

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
DE TELECOMUNICACIONES



APUNTES DE VHDL

APUNTES DE VHDL DE LA ASIGNATURA ELECTRÓNICA DIGITAL

BRUNO MONTERO HERNANGÓMEZ

Índice

1. Introducción	4
2. Elementos básicos	4
3. Tipos en VHDL	5
4. Operadores de VHDL	9
5. Entity	11
6. Architecture	12
6.1. Objetos principales	13
6.2. Estilos de arquitectura	15
6.2.1. Elementos concurrentes	17
6.2.2. Procesos	19
6.2.3. Autómatas (máquinas de estados)	23
7. Simulación	26
8. Práctica 1	26
8.1. circuito1	26
8.2. circuito2	27
8.3. multiplexor	28
8.4. full_system	28
9. Práctica 2	31
9.1. CP	31
9.2. ROM	32
9.3. MUX8_8	33
9.4. MUX8_1	34
9.5. OPERA	35
10. Práctica 3	41
10.1. flipflop_t	41
10.2. prescaler	42
10.3. reg	44
10.4. sec_counter	45
10.5. comparator	47
10.6. clock_system	47

11. Práctica 4	51
11.1. led_control	51

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 parte importante de ser capaz de interpretar los lenguajes de descripción de hardware viene de la capacidad para distinguir la parte del circuito que es **concurrente** y la parte que es **secuencial**, pensar en **componentes** en vez de funciones y **cables y entradas o salidas** en vez de variables.

2. Elementos básicos

Un sistema digital se describe en dos partes:

1. Su **aspecto exterior**: Que consistirá en los puertos (típicamente entradas y salidas) considerando el sistema como una caja negra. Se definirá como la entidad (**entity**) del sistema.
2. Su **aspecto interior**: Que involucra el comportamiento interno del circuito. Se definirá como la arquitectura (**architecture**) del sistema.

La descripción de un sistema tendrá 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. Los paquetes incluidos por defecto (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:

```
1 library (nombre de la librería);  
2 use (nombre del paquete);
```

VHDL

- 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.

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 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 más significativo; si se hace de derecha a izquierda, el primer bit por la

3. Tipos en VHDL

Nombre de tipo	Escritura	Explicación
		izquierda será el más significativo.
<code>std_logic</code>	<code>'(bit real)'</code>	Admite los valores '0', '1', 'X', 'U', 'Z', 'W', 'L', 'H' o '-'
<code>std_logic_vector(rango)</code>	<code>"⟨cadena de bits reales⟩"⟨(descripción del rango)⟩</code>	Admite una cadena de bits reales con el rango especificado 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 más significativo; si se hace de derecha a izquierda, el primer bit por la izquierda será el más significativo.
<code>signed(rango)</code>	<code>"⟨cadena de bits reales⟩"⟨(descripción del rango)⟩</code>	Admite una cadena de bits reales con el rango especificado NB: La definición del rango no solo describe la longitud de la cadena, sino también su dirección. Si se define una cade-

Nombre de tipo	Escritura	Explicación
		na 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.
unsigned(rango)	"⟨cadena de bits reales⟩"(⟨descripción del rango⟩)	Admite una cadena de bits reales con el rango especificado 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.
boolean	⟨booleano⟩	Admite true o false

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

3. Tipos en VHDL

- **⟨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 rangoEl máximo valor definido es max
natural rango	⟨número natural⟩	Admite números naturales dentro del rangoEl máximo valor definido es max
positive rango	⟨número positivo⟩	Admite números positivos dentro del rangoEl máximo valor definido es max
real rango	⟨número real⟩	Admite números positivos dentro del rangoEl máximo valor definido es max
time	⟨duración⟩ ⟨unidad de tiempo⟩	Admite cualquier número con las siguientes unidades: 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.

```
1 (nuevo tipo)((objeto))
```

VHDL

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)'length`) y pueden manejar conversiones más complejas 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
<code>to_bit()</code>	Convierte <code>std_logic</code> a <code>bit</code>
<code>to_stdlogic()</code>	Convierte <code>bit</code> a <code>std_logic</code>
<code>to_bitvector()</code>	Convierte <code>std_logic_vector</code> a <code>bit_vector</code>
<code>to_stdlogicvector()</code>	Convierte <code>bit_vector</code> a <code>std_logic_vector</code>

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

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 constantrnes) del circuito independientemente de su arquitectura.

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

```

1  entity <nombre del sistema> is
2      generic(
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(
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
17     puerto genérico1>;
18     <identificador del puerto genérico2> : modo <tipo del
19     puerto genérico2>;
20
21     -- si hay más de un puerto del mismo modo con el mismo tipo
22     pueden definirse conjuntamente separando los identificadores
23     con comas
24
25 );
26 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), aunque solo utilizaremos los dos primeros.

NB: Los valores de entrada no se podrán modificar en el sistema y los de salida se podrán modificar, pero nunca se podrán evaluar para el desarrollo interno del sistema.

Los valores genéricos se suelen utilizar para introducir constantes que se utilizarán (se las referenciará) en la arquitectura del sistema.

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:

```
1 component {identificador del componente}
2     port(
3         {identificación del puerto1} : modo {tipo del puerto1};
4         {identificación del puerto2} : modo {tipo del puerto2};
5         ...;
6     );
7 end component;
```

VHDL

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

```
1 begin
```

VHDL

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:

```
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
```

VHDL

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

```
1 signal (identificador de la señal) : (tipo de la señal) :=
  (valor de inicio);
```

VHDL

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:

```
1 (identificador de la señal) <= (valor o expresión que toma)
```

VHDL

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:

```
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
```

VHDL

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:

```
1 variable (identificador de la variable) : (tipo de la
   variable);
```

VHDL

```
2 -- 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:

```
1 variable (identificador de la variable) : (tipo de la
   variable) := (valor de inicio);
```

VHDL

Para hacerse una asignación se utiliza:

```
1 (identificador de la variable) := (valor o expresión que toma);
```

VHDL

```
-- se puede hacer una asignación en su declaración haciendo variable
```

```
2 (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 de definición de arquitectura. En concreto veremos como utilizar el funcional, estructural y una combinación entre ambos.

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:

```
1 architecture <nombre de la arquitectura> of <identificador de la entidad> is
2 begin
3     -- conexión de puertos utilizando operaciones proporcionadas por VHDL
4 end <nombre de la arquitectura>;
```

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

```
1 architecture <nombre de la arquitectura> of <identificador de la entidad> is
2
3     component <identificador del componente1>
4         port(
5             <identificación del puerto1> : modo <tipo del puerto1>;
6             <identificación del puerto2> : modo <tipo del puerto2>;
7             ...;
8         );
9     end component;
10
11     component <identificador del componente2>
12         port(
13             <identificación del puerto1>: modo <tipo del puerto1>;
14             <identificación del puerto2>: modo <tipo del puerto2>;
15             ...;
16         );
17     end component;
```



```
18     ...
19
20     signal {identificador de la señal1} : {tipo de la señal1};
21     signal {identificador de la señal2} : {tipo de la señal2};
22     ...;
23
24 begin
25     -- operaciones proporcionadas por VHDL
26     -- asignaciones de señal directa
27     -- asignaciones de señal condicional
28     -- asignaciones de señal seleccionada
29     -- instanciación de componentes
30     -- sentencias de verificación
31     -- procesos
32
33 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 (\leq) Consisten en conectar señales utilizando el operador \leq .

```
1 {señal1} <= {señal2};
```

VHDL

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

```
1 {señal variable} <= {primera señal}, {segunda señal} after
{tiempo1}, {tercera señal} after {tiempo2}, ...;
```

VHDL

2. Asignación condicional (when-else) Consiste en conectar una señal a otras en base a unas condiciones.

```
1 (señal modificable) <=
2     (señal1) when (condición1) else
3     (señal2) when (condición2) else
4     (señal3) when (condición3) else
5     ...
6     (señaln);
7 -- si no se cumplen ninguna de las condiciones, se conectará a la
   (señaln)
```

VHDL

3. Asignación de señal seleccionada (with-select-when) Consiste en conectar una señal a otra en base al valor de una señal de selección.

```
1 with (señal para la selección) select (señal modificable) <=
2     (señal1) when (valor1 de la señal para la selección),
3     (señal2) when (valor2 de la señal para la selección),
4     (señal3) when (valor3 de la señal para la selección),
5     ...
6     (señaln) when others;
7 -- se conectará a la (señaln) para el resto de valores no cubiertos
   en la expresión
```

VHDL

4. Instanciación de componentes Consiste en conectar los puertos de los componentes.

```
1 (identificador del elemento) : (identificador del componente)
   port map(
2     (identificador del puerto1) => (señal1),
3     (identificador del puerto2) => (señal2),
4     ...
5 );
```

VHDL

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 Consiste en reportar mensajes para evaluar resultados.

```
1 assert((condición lógica))
2     report (string que mostrar por pantalla)
3     severity (nivel de severidad);
```

VHDL

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: En realidad las instrucciones se realizan de forma paralela (concurrente) y no estrictamente secuencial también en los procesos, sin embargo, si hay más de una sentencia que asigna cierta señal, perdurará solamente la última sentencia (lo que da esa noción de secuncialidad).

Las cláusulas de espera son:

Cláusula	Explicación
<code>wait</code>	Suspende el funcionamiento de la sentencia del proceso de forma indefinida
<code>wait on ((señales separadas por comas))</code>	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)
<code>wait until ((condición lógica))</code>	Suspende el funcionamiento de la sentencia del proceso hasta que la <condición lógica> sea cierta (true)
<code>wait for (time)</code>	Suspende el funcionamiento de la sentencia del proceso hasta que haya transcurrido el <tiempo> impuesto

```
1 (nombre opcional del proceso): process
2 begin
3     -- sentencias secuenciales
4     -- sentencias condicionales
5     -- clausulas de espera
6 end process;
```

VHDL

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.

```
1 (nombre opcional del proceso): process((lista de
    sensibilidad))
```

VHDL

```

2 begin
3     -- sentencias secuenciales
4     -- sentencias condicionales
5 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 los procesos que describen circuitos secuenciales es muy útil utilizar la expresión:

```
1 (señal)'event
```

VHDL

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:

- `rising_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).

Sentencias utilizadas en los procesos:

1. if-then-else

```

1 if (condición lógica) then
2     (sentencias)
3 ...
```

VHDL

```
4 elsif otra_condición then
5     (sentencias)
6 ...
7 else
8     (sentencias)
9 end if;
```

2. case-when

```
1 case (señal para la elección) is
2     when (valor1 de la señal para la selección) =>
3         (sentencias)
4     when (valor1 de la señal para la selección) =>
5         (sentencias)
6     ...
7     when others =>
8         (sentencias)
9 end case;
```

VHDL

3. for-loop

```
1 for (variable para el bucle) in (rango) loop
2     (sentencias)
3 end loop;
```

VHDL

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

```
1 while (condición lógica) loop
2     (sentencias)
```

VHDL

```
3 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 verán).

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:

```
1 type {identificador del enumerado} is ({identificador1},
   {identificador2}, ..., {identificadorn});
```

VHDL

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:

```
1 signal {identificador de la señal} : {tipo de la señal} :=
   {valor de inicio};
```

VHDL

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:

```
1  architecture (nombre de la arquitectura) of (identificador de
    la entidad) is VHDL
2
3      type (identificador del enumerado de estados) is ((identificador
    del estado1), (identificador del estado2), ..., (identificador
    del estadoN));
4
5      signal (identificador de la señal de estado actual) :
    (identificador del enumerado de estados) := (identificador del
    estadoM);
6
7      signal (identificador de la señal de estado siguiente) :
    (identificador del enumerado de estados) := (identificador del
    estadoM);
8
9      -- el estadoM es el estado de reposo
10
11     ...
12
13     begin
14
15         process (clk, (señal asíncrona1), ..., (señal asíncronaN))
16         begin
17             -- sentencias de asignación del estado actual al estado
18             siguiente
19         end process;
20
21         process ((identificador de la señal de estado actual),
22             (entrada1), ..., (entrada1))
23         begin
24             case (identificador de la señal de estado actual) is
25                 when (identificador del estado1) =>
26                     -- sentencias if-else en función de las entradas
```



```

23      ...
24      end case;
25  end process;
26
27  process ({identificador de la señal de estado actual},
28           {entrada1}, ..., {entrada1})
29      begin
30          case {identificador de la señal de estado siguiente} is
31              when {identificador del estado1} =>
32                  -- sentencias if-else en función de las entradas
33                  ...
34              end case;
35          end process;
36
37          -- si el autómata es de Moore, la lógica de salida no
38          -- dependería de las entradas, las entradas no estarían en la lista
39          -- de sensibilidad y no habría que recurrir a sentencias if-else de
40          -- comprobación de las entradas
41
42      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 la entidad como un autómata, esta debe incluir una **señal que haga de reset** y se debe **asignar valores al estado siguiente** siempre, (bajo cualquier condición, aunque pueda parecer redundante).

7. Simulación

Para comenzar la simulación:

```
1 Simulation
2 {seleccionar tb}
3 ISim Simulator
4 Simulate Behavioural Model
5 {click derecho}
6 Run all
```

Para reiniciar la simulación:

```
1 ISim> restart
```

HDL

y realizarla en un periodo concreto:

```
1 ISim> run (tiempo deseado)
```

HDL

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

```
1 {click sobre el cronograma}
2 zoom to full view
```

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

```
1 ISim> step
```

HDL

8. Práctica 1

8.1. circuito1

Módulo de sistema de operaciones lógicas.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
```

VHDL

```
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.

```
1  library IEEE;                                       VHDL
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity circuito2 is
5      port(
6          A : in STD_LOGIC;
7          B : in STD_LOGIC;
8          C : in STD_LOGIC;
9          F2 : out STD_LOGIC
10     );
11 end circuito2;
12
13 architecture Behavioral of circuito2 is
```

```
14 begin
15     F2 <= ((not(A) or B) and (A or not(B))) and (B or not(C));
16 end Behavioral;
```

8.3. multiplexor

Multiplexor 2x8.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity multiplexor is
5      port(
6          F1 : in STD_LOGIC;
7          F2 : in STD_LOGIC;
8          S  : in STD_LOGIC;
9          Z  : out STD_LOGIC
10     );
11 end multiplexor;
12
13 architecture Behavioral of multiplexor is
14 begin
15     with S select Z <=
16         F1 when '0',
17         F2 when '1',
18         '0' when others;
19 end Behavioral;
```

8.4. full_system

Sistema completo.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity full_system is
5      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
```

8. Práctica 1

```
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;
```

9. Práctica 2

9.1. CP

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

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity CP is
5      port(
6          e : in STD_LOGIC_VECTOR (7 downto 0);
7          y : out STD_LOGIC_VECTOR (2 downto 0);
8          idle : out STD_LOGIC
9      );
10 end CP;
11
12 architecture Behavioral of CP is
13
14 begin
15
16     y <=
17         "111" when e(7) = '1' else
18         "110" when e(6) = '1' else
19         "101" when e(5) = '1' else
20         "100" when e(4) = '1' else
21         "011" when e(3) = '1' else
22         "010" when e(2) = '1' else
23         "001" when e(1) = '1' else
24         "000";
```

VHDL

```
25     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 .

```
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.

```
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);
11         D5 : in STD_LOGIC_VECTOR (7 downto 0);
12         D6 : in STD_LOGIC_VECTOR (7 downto 0);
13         D7 : in STD_LOGIC_VECTOR (7 downto 0);
14         S : in STD_LOGIC_VECTOR (2 downto 0);
15         Y : out STD_LOGIC_VECTOR (7 downto 0)
16     );
17 end MUX8_8;
18
19 architecture Behavioral of MUX8_8 is
20
21 begin
22
23     with S select Y <=
24         D0 when "000",
25         D1 when "001",
26         D2 when "010",
```

```
27      D3 when "011",
28      D4 when "100",
29      D5 when "101",
30      D6 when "110",
31      D7 when "111";
32
33 end Behavioral;
```

9.4. MUX8_1

Multiplexor 8x1.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity MUX8_1 is
5      port(
6          D : in STD_LOGIC_VECTOR (7 downto 0);
7          S : in STD_LOGIC_VECTOR (2 downto 0);
8          Y : out STD_LOGIC
9      );
10 end MUX8_1;
11
12 architecture Behavioral of MUX8_1 is
13
14 begin
15
16     with S select Y <=
17         D(0) when "000",
18         D(1) when "001",
19         D(2) when "010",
```

VHDL

```

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	$\log_2(\text{parte entera de } A)$ (\log_2 equivale a realizar una codificación de prioridad)	"1" si ($\text{parte entera de } A \leq 0$), "0" en caso contrario
011	$-7.25 \times \log_2(\text{parte entera de } A)$ (la multiplicación se realiza con la ROM)	"1" si ($\text{parte entera de } A \leq 0$), "0" en caso contrario
100	$A / 2$ (equivale a desplazar los bits una posición)	siempre "0"

Código de control	Operación	Estado de la salida (error)
	a la derecha y extender el bit de signo)	
101	«11111111» si $B \leq A$, «00000000» en caso contrario (las comparaciones se pueden realizar con facilidad convirtiendo los vectores de bits en representación C2 a tipo signed)	siempre “0”
110	«11100111» si $-23.5 < B < +24.75$, «00011000» en caso contrario	siempre “0”
111	siempre «11001100»	siempre “0”

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity OPERA is
6      port(
7          A : in STD_LOGIC_VECTOR (7 downto 0);
8          B : in STD_LOGIC_VECTOR (7 downto 0);
9          control : in STD_LOGIC_VECTOR (2 downto 0);
10         resultado : out STD_LOGIC_VECTOR (7 downto 0);
11         error : out STD_LOGIC

```

```
12     );
13 end OPERA;
14
15 architecture mixed of OPERA is
16
17     component CP
18     port(
19         e : in STD_LOGIC_VECTOR (7 downto 0);
20         y : out STD_LOGIC_VECTOR (2 downto 0);
21         idle : out STD_LOGIC
22     );
23 end component;
24
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,

```

9. Práctica 2

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;
```

10. Práctica 3

10.1. flipflop_t

Biastable tipo T.

```
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;
```

VHDL

```
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
26                 Q_sig <= Q_ant;
27             end if;
28         end if;
29     end process;
30
31     Q_ant <= Q_sig;
32     Q <= Q_sig;
33
34 end Behavioral;
```

10.2. prescaler

Generador de pulsos a cada segundo.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
```

VHDL

```
4
5  entity prescaler is
6      port(
7          clk: in STD_LOGIC;
8          rst: in STD_LOGIC;
9          en: in STD_LOGIC;
10         pulse_1_sec: out STD_LOGIC
11     );
12 end prescaler;
13
14 architecture Behavioral of prescaler is
15
16     signal cont: STD_LOGIC_VECTOR(2 downto 0);
17
18 begin
19     process(clk, rst)
20     begin
21         if (rst = '0') then
22             cont <= "000";
23             pulse_1_sec <= '0';
24         elsif (en = '1' and clk'event and clk = '1') then
25             if (cont < "100") then
26                 cont <= std_logic_vector(unsigned(cont) + "001");
27                 pulse_1_sec <= '0';
28             else
29                 cont <= "000";
30                 pulse_1_sec <= '1';
31             end if;
32         end if;
```

```
33     end process;
34
35 end Behavioral;
```

10.3. reg

Registro de minutos y segundos para la alarma.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity reg is
5      port(
6          min_in: in STD_LOGIC_VECTOR(5 downto 0);
7          sec_in: in STD_LOGIC_VECTOR(5 downto 0);
8          en: in STD_LOGIC;
9          rst: in STD_LOGIC;
10         clk: in STD_LOGIC;
11         min_out: out STD_LOGIC_VECTOR(5 downto 0);
12         sec_out: out STD_LOGIC_VECTOR(5 downto 0)
13     );
14 end reg;
15
16 architecture Behavioral of reg is
17
18     signal min_sig, sec_sig: STD_LOGIC_VECTOR(5 downto 0);
19
20 begin
21     process(clk, rst)
22     begin
23         if (rst = '0') then
```

VHDL

```
24         min_sig <= "111111";
25         sec_sig <= "111111";
26         elsif (en = '1' and clk'event and clk = '1') then
27             min_sig <= min_in;
28             sec_sig <= sec_in;
29         end if;
30     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.

```
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
```

VHDL

```
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)
32                                             + "000001");
33                     sec_c <= "000000";
34                 end if;
35             else
36                 sec_c <= std_logic_vector(unsigned(sec_c) +
37                                             "000001");
38             end if;
39         end if;
40     end process;
41
42     min <= min_c;
43     sec <= sec_c;
```

```

42
43 end Behavioral;

```

10.5. comparator

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

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity comparator is
5      port(
6          min_1: in STD_LOGIC_VECTOR(5 downto 0);
7          sec_1: in STD_LOGIC_VECTOR(5 downto 0);
8          min_2: in STD_LOGIC_VECTOR(5 downto 0);
9          sec_2: in STD_LOGIC_VECTOR(5 downto 0);
10         en: in STD_LOGIC;
11         comp_out: out STD_LOGIC
12     );
13 end comparator;
14
15 architecture Behavioral of comparator is
16
17 begin
18     comp_out <=
19         '1' when en = '1' and min_1 = min_2 and sec_1 = sec_2 else
20         '0';
21 end Behavioral;

```

10.6. clock_system

Sistema completo del reloj.

1	library IEEE;	VHDL
2	use IEEE.STD_LOGIC_1164.ALL;	
3		
4	entity clock_system is	
5	port(
6	min_in: in STD_LOGIC_VECTOR(5 downto 0);	
7	sec_in: in STD_LOGIC_VECTOR(5 downto 0);	
8	save_alarm: in STD_LOGIC;	
9	start_stop: in STD_LOGIC;	
10	clk: in STD_LOGIC;	
11	rst: in STD_LOGIC;	
12	min_out: out STD_LOGIC_VECTOR(5 downto 0);	
13	sec_out: out STD_LOGIC_VECTOR(5 downto 0);	
14	alarm: out STD_LOGIC	
15);	
16	end clock_system;	
17		
18	architecture structural of clock_system is	
19		
20	component flipflop_t	
21	port(
22	T, clk, rst: in STD_LOGIC;	
23	Q: out STD_LOGIC	
24);	
25	end component;	
26		
27	component prescaler	
28	port(
29	clk, rst, en: in STD_LOGIC;	


```
30         pulse_1_sec: out STD_LOGIC
31     );
32     end component;
33
34     component reg
35     port(
36         min_in, sec_in: in STD_LOGIC_VECTOR(5 downto 0);
37         en, rst, clk: in STD_LOGIC;
38         min_out, sec_out: out STD_LOGIC_VECTOR(5 downto 0)
39     );
40     end component;
41
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
52             downto 0);
53         en: in STD_LOGIC;
54         comp_out: out STD_LOGIC
55     );
56     end component;
57
58     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;
```

11. Práctica 4

11.1. led_control

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

11. Práctica 4

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

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity led_control is
5      port(
6          clk : in STD_LOGIC;
7          rst : in STD_LOGIC;
8          puls : in STD_LOGIC;
9          light : out STD_LOGIC
10     );
11  end led_control;
12
13  architecture automata_moore of led_control is
14
15      type modos_de_funcionamiento is (A, B, C1, C2, C3, C4, C5, C6);
16
17      signal estado_actual : modos_de_funcionamiento := A;
18      signal estado_siguiete : modos_de_funcionamiento := A;
19
20  begin
21
22      -- lógica de estado siguiente, función F
23      F: process (estado_actual, rst, puls)
24      begin
25          case estado_actual is
```

VHDL

```
26      when A =>
27          if (rst = '1' or puls = '0') then
28              estado_siguiete <= A;
29          else
30              estado_siguiete <= B;
31          end if;
32
33      when B =>
34          if (rst = '1') then
35              estado_siguiete <= A;
36          elsif (puls = '0') then
37              estado_siguiete <= B;
38          else
39              estado_siguiete <= C1;
40          end if;
41
42      when C1 =>
43          if (rst = '1' or puls = '1') then
44              estado_siguiete <= A;
45          else
46              estado_siguiete <= C2;
47          end if;
48
49      when C2 =>
50          if (rst = '1' or puls = '1') then
51              estado_siguiete <= A;
52          else
53              estado_siguiete <= C3;
54          end if;
```

11. Práctica 4

```
55
56     when C3 =>
57         if (rst = '1' or puls = '1') then
58             estado_siguiiente <= A;
59         else
60             estado_siguiiente <= C4;
61         end if;
62
63     when C4 =>
64         if (rst = '1' or puls = '1') then
65             estado_siguiiente <= A;
66         else
67             estado_siguiiente <= C5;
68         end if;
69
70     when C5 =>
71         if (rst = '1' or puls = '1') then
72             estado_siguiiente <= A;
73         else
74             estado_siguiiente <= C6;
75         end if;
76
77     when C6 =>
78         if (rst = '1' or puls = '1') then
79             estado_siguiiente <= A;
80         else
81             estado_siguiiente <= 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;
```