input)
    begin
        case state is
            when S0 =>
                if (input = ...) then
            ...
            when . . .
        end case;
    end process ;
end;
```

Especificación en VHDL utilizando 2 procesos

Las posiciones de las diferencias se marcan con un trazo en la parte izquierda de la especificación.

Automáta de Moore

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity patternMoore is
    port (clk, reset: in STD_LOGIC;
```

Autómata de Mealy

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity patternMealy is
    port (clk, reset: in STD_LOGIC;
```

```
        input: in STD_LOGIC;
        output: out STD_LOGIC);
end;

architecture synth of patternMoore is
                                    -- define the state type
    type statetype is (S0, S1, . . .);
                                -- declare present state and next state
    signal state, nextstate: statetype;
begin

-- state register
    process (clk, reset)
    begin
        if reset  '1' then
                                --choose reset state
            state <= S0;
        elsif clk'event and clk  '1' then
                state <= nextstate;
        end if;
    end process;

-- next state logic, output logic
                                -- sensitive to both input and present state
    process (state, input)
    begin
        case state is
            when S0  =>
            --Moore ouput before the if statement


                output <= ... ;
                if (input = ...) then
                    nextstate <= S1;
                else
                    nextstate <= S0;

                end if;
            . . .
            when others => nextstate  ... ;
                output <= ... ;
                nextstate  ... ;
        end case;
    end process;
end;
```

```
        input: in STD_LOGIC;
        output: out STD_LOGIC);
end;

architecture synth of patternMealy is

    type statetype is (S0, S1, . . .);
    signal state, nextstate: statetype;
begin

-- state register
    process (clk, reset)
    begin
        if reset  '1' then

            state <= S0;
        elsif clk'event and clk  '1' then
            state <= nextstate;
        end if;
    end process;

-- next state logic, output logic
    process (state, input)
    begin
        case state is
            when S0 =>
            --normally, Mealy output goes after the if statement
            -- NOTE: Some Mealy outputs act like Moore outputs.
            -- output <= ...; can be moved before the IF statement.
                if (input = ...)  then
                    nextstate <= S1;
                    output <= ... ;
                else
                    nextstate <= S0;
                    output <= ... ;
                end if;
            . . .
            when others =>
                output <= ... ;
                nextstate  ... ;
        end case;
    end process;
end;
```

Resumen

Especifcación de un autómata utilizando tres procesos.

- proceso estado: registros del autómata
  (el mismo proceso para Moore y Mealy)
- proceso próximo estado
  (el mismo proceso para Moore y Mealy)
- Lógica de salida: pueden utilizarse sentencias concurrente o procesos.
  Autómata de Moore: dependen sólo del estado.
  Autómata de Mealy: dependen del estado y de las entradas.

Especificación de un autómata utilizando dos procesos.

En un autómata de Moore las salidas se asignan antes de la sentencia condicional que evalúa entradas.

En un autómata de Mealy las salidas se asignan después de la sentencia condicional que evalúa entradas.

Apéndice 4: Descripción en VHDL de un contador y de un registro

Descripción VHDL de un contador.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity contador is
port (        reloj, pcero, inic : in std_logic;
              q: out                  std_logic );
end contador;

architecture trespart of contador is
signal r_reg: std_logic_vector (1 downto 0);
signal r_prox: std_logic_vector (1 downto 0);
begin

--Estado
reg: Process (reloj, pcero)
begin
        if (pcero =’1’) then
              r_reg <= (others => ’1’) after 2 ns;
        elsif (reloj’event and reloj = ’1’) then
               r_reg <= r_prox after 2 ns;
        end if ;
end process ;

-- Logica de proximo estado
p_r: process (inic, r_reg)
begin
         if (inic = ’1’ and r_reg ="11") then
              r_prox <= (others => ’0’);
         elsif r_reg ="11" then
              r_prox <= r_reg;
         else
              r_prox <= r_reg +1;
         end if;
end process;

-- Logica de salida
   with r_reg select
              q <= ’1’ when "11",
              ’0’ when others;
end trespart;
```

Descripción VHDL de un registro.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY reg IS PORT (
    reloj: IN STD_LOGIC;
    pcero: IN STD_LOGIC;
    D: IN STD_LOGIC;
    Q: OUT STD_LOGIC);
END reg;

ARCHITECTURE compor OF reg IS
BEGIN
  PROCESS(reloj,pcero)
   BEGIN
    IF (pcero = '1') THEN
      Q <= '0';
    ELSIF (reloj'EVENT AND reloj = '1') THEN
      Q <= D;
    END IF;
END PROCESS;
END compor;
```