Apuntes de VHDL de la asignatura Electrónica Digital

autor	**Bruno Montero Hernangómez**
siglas de la asignatura	**EDIG**
fecha de última modificación	**2025-08-26**

Índice

CAPÍTULO 1

Introducción

VHDL (Very high speed integrated circuits Hardware Description Language) es un lenguaje de descripción del hardware que permite describir circuitos síncronos y asíncronos para acelerar el proceso de su diseño antes de montarlo en una PLD (típicamente una FPGA).

A diferencia de un lenguaje de programación de software, los lenguajes de descripción de hardware requieren el manejo de estructuras, concurrencia y tiempo, describiendo los sistemas como interconexiones entre sus componentes.

Una comprensión sólida de los lenguajes de descripción de hardware requiere distinguir claramente entre las secciones **concurrentes** y **secuenciales** de un circuito. En este contexto, es fundamental adoptar una perspectiva orientada a **componentes**, considerando cada módulo como una entidad autónoma con entradas y salidas bien definidas, en lugar de pensar en términos de funciones o variables como en la programación tradicional. Además, resulta esencial visualizar las conexiones entre componentes como **interconexiones físicas** (cables) que transportan señales, lo que permite modelar de manera precisa el comportamiento y la estructura del sistema digital. Esta aproximación facilita el diseño jerárquico y modular, y es clave para la correcta interpretación y síntesis de los circuitos descritos en VHDL u otros lenguajes similares.

CAPÍTULO 2

Elementos básicos

Un sistema digital se describe atendiendo a dos aspectos principales:

1. Su **aspecto exterior**: Compuesto por puertos (típicamente entradas y salidas) considerando el sistema como una caja negra. Se define mediante la entidad (**`entity`**) del sistema.
2. Su **aspecto interior**: Involucra el comportamiento interno del circuito. Se define mediante la arquitectura (**`architecture`**) del sistema.

La descripción de un sistema debe que incluir todas las librerías y paquetes apropiados (i.e. que se utilizarán para definirlo). Estas librerías y paquetes tendrán que aparecer antes de la definición de la entidad y arquitectura del sistema.

El paquete `STD` se incluye por defecto. Los paquetes que se usan con más frecuencia (dentro de la librería IEEE) son:

- `IEEE.STD_LOGIC_1164.ALL`
- `IEEE.NUMERIC_STD.ALL`

Para incluir una librería y paquete se tendrá que referenciar de la siguiente forma:

```VHDL
library ⟨nombre de la librería⟩;
use ⟨nombre del paquete⟩;
```

`IEEE.NUMERIC_STD.ALL` introduce los tipos **`signed`** y **`unsigned`** sobre los que se permite realizar operaciones aritméticas. Hay otras librerías dentro de VHDL que permiten la manipulación aritmética (bien sea entre tipos `signed` y `unsigned` o directamente entre `STD_LOGIC` y `STD_LOGIC_VECTOR`); sin embargo, `IEEE.NUMERIC_STD.ALL` **es la única estandarizada y la más recomendada**.

CAPÍTULO 3

Tipos en VHDL

Los tipos principales de VHDL son:

Nombre de tipo	Escritura	Explicación
`bit`	'⟨bit⟩'	Admite los valores `'0'` o `'1'`.
`bit_vector(rango)`	"⟨cadena de bits⟩"(⟨descripción del rango⟩)	Admite una cadena de bits con el rango especificado.
`std_logic`	'⟨bit real⟩'	Admite los valores `'0'`, `'1'`, `'X'`, `'U'`, `'Z'`, `'W'`, `'L'`, `'H'` o `'-'`.
`std_logic_vector(rango)`	"⟨cadena de bits reales⟩"(⟨descripción del rango⟩)	Admite una cadena de bits reales con el rango especificado.
`signed(rango)`	"⟨cadena de bits reales⟩"(⟨descripción del rango⟩)	Admite una cadena de bits reales con el rango especificado.
`unsigned(rango)`	"⟨cadena de bits reales⟩"(⟨descripción del rango⟩)	Admite una cadena de bits reales con el rango especificado.
`boolean`	⟨booleano⟩	Admite `true` o `false`.

NB: La definición del rango no solo describe la longitud de la cadena, sino también su dirección. Si se define una cadena de izquierda a derecha, el primer bit por la izquierda será el mas significativo; si se hace de derecha a izquierda, el primer bit por la izquierda será el más significativo.

En el caso de `bit_vector(rango)`, `std_logic_vector(rango)`, `signed(rango)` y `unsigned(rango)` los rangos se pueden definir utilizando:

- ⟨posición del bit más significativo⟩ **downto** ⟨posición del bit menos significativo⟩ (bit más significativo a la izquierda)
- ⟨posición del bit menos significativo⟩ **to** ⟨posición del bit más significativo⟩ (bit más significativo a la derecha).

NB: Los tipos signed y unsigned permiten realizar operaciones con las cadenas de bits.

Nombre de tipo	Escritura	Explicación
`integer rango`	⟨número entero⟩	Admite números enteros dentro del rango. El máximo valor definido es `max`.
`natural rango`	⟨número natural⟩	Admite números naturales dentro del rango. El máximo valor definido es `max`.
`positive rango`	⟨número positivo⟩	Admite números positivos dentro del rango. El máximo valor definido es `max`.
`real rango`	⟨número real⟩	Admite números positivos dentro del rango. El máximo valor definido es `max`.
`time`	⟨duración⟩ ⟨unidad de tiempo⟩	Admite cualquier número con las siguientes unidades temporales: `hr`, `min`, `sec`, `ms`, `us`, `ns`, `ps`, `fs`.

La notación para definir los rangos es la misma que ya se ha explicado y suele ser `0 to max` (en este caso no se utilizan paréntesis).

Nombre de tipo	Escritura	Explicación
`character`	'⟨carácter⟩'	Admite cualquier carácter en ascii
`string`	"⟨cadena de caracteres⟩"	Admite cualquier cadena de caracteres en ascii

El casting se realiza introduciendo el nombre de un objeto entre paréntesis precedido por el nuevo tipo al que quiere convertir.

```VHDL
⟨nuevo tipo⟩(⟨objeto⟩)
```

Además del casting se pueden utilizar las funciones `to_⟨nuevo tipo⟩(⟨objeto⟩)`. Estas funciones son menos concisas, pero realizan comprobaciones y manipulaciones adicionales (como ajustar el tamaño del vector `to_⟨nuevo tipo⟩(⟨objeto⟩, ⟨length⟩)`, donde resulta útil poder consultar la longitud de los objetos utilizando `⟨objeto⟩'lenght`) y pueden manejar conversiones más completas que el casting. A pesar de que en construcciones complejas es preferible utilizar estas funciones, en las descripciones de circuitos básicos que se ven en la asignatura se utilizará el casting en mayor medida.

Las funciones `to_⟨nuevo tipo⟩(⟨objeto⟩)` más importantes son:

Función	Descripción
`to_bit()`	Convierte `std_logic` a `bit`.
`to_stdulogic()`	Convierte `bit` a `std_logic`.
`to_bitvector()`	Convierte `std_logic_vector` a `bit_vector`.
`to_stdlogicvector()`	Convierte `bit_vector` a `std_logic_vector`.

CAPÍTULO 4

Operadores de VHDL

Los operadores principales de VHDL son:

Símbolo	Explicación
<=	Conexión

Símbolo	Explicación
and	Operador lógico y
or	Operador lógico o
not	Operador lógico de negación
nand	Operador lógico no y
nor	Operador lógico no o
xor	Operador lógico o exclusivo
xnor	Operador lógico no o exclusivo

Símbolo	Explicación
=	Operador relacional igual que
/=	Operador relacional diferente a
<	Operador relacional menor que
>	Operador relacional mayor que
<=	Operador relacional menor o igual que
>=	Operador relacional mayor o igual que

Símbolo	Explicación
&	Concatenación
+	Suma
-	Resta
*	Multiplicación

Símbolo	Explicación
/	División
**	Exponente
mod	Módulo
abs	Absoluto
rem	Resto

CAPÍTULO 5

Entity

La entidad de un sistema se refiere a la estructura externa del sistema, compuesta por los puertos (**ports**) con entradas y salidas del sistema así como sus tipos. En la entidad se describe toda la información necesaria para conectar el sistema con sistemas externos.

Dentro de la entidad también se pueden definir unos valores genéricos (**generic**) que se utilizan para definir propiedades (i.e. declarar constatnes) del circuito independientemente de su arquitectura.

La sintaxis para definir la entidad de un sistema general es:

```
1  entity ⟨nombre del sistema⟩ is                                    VHDL
2      generic(                              Sección para valores genéricos
3          ⟨identificador de la constante1⟩ : ⟨tipo de la
   constante1⟩ := ⟨valor que toma la constante1⟩;
4          ⟨identificador de la constante2⟩ : ⟨tipo de la
   constante2⟩ := ⟨valor que toma la constante2⟩;
5          ...;
6      );
7      port(                              Sección para declarar los puertos
8          ⟨identificador de la entrada1⟩: in ⟨tipo de la entrada1⟩;
9          ⟨identificador de la entrada2⟩: in ⟨tipo de la entrada2⟩;
10         ...;
11
12         ⟨identificador de la salida1⟩ : out ⟨tipo de la salida1⟩;
13         ⟨identificador de la salida2⟩ : out ⟨tipo de la salida2⟩;
14         ...;
15
16         ⟨identificador del puerto genérico1⟩ : modo ⟨tipo del
   puerto genérico1⟩;
17         ⟨identificador del puerto genérico2⟩ : modo ⟨tipo del
   puerto genérico2⟩;
18         ...;
19
20         -- si hay más de un puerto del mismo modo con el mismo
   tipo pueden definirse conjuntamente separando los identificadors
   con comas
```

```
21
22      );
23  end ⟨nombre del sistema⟩;
```

Los modos de los puertos que pueden tener los sistemas son **`in`** (entrada), **`out`** (salida), `inout` (entrada y salida) y `buffer` (buffer). En la asignatura solo se hará uso de los dos primeros modos de puertos (`in` y `out`).

NB: Los valores de entrada (`in`) no se podrán modificar en el sistema que los recibe y los de salida (`out`) se podrán modificar, pero nunca se podrán evaluar para el desarrollo interno del sistema que los devuelve.

Los valores genéricos se suelen utilizar para incluir constantes útiles para el funcionamiento del sistema para su posterior utilización.

CAPÍTULO 6

Architecture

La arquitectura se refiere a las operaciones necesarias para relacionar los valores de los puertos del sistema. En la arquitectura se definen las operaciones internas de forma genérica para describir como debe de reaccionar el sistema (i.e. se define el funcionamiento del módulo definido en una entidad). La arquitectura siempre va ligada a una entidad de la que describe su funcionamiento.

Dentro de la arquitectura se hacen las declaraciones de los objetos, que se utilizarán en la arquitectura, y se conectan los elementos concurrentes (las partes del sistema que están conectadas). Los objetos principales consisten en señales (`signal`), constantes (`constant`) y variables (`variable`). Las declaraciones de objeto se harán antes de la sección concurrente y los elementos concurrentes se conectan en la sección concurrente.

Los elementos concurrentes se denominan componentes (`component`) y tienen que describirse en otro módulo de VHDL para poder utilizarse. El identificador del componente corresponde con el nombre del módulo al que hace referencia. Para declarar un componente que se utilizará en una arquitectura se utiliza:

```VHDL
component ⟨identificador del componente⟩
    port(
        ⟨identificación del puerto1⟩ : modo ⟨tipo del puerto1⟩;
        ⟨identificación del puerto2⟩ : modo ⟨tipo del puerto2⟩;
        ...;
    );
end component;
```

La sección concurrente dentro de la arquitectura se describe utilizando la palabra:

```VHDL
1 begin
```

6.1. Objetos principales

Las señales hacen referencia a conexiones en forma de cables. Conectarán los elementos concurrentes del sistema. Una señal se declara de la siguiente forma:

```VHDL
1 signal ⟨identificador de la señal⟩ : ⟨tipo de la señal⟩;
2 -- si hay más de una señal con el mismo tipo, pueden definirse
conjuntamente separando los identificadores con comas
```

Se puede inicializar un valor para la señal de la siguiente forma:

```VHDL
1 signal ⟨identificador de la señal⟩ : ⟨tipo de la señal⟩ :=
⟨valor de inicio⟩;
```

el valor de inicio suele ser `others <= 'X'` que asigna bits indefinidos o `others <= '0'` que asigna un valor nulo. Sin embargo, es una buena práctica evitar abusar de la inicialización de las señales y utilizar conexiones siempre que se pueda para evitar posibles errores de conexión y desconexión de las señales.

Para realizar una conexión entre componentes y señales (en la sección concurrente) se utiliza:

```VHDL
1 ⟨identificador de la señal⟩ <= ⟨valor o expresión que toma⟩;
```

NB: Los puertos definidos en la entidad del módulo o un componente dentro de su arquitectura tienen señales asociadas con el identificador con las que se asignan.

Las constantes hacen referencia a valores que no se pueden modificar durante la simulación. Una constante se declara y define de la siguiente forma:

```VHDL
1 constant ⟨identificador de la constante⟩ : ⟨tipo de la
constante⟩ := ⟨valor de la constante⟩;
2 -- las constantes se tienen que inicializar en su declaración y
nunca cambian de valor durante la simulación
3 -- si hay más de una constante con el mismo tipo y valor, pueden
definirse conjuntamente separando los identificadores con comas
```

Las variables hacen referencia a valores que pueden modificarse durante la simulación con una sentencia de asignación. Una variable se declara de la siguiente forma:

```vhdl
variable ⟨identificador de la variable⟩ : ⟨tipo de la variable⟩;
-- si hay más de una variable con el mismo tipo, pueden definirse conjuntamente separando los identificadores con comas
```

De forma análoga a las señales y constantes, se puede inicializar una variable de la siguiente forma:

```vhdl
variable ⟨identificador de la variable⟩ : ⟨tipo de la variable⟩ := ⟨valor de inicio⟩;
```

Para hacerse una asignación se utiliza:

```vhdl
⟨identificador de la variable⟩ := ⟨valor o expresión que toma⟩;
-- se puede hacer una asginación en su declaración haciendo variable ⟨identificador de la variable⟩ : ⟨tipo de la variable⟩ := ⟨valor o expresión que toma⟩;
```

Las variables se suelen utilizar en la creación de bucles y pueden producir efectos perversos en la simulación.

6.2. Estilos de arquitectura

Hay varios estilos para definir la arquitectura deun sistema, sindo los pincipales el funcional, estructural y mixto (combinación de funcional y estructural).

El **estilo funcional** consiste en utilizar las operaciones proporcionadas por VHDL para describir el funcionamiento del sistema. En este estilo no se utilizan componentes, que hacen referencia a entidades ya descritas, y no es necesario definir señales.

El **estilo estructural** consiste en declarar componentes (que hacen referencia a entidades ya descritas) antes de la sección concurrente (creando identificadores adecuados para sus puertos) que luego se utilizarán y conectarán en la sección concurrente.

Las arquitecturas **funcionales** son de la forma:

```vhdl
architecture ⟨nombre de la arquitectura⟩ of ⟨identificador de la entidad⟩ is
begin
    -- conexión de puertos utilizando operaciones proporcionadas por VHDL
end ⟨nombre de la arquitectura⟩;
```

Las arquitecturas **estructurales** son de la forma:

```VHDL
architecture ⟨nombre de la arquitectura⟩ of ⟨identificador de la entidad⟩ is

    component ⟨identificador del componente1⟩
        port(
            ⟨identificación del puerto1⟩ : modo ⟨tipo del puerto1⟩;
            ⟨identificación del puerto2⟩ : modo ⟨tipo del puerto2⟩;
            ...;
        );
    end component;

    component ⟨identificador del componente2⟩
        port(
            ⟨identificación del puerto1⟩: modo ⟨tipo del puerto1⟩;
            ⟨identificación del puerto2⟩: modo ⟨tipo del puerto2⟩;
            ...;
        );
    end component;
    ...

    signal ⟨identificador de la señal1⟩ : ⟨tipo de la señal1⟩;
    signal ⟨identificador de la señal2⟩ : ⟨tipo de la señal2⟩;
    ...;

begin
    -- operaciones proporcionadas por VHDL
    -- asignaciones de señal directa
    -- asignaciones de señal condicional
    -- asignaciones de señal seleccionada
    -- instanciación de componentes
    -- sentencias de verificación
    -- procesos

end ⟨nombre de la arquitectura⟩;
```

6.2.1. Elementos concurrentes

Existen 6 formas de conexión utilizadas al definir la parte concurrente de la arquitectura de un módulo.

1. **Asignación de señal directa** (<=)

   Se conectan señales utilizando el operador <=.

```VHDL
⟨señal1⟩ <= ⟨señal2⟩;
```

En las simulaciones estas conexiones pueden variar en función de un valor temporal (`time`) que se inicia en la ejecución de la simulación.

```VHDL
⟨señal variable⟩ <= ⟨primera señal⟩, ⟨segunda señal⟩ after ⟨tiempo1⟩, ⟨tercera señal⟩ after ⟨tiempo2⟩, ...;
```

2. **Asignación condicional** (when-else)

   Se conecta una señal a otras en base a unas condiciones.

```VHDL
⟨señal modificable⟩ <=
    ⟨señal1⟩ when ⟨condición1⟩ else
    ⟨señal2⟩ when ⟨condición2⟩ else
    ⟨señal3⟩ when ⟨condición3⟩ else
    ...
    ⟨señaln⟩;
-- si no se cumplen ninguna de las condiciones, se conectará a la ⟨señaln⟩
```

3. **Asignación de señal seleccionada** (with-select-when)

   Se conecta una señal a otra en base al valor de una señal de selección.

```VHDL
with ⟨señal para la selección⟩ select ⟨señal modificable⟩ <=
    ⟨señal1⟩ when ⟨valor1 de la señal para la selección⟩,
    ⟨señal2⟩ when ⟨valor2 de la señal para la selección⟩,
    ⟨señal3⟩ when ⟨valor3 de la señal para la selección⟩,
    ...
    ⟨señaln⟩ when others;
-- se conectará a la ⟨señaln⟩ para el resto de valores no cubiertos en la expresión
```

4. **Instanciación de componentes**

   Se conectan los puertos de los componentes.

```VHDL
⟨identificador del elemento⟩ : ⟨identificador del componente⟩ port map(
    ⟨identificador del puerto1⟩ => ⟨señal1⟩,
    ⟨identificador del puerto2⟩ => ⟨señal2⟩,
    ...
);
```

NB: La única forma de acceder y conectar los puertos de los componentes es a través de `portmap`, y si se quiere conectar los puertos de varios componentes habrá que definir señales intermedias.

5. **Sentencias de verificación**

   Reporta mensajes para evaluar resultados.

```VHDL
assert(⟨condición lógica⟩)
    report ⟨string que mostrar por pantalla⟩
    severity ⟨nivel de severidad⟩;
```

   Los posibles niveles de severidad de más bajo a más alto son: `note<warning<error<failure`

6. **Procesos**

   Se verán con más detalle en la siguiente sección.

6.2.2. Procesos

Los procesos son sentencias que se utilizan para asignar valores sin la necesidad de tener definidas todas las señales de las que dependen. Esto implica que las sentencias de procesos almacenan valores de sus entradas y permiten describir **circuitos secuenciales**. En la simulación de los circuitos solo se ejecutan las instrucciones de los procesos en el origen de tiempos de la simulación (primer instante) y cuando alguna de las señales de su **lista de sensibilidad** cambia de valor o bajo requisitos impuestos por **cláusulas de espera** («wait clauses»). Sus instrucciones internas se realizarán de forma «secuencial». Esto es especialmente importante ya que permite describir circuitos secuenciales (mantener una memoria de los valores pasados) y reescribir valores.

NB: Dentro de un proceso, las instrucciones se ejecutan de forma secuencial, y el orden de las sentencias afecta al resultado. Si hay más de una asignación a una misma señal, solo la última tendrá efecto, ya que las asignaciones a señales dentro de procesos se programan y se actualizan al finalizar el proceso.

Las cláusulas de espera son:

Cláusula	Explicación
`wait`	Suspende el funcionamiento de la sentencia del proceso de forma indefinida.

Cláusula	Explicación
`wait on (⟨señales separadas por comas⟩)`	Suspende el funcionamiento de la sentencia del proceso hasta que haya un evento en las ⟨señales separadas por comas⟩ (i.e. alguna de ellas cambie de valor).
`wait until (⟨condición lógica⟩)`	Suspende el funcionamiento de la sentencia del proceso hasta que la ⟨condición lógica⟩ sea cierta (`true`).
`wait for ⟨time⟩`	Suspende el funcionamiento de la sentencia del proceso hasta que haya transcurrido el ⟨tiempo⟩ impuesto.

La escritura de un roceso sin lista de sensibilidad sería:

```VHDL
⟨nombre opcional del proceso⟩: process
begin
    -- sentencias secuenciales
    -- sentencias condicionales
    -- clausulas de espera
end process;
```

Si hay una lista de sensibilidad no podrá haber clausulas de espera. Sin embargo, para el diseño de circuitos secuenciales siempre se recurrirá al uso de la lista de sensibilidad.

```VHDL
⟨nombre opcional del proceso⟩: process(⟨lista de sensibilidad⟩)
begin
    -- sentencias secuenciales
    -- sentencias condicionales
end process;
```

La lista de sensibilidad compone una lista de señales tales que se suspende el funcionamiento de la sentencia del proceso hasta que haya un evento en alguna de ellas (i.e. alguna de ellas cambie de valor).

En caso de que no se quiera incluir un nombre opcional para el proceso se tendría que sustituir `process` por ⟨nombre opcional del proceso⟩`: process`.

En los procesos que describen circuitos secuenciales es muy útil utilizar la expresión:

```VHDL
1 ⟨señal⟩'event
```

que se activa, tomando el valor `'1'`, cuando hay un cambio en la señal y desactiva, tomando el valor `'0'`, mientras no lo haya. Además de las funciones:

- `risign_edge(⟨señal⟩)`: que se activa, tomando el valor `'1'`, cuando hay un cambio creciente en la señal y desactiva, tomando el valor `'0'`, mientras no lo haya.
- `falling_edge`: que se activa, tomando el valor `'1'`, cuando hay un cambio decreciente en la señal y desactiva, tomando el valor `'0'`, mientras no lo haya.

Estas sentencias para señales que sean `STD_LOGIC` o `BIT` serán equivalentes a `⟨señal⟩'event and ⟨señal⟩ = '1'` y `⟨señal⟩'event and ⟨señal⟩ = '0'` respectivamente (asumiendo un cambio instantáneo).

Las sentencias de flujo de control de los procesos son:

1. **if-then-else**

```VHDL
if ⟨condición lógica⟩ then
    ⟨sentencias⟩
...
elsif otra_condición then
    ⟨sentencias⟩
...
else
    ⟨sentencias⟩
end if;
```

2. **case-when**

```VHDL
case ⟨señal para la elección⟩ is
    when ⟨valor1 de la señal para la selección⟩ =>
        ⟨sentencias⟩
    when ⟨valor1 de la señal para la selección⟩ =>
        ⟨sentencias⟩
    ...
    when others =>
        ⟨sentencias⟩
end case;
```

3. **for-loop**

```VHDL
for ⟨variable para el bucle⟩ in ⟨rango⟩ loop
    ⟨sentencias⟩
end loop;
```

donde ⟨rango⟩ será típicamente un intervalo descrito como ⟨mínimo valor⟩ to ⟨máximo valor⟩ o ⟨máximo valor⟩ downto ⟨mínimo valor⟩ dependiendo del orden en el que se quiera procesar.

4. **while-loop**

```VHDL
while ⟨condición lógica⟩ loop
    ⟨sentencias⟩
end loop;
```

6.2.3. Autómatas (máquinas de estados)

VHDL permite realizar descripciones algorítmicas de alto nivel de autómatas. Esto significa que permite describir los autómatas de una forma similar a como se haría desde un diagrama de estados finitos (y otras formas menos triviales que no se estudiarán en la asignatura).

La descripción típica de un autómata parte de la **declaración de un tipo enumerado** asignando un identificador a cada estado del autómata. La herramienta de síntesis se encargará de la codificación óptima de estos estados.

Los enumerados se declaran de la siguiente forma:

```VHDL
type ⟨identificador del enumerado⟩ is (⟨identificador1⟩,
⟨identificador2⟩, ..., ⟨identificadorn⟩);
```

Además de la declaración del enumerado se necesitará **identificar el estado actual y el estado siguiente con dos señales del tipo descrito por el enumerado**. En el comienzo de la ejecución de la simulación se deberá asignar a las señales de estado actual y estado siguiente el estado de reposo de la siguiente forma:

```VHDL
signal ⟨identificador de la señal⟩ : ⟨tipo de la señal⟩ :=
⟨valor de inicio⟩;
```

Además de definir el registro que realiza el paso del estado actual al estado siguiente, dentro de la arquitectura del autómata se suele utilizar la construcción **case-when** dentro de procesos para asignar el estado siguiente a partir de las señales de entrada (lógica de estado siguiente, F) y asignar la salida en base al estado actual (lógica de salida, G).

Por lo que la arquitectura de los autómatas será de la siguiente forma:

VHDL

```
architecture ⟨nombre de la arquitectura⟩ of ⟨identificador de la entidad⟩ is

    type ⟨identificador del enumerado de estados⟩ is (⟨identificador del estado1⟩, ⟨identificador del estado2⟩, ..., ⟨identificador del estadon⟩);

    signal ⟨identificador de la señal de estado actual⟩ : ⟨identificador del enumerado de estados⟩ := ⟨identificador del estadom⟩;
    signal ⟨identificador de la señal de estado siguiente⟩ : ⟨identificador del enumerado de estados⟩ := ⟨identificador del estadom⟩;
    ...

    -- el estadom es el estado de reposo

begin

    process (clk, ⟨señal asíncrona1⟩, ..., ⟨señal asíncronan⟩)
    begin
        -- sentencias de asignación del estado actual al estado siguiente
    end process;

    process (⟨identificador de la señal de estado actual⟩, ⟨entrada1⟩, ...,  ⟨entrada1⟩)
    begin
        case ⟨identificador de la señal de estado actual⟩ is
            when ⟨identificador del estado1⟩ =>
                -- sentencias if-else en función de las entradas
            ...
        end case;
    end process;

    process (⟨identificador de la señal de estado actual⟩, ⟨entrada1⟩, ...,  ⟨entrada1⟩)
    begin
        case ⟨identificador de la señal de estado siguiente⟩ is
            when ⟨identificador del estado1⟩ =>
                -- sentencias if-else en función de las entradas
            ...
        end case;
```

Declaración del enumerado

```
34      end process;
35      -- si el autómata es de Moore, la lógica de salida no
        dependería de las entradas, las entradas no estarían en la lista
        de sensibilidad y no habría que recurrir a sentencias if-else de
        comprobación de las entradas
36
37  end ⟨nombre de la arquitectura⟩;
```

Si las conexiones de la lógica de estado siguiente y la lógica de salida son concurrentes, las señales de la lista de sensibilidad tienen que ser todas las señales y se podría realizar la misma descripción sin utilizar más procesos que el del registro. Sin embargo, simplificar la descripción de la lógica de estado siguiente y la lógica de salida utilizando una descripción concurrente puede conllevar una mayor carga y complejidad en la descripción del registro, teniendo que considerar el estado actual y más opciones para describir el funcionamiento del autómata.

Para que Xilinx reconozca correctamente una descripción como un autómata esta debe incluir una **señal que haga de reset** (síncrona o asíncrona) y siempre se debe **asignar valores al estado siguiente**; por lo que habrá que asegurarse de que el estado siguiente se asigna en todas las ramas posibles del proceso, es decir, que no queden caminos sin asignación explícita.

CAPÍTULO 7

Simulación

Para comenzar la simulación se deben seguir los siguientes pasos:

```
Simulation                                                    Pasos
{seleccionar tb}
ISim Simulator
Simulate Behavioural Model
{click derecho}
Run all
```

Para reiniciar la simulación habrá que introducir en la línea de comandos de Xilinx:

```
ISim> restart                                                 HDL
```

y para realizarla en un periodo concreto:

```
ISim> run ‹tiempo deseado›                                    HDL
```

Para visualizar las señales acotas al tiempo de la simulación en el .wcfg:

```
{click sobre el cronograma}                                   Pasos
zoom to full view
```

Para entrar a los procesos que se realizan paso por paso:

```
ISim> step                                                    HDL
```

CAPÍTULO 8

Práctica 1

8.1. circuito1

Módulo de sistema de operaciones lógicas.

VHDL

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity circuito1 is                                    Entidad
5      port(
6          A : in STD_LOGIC;
7          B : in STD_LOGIC;
8          C : in STD_LOGIC;
9          F1 : out STD_LOGIC
10     );
11 end circuito1;
12
13 architecture Behavioral of circuito1 is                Arquitectura
14 begin
15     F1 <= (not(A) and B) or (not(A) and B and C);
16 end Behavioral;
```

8.2. circuito2

Módulo de sistema de operaciones lógicas.

VHDL

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity circuito2 is
5      port(
6          A : in STD_LOGIC;
```

```
        B : in STD_LOGIC;
        C : in STD_LOGIC;
        F2 : out STD_LOGIC
    );
end circuito2;

architecture Behavioral of circuito2 is
begin
    F2 <= ((not(A) or B) and (A or not(B))) and (B or not(C));
end Behavioral;
```

8.3. multiplexor

Multiplexor 2x8.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity multiplexor is
    port(
        F1 : in STD_LOGIC;
        F2 : in STD_LOGIC;
        S : in STD_LOGIC;
        Z : out STD_LOGIC
    );
end multiplexor;

architecture Behavioral of multiplexor is
begin
    with S select Z <=
        F1 when '0',
        F2 when '1',
        '0' when others;
end Behavioral;
```

8.4. full_system

Sistema completo.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity full_system is
    port(
```

```
6         A : in STD_LOGIC;
7         B : in STD_LOGIC;
8         C : in STD_LOGIC;
9         S : in STD_LOGIC;
10        Z : out STD_LOGIC
11    );
12 end full_system;
13
14 architecture structural of full_system is
15
16    component circuito1
17        port(
18            A, B, C : in STD_LOGIC;
19            F1 : out STD_LOGIC
20        );
21    end component;
22
23    component circuito2
24        port(
25            A, B, C : in STD_LOGIC;
26            F2 : out STD_LOGIC
27        );
28    end component;
29
30    component multiplexor
31        port(
32            F1, F2, S : in STD_LOGIC;
33            Z : out STD_LOGIC
34        );
35    end component;
36
37    signal F1, F2 : STD_LOGIC;
38
39 begin
40    C1 : circuito1 port map(
41        A => A,
42        B => B,
43        C => C,
44        F1 => F1
45    );
46    C2 : circuito2 port map(
```

```
47          A => A,
48          B => B,
49          C => C,
50          F2 => F2
51      );
52      M : multiplexor port map(
53          F1 => F1,
54          F2 => F2,
55          S => S,
56          Z => Z
57      );
58  end structural;
```

CAPÍTULO 9

Práctica 2

9.1. CP

Codificador de prioridad (realizará la operación de log base 2).

VHDL

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

entity CP is
    port(
        e : in STD_LOGIC_VECTOR (7 downto 0);
        y : out STD_LOGIC_VECTOR (2 downto 0);
        idle : out STD_LOGIC
    );
end CP;

architecture Behavioral of CP is

begin

    y <=
        "111" when e(7) = '1' else
        "110" when e(6) = '1' else
        "101" when e(5) = '1' else
        "100" when e(4) = '1' else
        "011" when e(3) = '1' else
        "010" when e(2) = '1' else
        "001" when e(1) = '1' else
        "000";
    idle <= not(e(7) or e(6) or e(5) or e(4) or e(3) or e(2) or
    e(1) or e(0));
```

```
26
27 end Behavioral;
```

9.2. ROM

Memoria ROM que realiza la operación de multiplicación por -7.25.

```VHDL
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity ROM is
5      port(
6          addr : in STD_LOGIC_VECTOR (2 downto 0);
7          data : out STD_LOGIC_VECTOR (7 downto 0)
8      );
9  end ROM;
10
11 architecture Behavioral of ROM is
12
13 begin
14
15     with addr select data <=
16         "10001100" when "100",
17         "10101001" when "011",
18         "11000110" when "010",
19         "11100011" when "001",
20         "00000000" when others;
21
22 end Behavioral;
```

9.3. MUX8_8

Multiplexor 8x8.

```VHDL
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity MUX8_8 is
5      port(
6          D0 : in STD_LOGIC_VECTOR (7 downto 0);
7          D1 : in STD_LOGIC_VECTOR (7 downto 0);
8          D2 : in STD_LOGIC_VECTOR (7 downto 0);
9          D3 : in STD_LOGIC_VECTOR (7 downto 0);
10         D4 : in STD_LOGIC_VECTOR (7 downto 0);
```

```
        D5 : in STD_LOGIC_VECTOR (7 downto 0);
        D6 : in STD_LOGIC_VECTOR (7 downto 0);
        D7 : in STD_LOGIC_VECTOR (7 downto 0);
        S : in STD_LOGIC_VECTOR (2 downto 0);
        Y : out STD_LOGIC_VECTOR (7 downto 0)
    );
end MUX8_8;

architecture Behavioral of MUX8_8 is

begin

    with S select Y <=
        D0 when "000",
        D1 when "001",
        D2 when "010",
        D3 when "011",
        D4 when "100",
        D5 when "101",
        D6 when "110",
        D7 when "111";

end Behavioral;
```

9.4. MUX8_1

Multiplexor 8x1.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX8_1 is
    port(
        D : in STD_LOGIC_VECTOR (7 downto 0);
        S : in STD_LOGIC_VECTOR (2 downto 0);
        Y : out STD_LOGIC
    );
end MUX8_1;

architecture Behavioral of MUX8_1 is

begin
```

```
15
16      with S select Y <=
17          D(0) when "000",
18          D(1) when "001",
19          D(2) when "010",
20          D(3) when "011",
21          D(4) when "100",
22          D(5) when "101",
23          D(6) when "110",
24          D(7) when "111";
25
26  end Behavioral;
```

9.5. OPERA

Módulo de operaciones aritméticas.

Realiza:

Código de control	Operación	Estado de la salida (error)
000	A + B	"1" si hay desbordamiento, "0" si no lo hay
001	A - B	"1" si hay desbordamiento, "0" si no lo hay
010	log2(parte entera de A) (log2 equivale a realizar una codificación de prioridad)	"1" si (parte entera de A) ≤ 0, "0" en caso contrario
011	−7.25 x log2(parte entera de A) (la multiplicación se realiza con la ROM)	"1" si (parte entera de A) ≤ 0, "0" en caso contrario
100	A / 2 (equivale a desplazar los bits una posición a la derecha y extender el bit de signo)	siempre "0"
101	«11111111» si B ≤ A, «00000000» en ca-	siempre "0"

Código de control	Operación	Estado de la salida (error)
	so contrario (las comparaciones se pueden realizar con facilidad convirtiendo los vectores de bits en representación C2 a tipo signed)	
110	«11100111» si −23.5 < B < + 24.75, «00011000» en caso contrario	siempre “0”
111	siempre «11001100»	siempre “0”

VHDL

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity OPERA is
    port(
        A : in STD_LOGIC_VECTOR (7 downto 0);
        B : in STD_LOGIC_VECTOR (7 downto 0);
        control : in STD_LOGIC_VECTOR (2 downto 0);
        resultado : out STD_LOGIC_VECTOR (7 downto 0);
        error : out STD_LOGIC
    );
end OPERA;

architecture mixed of OPERA is

    component CP
        port(
            e : in STD_LOGIC_VECTOR (7 downto 0);
            y : out STD_LOGIC_VECTOR (2 downto 0);
            idle : out STD_LOGIC
        );
    end component;

```

```
25      component ROM
26          port(
27              addr : in STD_LOGIC_VECTOR (2 downto 0);
28              data : out STD_LOGIC_VECTOR (7 downto 0)
29          );
30      end component;
31
32      component MUX8_8
33          port(
34              D0 : in STD_LOGIC_VECTOR (7 downto 0);
35              D1 : in STD_LOGIC_VECTOR (7 downto 0);
36              D2 : in STD_LOGIC_VECTOR (7 downto 0);
37              D3 : in STD_LOGIC_VECTOR (7 downto 0);
38              D4 : in STD_LOGIC_VECTOR (7 downto 0);
39              D5 : in STD_LOGIC_VECTOR (7 downto 0);
40              D6 : in STD_LOGIC_VECTOR (7 downto 0);
41              D7 : in STD_LOGIC_VECTOR (7 downto 0);
42              S : in STD_LOGIC_VECTOR (2 downto 0);
43              Y : out STD_LOGIC_VECTOR (7 downto 0)
44          );
45      end component;
46
47      component MUX8_1
48          port(
49              D : in STD_LOGIC_VECTOR (7 downto 0);
50              S : in STD_LOGIC_VECTOR (2 downto 0);
51              Y : out STD_LOGIC
52          );
53      end component;
54
55      signal AE, D0, D1, D2, D3, D4, D5, D6, D7, E :
        STD_LOGIC_VECTOR (7 downto 0);
56      signal L2 : STD_LOGIC_VECTOR (2 downto 0); signal idle :
        STD_LOGIC;
57
58  begin
59
60      AE <= ("000" & A(6 downto 2));                        -- no se
        puede utilizar funciones dentro de un port map (que no sean
        conversión de tipos); AE es la parte entera de A
61
62      D0 <= std_logic_vector(signed(A) + signed(B));
```

```
63      D1 <= std_logic_vector(signed(A) - signed(B));
64      LOG : CP port map(
65          e => AE,
66          y => L2,
            -- L2 es el resultado del logaritmo en base dos de la
            parte entera de A
67          idle => idle
68      );
69
70      D2 <= ("000" & L2 & "00");                              --
        no se puede utilizar funciones dentro de un port map (que
        no sean conversión de tipos); el resultado del logaritmo se
        tiene que pasar al formato deseado
71
72      KLOG : ROM port map(
73          addr => L2,
74          data => D3
75      );
76      D4 <= (A(7) & A(7 downto 1));
77      with (signed(B) <= signed(A)) select D5 <=
78          "11111111" when true,
79          "00000000" when others;
80      with (("10100010" < signed(B)) and (signed(B) < "01100011"))
        select D6 <=
81          "11100111" when true,
82          "00011000" when others;
83      D7 <= "11001100";
84
85      M88 : MUX8_8 port map(
86          D0 => D0,
87          D1 => D1,
88          D2 => D2,
89          D3 => D3,
90          D4 => D4,
91          D5 => D5,
92          D6 => D6,
93          D7 => D7,
94          S => control,
95          Y => resultado
96      );
97
```

```
98      with (to_stdlogicvector(A(7) & B(7) & D0(7))) select
        E(0) <=         -- al utilizar la librería numeric
        detecta más de una definición de concatenar, si se fuerza
        una salida std_logic_vector desaparece la ambigüedad con la
        concatenación de bit_vector
99          '1' when "110",
100         '1' when "001",
101         '0' when others;
102
103     with (to_stdlogicvector(A(7) & B(7) & D1(7))) select E(1) <=
        -- igual que arriba
104         '1' when "100",
105         '1' when "011",
106         '0' when others;
107
108     E(2) <=
109         '1' when A(7) = '1' else
            -- parte entera de A menor a 0 (bit de signo
            negativo)
110         '1' when idle = '1' else
            -- parte entera de A igual a 0 (todos los bits de la
            parte entera son 0 i.e. el idle es 1)
111         '0';
112
113     E(3) <= E(2);
114
115     E(7 downto 4) <= "0000";
116
117
118     M81 : MUX8_1 port map(
119         D => E,
120         S => control,
121         Y => error
122     );
123
124 end mixed;
```

CAPÍTULO 10

Práctica 3

10.1. flipflop_t

Biestable tipo T.

VHDL

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity flipflop_t is
5      port(
6          T: in STD_LOGIC;
7          Q: out STD_LOGIC;
8          clk: in STD_LOGIC;
9          rst: in STD_LOGIC
10     );
11 end flipflop_t;
12
13 architecture Behavioral of flipflop_t is
14
15     signal Q_ant, Q_sig: STD_LOGIC;
16
17 begin
18     process(clk, rst)
19     begin
20         if (rst = '0') then
21             Q_sig <= '0';
22         elsif (clk'event and clk = '1') then
23             if T = '1' then
24                 Q_sig <= not(Q_ant);
25             else
```

```
                Q_sig <= Q_ant;
            end if;
        end if;
    end process;

    Q_ant <= Q_sig;
    Q <= Q_sig;

end Behavioral;
```

10.2. prescaler

Generador de pulsos a cada segundo.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity prescaler is
    port(
        clk: in STD_LOGIC;
        rst: in STD_LOGIC;
        en: in STD_LOGIC;
        pulse_1_sec: out STD_LOGIC
    );
end prescaler;

architecture Behavioral of prescaler is

    signal cont: STD_LOGIC_VECTOR(2 downto 0);

begin
    process(clk, rst)
    begin
        if (rst = '0') then
            cont <= "000";
            pulse_1_sec <= '0';
        elsif (en = '1' and clk'event and clk = '1') then
            if (cont < "100") then
                cont <= std_logic_vector(unsigned(cont) + "001");
                pulse_1_sec <= '0';
            else
```

```
                cont <= "000";
                pulse_1_sec <= '1';
            end if;
        end if;
    end process;

end Behavioral;
```

10.3. reg

Registro de minutos y segundos para la alarma.

```VHDL
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity reg is
    port(
        min_in: in STD_LOGIC_VECTOR(5 downto 0);
        sec_in: in STD_LOGIC_VECTOR(5 downto 0);
        en: in STD_LOGIC;
        rst: in STD_LOGIC;
        clk: in STD_LOGIC;
        min_out: out STD_LOGIC_VECTOR(5 downto 0);
        sec_out: out STD_LOGIC_VECTOR(5 downto 0)
    );
end reg;

architecture Behavioral of reg is

    signal min_sig, sec_sig: STD_LOGIC_VECTOR(5 downto 0);

begin
    process(clk, rst)
    begin
        if (rst = '0') then
            min_sig <= "111111";
            sec_sig <= "111111";
        elsif (en = '1' and clk'event and clk = '1') then
            min_sig <= min_in;
            sec_sig <= sec_in;
        end if;
    end process;
```

```
31
32      min_out <= min_sig;
33      sec_out <= sec_sig;
34
35  end Behavioral;
```

10.4. sec_counter

Contador de minutos y segundos.

```VHDL
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity sec_counter is
6       port(
7               clk: in STD_LOGIC;
8               en: in STD_LOGIC;
9               rst: in STD_LOGIC;
10              min: out STD_LOGIC_VECTOR(5 downto 0);
11              sec: out STD_LOGIC_VECTOR(5 downto 0)
12      );
13  end sec_counter;
14
15  architecture Behavioral of sec_counter is
16
17      signal min_c, sec_c: STD_LOGIC_VECTOR(5 downto 0);
18
19  begin
20      process(clk, rst)
21      begin
22          if (rst = '0') then
23              min_c <= "000000";
24              sec_c <= "000000";
25          elsif (en = '1' and clk'event and clk = '1') then
26              if (sec_c >= "111011") then
27                  if (min_c >= "111011") then
28                      min_c <= "000000";
29                      sec_c <= "000000";
30                  else
31                      min_c <= std_logic_vector(unsigned(min_c)
                        + "000001");
```

```
                    sec_c <= "000000";
                end if;
            else
                sec_c <= std_logic_vector(unsigned(sec_c)
                + "000001");
            end if;
        end if;
    end process;

    min <= min_c;
    sec <= sec_c;

end Behavioral;
```

10.5. comparator

Comparador utilizado para comprobar si ha llegado la hora de la alarma.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comparator is
    port(
        min_1: in STD_LOGIC_VECTOR(5 downto 0);
        sec_1: in STD_LOGIC_VECTOR(5 downto 0);
        min_2: in STD_LOGIC_VECTOR(5 downto 0);
        sec_2: in STD_LOGIC_VECTOR(5 downto 0);
        en: in STD_LOGIC;
        comp_out: out STD_LOGIC
    );
end comparator;

architecture Behavioral of comparator is

begin
    comp_out <=
        '1' when en = '1' and min_1 = min_2 and sec_1 = sec_2
        else
        '0';
end Behavioral;
```

10.6. clock_system

Sistema completo del reloj.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clock_system is
    port(
        min_in: in STD_LOGIC_VECTOR(5 downto 0);
        sec_in: in STD_LOGIC_VECTOR(5 downto 0);
        save_alarm: in STD_LOGIC;
        start_stop: in STD_LOGIC;
        clk: in STD_LOGIC;
        rst: in STD_LOGiC;
        min_out: out STD_LOGIC_VECTOR(5 downto 0);
        sec_out: out STD_LOGiC_VeCTOR(5 downto 0);
        alarm: out STD_LOGIC
    );
end clock_system;

architecture structural of clock_system is

    component flipflop_t
        port(
            T, clk, rst: in STD_LOGIC;
            Q: out STD_LOGIC
        );
    end component;

    component prescaler
        port(
            clk, rst, en: in STD_LOGIC;
            pulse_1_sec: out STD_LOGIC
        );
    end component;

    component reg
        port(
            min_in, sec_in: in STD_LOGIC_VECTOR(5 downto 0);
            en, rst, clk: in STD_LOGIC;
            min_out, sec_out: out STD_LOGIC_VECTOR(5 downto 0)
        );
    end component;

```

```
42      component sec_counter
43          port(
44              clk, en, rst: in STD_LOGIC;
45              min, sec: out STD_LOGIC_VECTOR(5 downto 0)
46          );
47      end component;
48
49      component comparator
50          port(
51              min_1, sec_1, min_2, sec_2: in STD_LOGIC_VECTOR(5
                downto 0);
52              en: in STD_LOGIC;
53              comp_out: out STD_LOGIC
54          );
55      end component;
56
57      signal en, en1, en2, pulse_1_sec: STD_LOGIC;
58      signal min_1, sec_1, min_2, sec_2: STD_LOGIC_VECTOR(5 downto
        0);
59
60
61  begin
62      FFT : flipflop_t port map(
63          T => start_stop,
64          Q => en,
65          clk => clk,
66          rst => rst
67      );
68
69      PRS : prescaler port map(
70          clk => clk,
71          rst => rst,
72          en => en,
73          pulse_1_sec => pulse_1_sec
74      );
75
76      R : reg port map(
77          min_in => min_in,
78          sec_in => sec_in,
79          en => en1,
80          rst => rst,
```

```
81          clk => clk,
82          min_out => min_1,
83          sec_out => sec_1
84      );
85
86      SECC : sec_counter port map(
87          clk => clk,
88          en => en2,
89          rst => rst,
90          min => min_2,
91          sec => sec_2
92      );
93
94      COMP : comparator port map(
95          min_1 => min_1,
96          sec_1 => sec_1,
97          min_2 => min_2,
98          sec_2 => sec_2,
99          en => en,
100         comp_out => alarm
101     );
102
103     en1 <= en and save_alarm;
104     en2 <= en and pulse_1_sec;
105     min_out <= min_2;
106     sec_out<= sec_2;
107
108 end structural;
```

CAPÍTULO 11

Práctica 4

11.1. led_control

Sistema de luz-led para bicicletas con funcionamiento de autómata moore que maneja tres estados.

- Luz apagada
- Luz encendida fija
- Luz encendida intermitentemente (medio segundo encendida, medio segundo apagada)

```VHDL
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity led_control is
  port(
    clk : in STD_LOGIC;
    rst : in STD_LOGIC;
    puls : in STD_LOGIC;
    light : out STD_LOGIC
  );
end led_control;

architecture automata_moore of led_control is

  type modos_de_funcionamiento is (A, B, C1, C2, C3, C4, C5,
  C6);

  signal estado_actual : modos_de_funcionamiento := A;
  signal estado_siguiente : modos_de_funcionamiento := A;

begin

  -- lógica de estado siguiente, función F
  F: process (estado_actual, rst, puls)
  begin
```

```
25      case estado_actual is
26        when A =>
27          if (rst = '1' or puls = '0') then
28            estado_siguiente <= A;
29          else
30            estado_siguiente <= B;
31          end if;
32
33        when B =>
34          if (rst = '1') then
35            estado_siguiente <= A;
36          elsif (puls = '0') then
37            estado_siguiente <= B;
38          else
39            estado_siguiente <= C1;
40          end if;
41
42          when C1 =>
43          if (rst = '1' or puls = '1') then
44            estado_siguiente <= A;
45          else
46            estado_siguiente <= C2;
47          end if;
48
49          when C2 =>
50          if (rst = '1' or puls = '1') then
51            estado_siguiente <= A;
52          else
53            estado_siguiente <= C3;
54          end if;
55
56        when C3 =>
57          if (rst = '1' or puls = '1') then
58            estado_siguiente <= A;
59          else
60            estado_siguiente <= C4;
61          end if;
62
63        when C4 =>
64          if (rst = '1' or puls = '1') then
65            estado_siguiente <= A;
```

```
66          else
67            estado_siguiente <= C5;
68          end if;
69
70        when C5 =>
71          if (rst = '1' or puls = '1') then
72            estado_siguiente <= A;
73          else
74            estado_siguiente <= C6;
75          end if;
76
77        when C6 =>
78          if (rst = '1' or puls = '1') then
79            estado_siguiente <= A;
80          else
81            estado_siguiente <= C1;
82          end if;
83
84      end case;
85    end process;
86
87    -- registro
88    reg: process (clk)
89    begin
90      if (clk'event and clk = '1') then
91        estado_actual <= estado_siguiente;
92      end if;
93    end process;
94
95    -- lógica de salida, función G
96    with estado_actual select light <=
97      '1' when B,
98      '1' when C1,
99      '1' when C2,
100     '1' when C3,
101     '0' when others;
102
103 end automata_moore;
```