Práctica 1

Multiplicador secuencial de números sin signo codificados en binario utilizando el algoritmo de "lápiz y papel".

La implementación de un multiplicador con una estructura puramente combinacional tiene la ventaja de que proporciona una gran velocidad de cómputo. El inconveniente sin embargo está en que el tamaño del circuito crece rápidamente con el número de bits de los operandos (el número de celdas es igual a $2^m \times 2^q$, siendo m y q el tamaño del multiplicando y multiplicador respectivamente). Para solventar este problema puede utilizarse otra estrategia que utiliza una estructura secuencial, en la que el producto se va obteniendo en varias etapas. Esta aproximación resulta en un circuito más lento, pero con un tamaño mucho más reducido, que resulta ventajosa cuando el número de bits de los operandos es elevado.

Para recordar el funcionamiento del algoritmo de "lápiz y papel" se considerará la multiplicación de dos números de 4 bits expresados en binario. El procedimiento se muestra en la Figura 1 para el caso de 13×10: se examinan los bits del multiplicador comenzando por el bit menos significativo, cuando su valor es '1' se copia el multiplicando convenientemente desplazado según el bit examinado para sumarlo posteriormente con los restantes resultados parciales, cuando vale '0' se copia una fila de ceros.

```
    1101  → 13
  × 1010  → 10
  ------
    0000
   1101
  0000
 1101
--------
 10000010 → 130
```

Figura 1: Algoritmo clásico de "lápiz y papel" para multiplicación de dos números.

Para implementar por hardware este algoritmo se pueden realizar algunos cambios que simplificarán el diseño:

- Se pueden sumar los resultados parciales dos a dos, en lugar de esperar hasta obtenerlos todos. De esta forma se necesitan tan solo un sumador de dos operandos (en lugar de q) y un registro donde almacenar el resultado parcial cada paso, lo que supone un gran ahorro en circuitería. De hecho, la operación de suma no será necesaria en los pasos en los que el bit del multiplicando es '0', ya que sólo se sumaría una fila de ceros al producto parcial acumulado.
- El hecho de desplazar el multiplicando a la izquierda en cada paso implica que se necesita un sumador de $m \times q$ bits, que es el tamaño final del producto. En lugar de ésto, se propone desplazar el producto parcial acumulado a la derecha, con lo que se puede realizar la suma de únicamente los m bits más significativos del producto parcial, puesto que el resto quedan invariantes. En el caso usual de $m = q$, el tamaño del sumador se reduce a la mitad.

Un ejemplo del esta versión hardware del algoritmo se muestra en la Figura 2. En cada uno de los cuatro pasos, se indica si se debe sumar ($+$) y desplazar el resultado ($\hookrightarrow$), o únicamente desplazar, cuando la suma no es necesaria porque el bit del multiplicador es '0'. El resultado que se debe almacenar al final de cada paso para tenerlo en cuenta en los siguientes (producto parcial acumulado) se ha recuadrado.

Dado que este problema es suficientemente complejo, se utilizará una estrategia de diseño a nivel de transferencia de registros (RTL, *Register Transfer Level*), dividiendo el circuito en una Unidad de Procesamiento (UP) o ruta de datos, y una Unidad de Control (UC). La UP estará compuesta por los *elementos de memoria* para almacenar los operandos y los resultados, los *recursos de cálculo* o conjunto de operadores que procesan los datos, y finalmente los *recursos de conexionado*, que determinan las posibles rutas de los datos entre los elementos de memoria y los recursos de cálculo.

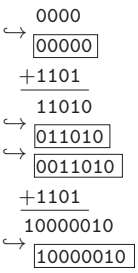

Figura 2: Algoritmo hardware de "lápiz y papel" para multiplicación de dos números.

La UC determina el funcionamiento de la UP. Para ello recibe información en cada instante sobre los resultados de las operaciones anteriores y el estado del sistema, y en función de ésta información y del algoritmo que deba implementar, genera una serie de instrucciones que controlan los recursos de la ruta de datos, además de determinar cual debe ser el siguiente paso a ejecutar del algoritmo en curso. La UC es un circuito secuencial típico, o Máquina de Estados Finitos (FSM, *Finite State Machine*) que puede implementarse utilizando una estrategia basada en Lógica Cableada o en Lógica Microprogramada.

Para la implementación de la UP de nuestro multiplicador secuencial se necesitará a primera vista un registro M de m bits para el multiplicando, otro Q de q bits para el multiplicador, un sumador de m bits y un registro acumulador A de $m + q$ bits donde se almacenará finalmente el producto. Esta estrategia resulta ineficiente, porque al comienzo los productos parciales no ocuparán todo el registro A, mientras en el registro Q ocurre lo contrario, en cada paso se libera espacio al desplazar hasta que al final queda a cero. Dado que el ritmo de crecimiento de los productos parciales es igual al de decremento del multiplicador, es posible utilizar este hecho para compartir espacio y reducir el tamaño de A. De esta forma, se puede definir A de m bits, los necesarios para almacenar los resultados del sumador, mientras que los q bits restantes se van almacenando en el espacio que va quedando libre en el registro Q. A y Q se comportan de esta forma como un único registro que se desplaza a la derecha en cada paso del algoritmo. En el paso final, A y Q almacenarán los m bits más significativos y los q bits menos significativos respectivamente del producto.

Utilizando esta estrategia, se pueden resumir los recursos que utilizará nuestra UP:

- Un registro M de m bits para almacenar el multiplicando, con entrada de habilitación de carga weM.
- Un registro Q de q bits para almacenar el multiplicador, con entrada de habilitación de carga weQ. Éste además deberá tener capacidad de desplazamiento a la derecha, habilitado por la entrada de control shQ.
- Un registro acumulador A de m bits con control de carga en paralelo ldA y de desplazamiento shA. Además deberá tener una entrada de puesta a cero $initA$ que se deberá activar al comienzo de una nueva operación de multiplicación, para eliminar el resultado de la anterior.
- Un sumador binario de números de m bits.
- Un registro de un único bit C que almacenará el acarreo de salida del sumador, en caso de que lo hubiese. Este registro es necesario porque al desplazar A y Q a la derecha, el bit más significativo de A debe cargarse con el valor del acarreo de salida del sumador en caso de que este sea '1', y que por lo tanto se deberá tener almacenado en C. De esta forma los registros C, A y Q se comportarán como un único registro de desplazamiento monolítico. Al igual que A, el registro C necesitará una entrada de control de la puesta a cero ($initC$), una entrada de habilitación de carga (ldC) y otra para el desplazamiento (shC).
- Un contador descendente CNT de n bits para determinar el número de pasos de la multiplicación, por lo que deberá verificar que $n \geq \lceil \log_2 q \rceil$, siendo $\lceil \log_2 q \rceil$ el menor número entero mayor que $\log_2 q$. Inicializará con el valor $q-1$, e indicará el final del algoritmo de multiplicación cuando llegue a cero. Necesitará por tanto una entrada de control de inicialización $initCNT$ y una entrada de habilitación de cuenta $shCNT$.
- Relacionado con el anterior, un circuito de detección de cero Z, con n entradas y una salida que se activará en el último paso de la multiplicación, indicando a la UC el fin del algoritmo.

La estructura de este multiplicador se muestra en la Figura 3. En ella se incluye también un bloque para representar a la UC. Se detallan las señales de control necesarias para cada unidad (a excepción de la señal de reset global rst y la señal de reloj clk, que son comunes a todos los elementos secuenciales).

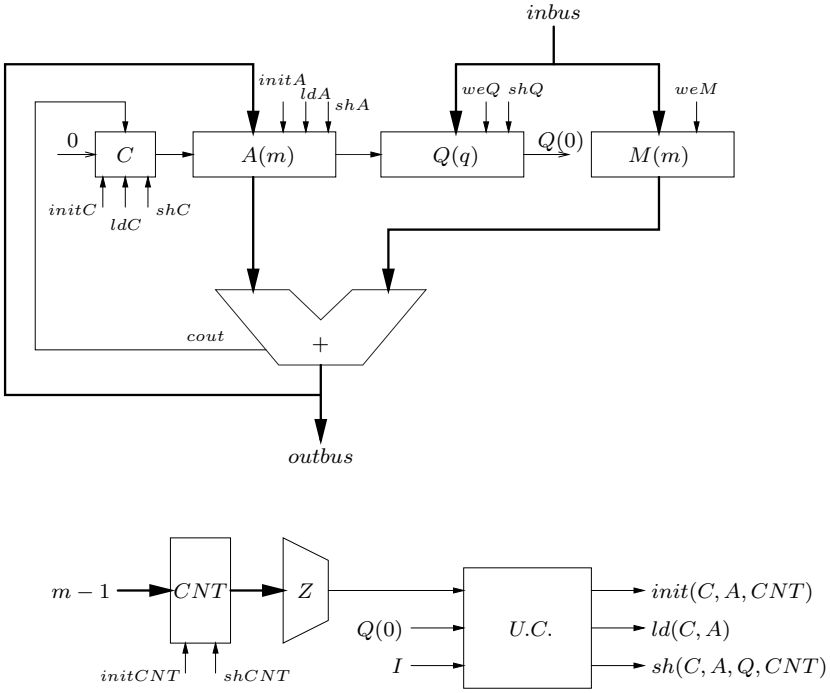

Figura 3: Estructura del multiplicador secuencial.

La UC del circuito necesitará como entrada una señal de inicio, I, proporcionada conceptualmente por un dispositivo externo tras haber cargado multiplicando y multiplicador en los registros M y Q respectivamente, para indicar a la UC que ya puede comenzar a ejecutar el algoritmo de multiplicación. La UC necesitará también como entrada el bit menos significativo del multiplicador $Q(0)$ en cada paso del algoritmo, para determinar si se debe sumar M y desplazar el producto parcial resultante, o sólo desplazar el producto parcial anterior. Finalmente, una tercera entrada Z proveniente del detector de cero le indicará cuando ha finalizado la multiplicación y debe volver al estado de reposo.

En cuanto a las salidas, se necesitarán tantas como señales de control se han definido para la UP. En el Cuadro 1 se muestran estas nueve señales agrupadas por componente de la ruta de datos. Debe recordarse que las señales weQ y weM no son proporcionadas por la UC.

Un sistema compuesto de una UP y una UC, como nuestro multiplicador, se describe de forma clara y concisa utilizando un diagrama de Máquina de Estado Algorítmica, o diagrama ASM (*Algorithmic State Machine*). En el diagrama ASM, la UP se representa indicando las operaciones a nivel RTL que debe realizar la ruta de datos en cada estado del algoritmo. La UC viene representada por la propia estructura del diagrama, que indica los diferentes estados de la UC, las decisiones que deben tomarse para determinar el estado siguiente a un estado dado, y las salidas de control que se deben generar en cada uno de ellos para controlar las operaciones de la ruta de datos. El diagrama ASM añade además la ventaja de que permite representar algunas salidas como tipo Mealy y otras como tipo Moore dentro de la misma máquina de estados de la UC, a diferencia de las representaciones tradicionales de FSM con Diagramas de Estados.

El diagrama ASM del multiplicador secuencial se muestra en la Figura 4, y consta de tres estados. Inicialmente se supondrá que el multiplicando y el multiplicador son almacenados en los registros M y Q de forma externa (como se explicó anteriormente). Mientras, el multiplicador se encuentra en un estado de espera $INICIO$, en el que no realiza ninguna operación mientras que la señal de inicio I valga '0'. La multiplicación comienza cuando el dispositivo externo pone I a '1'. En ese momento, la UC debe inicializar la ruta de datos, poniendo a '0' los registros C y A y cargando $n-1$ en el contador. Además, se define $SUM\&ACU$ como estado siguiente, en lugar de $INICIO$. Esto se corresponde con una estructura tipo Mealy, por lo que las acciones/salidas no se muestran

Componente	Operación RTL	Señal de Control
Registro M	$M \leftarrow inbus$	–
Registro Q	$Q \leftarrow inbus$	–
	$RShift(Q)$	shQ
Registro A	$A \leftarrow 0$	$initA$
	$A \leftarrow A + M$	ldA
	$RShift(A)$	shA
Registro C	$C \leftarrow 0$	$initC$
	$C \leftarrow C_{out}$	ldC
	$RShift(C)$	shC
Contador	$CNT \leftarrow n - 1$	$initCNT$
	$CNT \leftarrow CNT - 1$	$shCNT$

Cuadro 1: Operaciones RTL y señales de control correspondientes para el multiplicador secuencial.

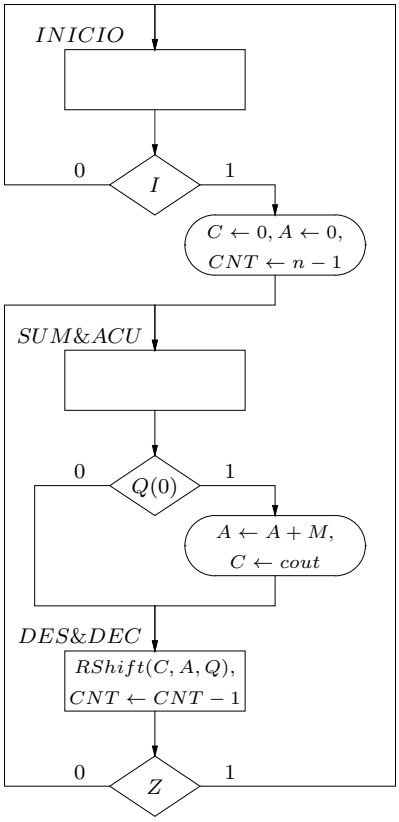

Figura 4: Diagrama ASM del multiplicador secuencial.

en el Bloque del Estado, sino en el Bloque Condicional situado tras el Bloque de Decisión.

En el estado $SUM\&ACU$ se efectúa una decisión en función del valor del bit menos significativo del multiplicador. Si $Q(0)$ vale '1' se suma el contenido de los registros A y M y el resultado se almacena de nuevo en A (además, el acarreo saliente del sumador se almacena en C). Si $Q(0)$ es '0', no se modifican los contenidos de A y C. Finalmente, en ambos casos se define $DES\&DEC$ como estado siguiente. De nuevo se observa como las acciones que realiza la ruta de datos son condicionales, por lo que las salidas de la UC que deben forzar la ejecución de esas acciones serán de tipo Mealy.

En el estado $DES\&DEC$ se realiza un desplazamiento a la derecha de los registros C, A y Q de forma

conjunta, como si formasen un único registro de $1+m+q$ bits. De igual forma, se debe decrementar el contador, que indica el número de bits del multiplicador que faltan por procesar. Las acciones a realizar en este estado, a diferencia de los anteriores, no dependen de las variables de entrada, por lo que se definen dentro del bloque de estado, y las salidas de control asociadas serán salidas de Moore.

Finalmente, en $DES\&DEC$ se comprueba el valor del detector de cero, que en las primeras $n-1$ ocasiones será '0'. En ese caso, el algoritmo no ha finalizado, y el estado siguiente será $SUM\&ACU$ para procesar el siguiente bit. La siguiente vez que se examina Z valdrá '1', puesto que el contador ha llegado a '0', lo que indica el fin del algoritmo y el estado siguiente será el estado de reposo $INICIO$.

Cabe destacar que aunque pueda parecer que el decremento del contador se realiza antes de comprobar el detector de cero, el nuevo valor del contador decrementado no se actualiza hasta el siguiente pulso de reloj (cuando el sistema pasa al estado $SUM\&ACU$ o $INICIO$). Por tanto, durante el periodo de reloj en el que el sistema se encuentra en el estado $DES\&DEC$ en el que se realiza la comprobación de Z, el contador todavía mantiene su valor anterior.

En este punto la estructura de la UP está totalmente determinada, sólo queda definir la UC. Si se utiliza una estrategia de diseño cableado, la UC debe definirse como una FSM, utilizando un Diagrama o Tabla de Estados, que pueden obtenerse directamente del diagrama ASM. Para ello no hay más que considerar los mismos estados y estructura de interconexión entre ellos que en el diagrama ASM, eliminar la información correspondiente a las microoperaciones a realizar en cada estado y sustituirla por las salidas correspondientes de la unidad de control. En nuestro caso se obtiene un Diagrama de Estados de Mealy, puesto que existen salidas condicionales, como se muestra en la Figura 5. Para obtener un autómata de Moore se debería modificar el diagrama ASM para que no contuviera bloques condicionales, aunque ello significaría un aumento del número de estados.

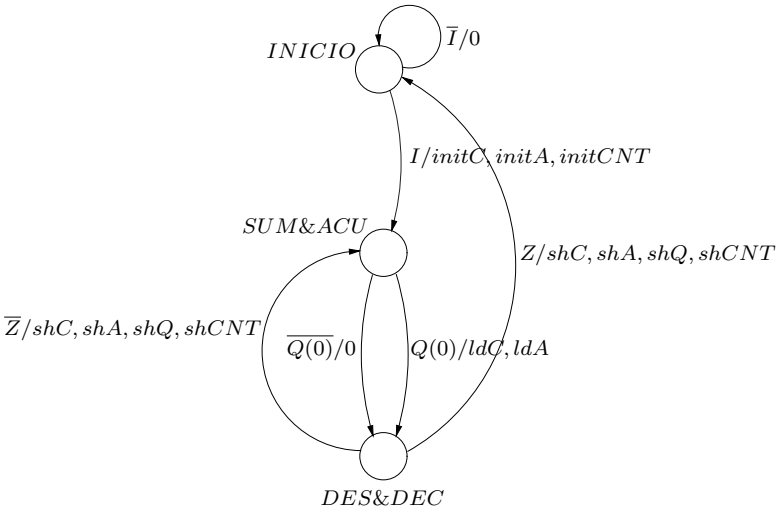

Figura 5: Diagrama de Estados de la UC del multiplicador secuencial.

A la vista del diagrama de estados se pueden hacer algunas simplificaciones para reducir el número de señales de control. Por ejemplo, la inicialización de los registros C y A y del contador CNT sólo se realiza en el estado $INICIO$ y en ningún otro, por lo que se pueden sustituir las tres salidas de inicialización por una sola común, $init$. De igual forma, la carga de los registros C y A se produce únicamente en el estado $SUM\&ACU$, por lo que se puede utilizar la misma señal para ambos, ld. Finalmente, en el estado $DES\&DEC$ se deben desplazar C, A y Q, y decrementar el contador simultáneamente, operaciones que siempre se realizan conjuntamente, por lo que se pueden sustituir las cuatro salidas por una sóla, sh. El diagrama de estados simplificado se muestra en la Figura 6.

A partir del Diagrama de Estados simplificado de la UC, puede obtenerse un circuito a nivel de puertas y flip–flops que implemente la máquina de estados utilizando las técnicas tradicionales (implementación mínima, con un elemento de memoria por estado, con una ROM y un registro, con un registro de secuencia y un decodificador, con un contador y un decodificador, etc.). Sin embargo, en nuestro caso se utilizará una descripción VHDL de alto nivel, que se obtiene directamente a partir del diagrama de estados de la máquina, lo que evitará la necesidad de recurrir a una especificación a nivel de puertas.

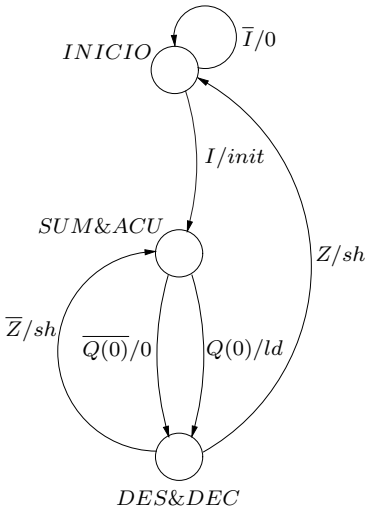

Figura 6: Diagrama de Estados simplificado de la UC del multiplicador secuencial.

La descripción VHDL del multiplicador secuencial se muestra en el Listado 1. La declaración de la entidad incluye los puertos *inbus* y *outbus* para los datos de entrada (multiplicando y multiplicador) y de salida (producto), la señal de inicio I, las entradas de habilitación de carga de los registros M y Q y las entradas de reset y reloj (rst y clk respectivamente).

Dentro de la arquitectura, las líneas 17 a 24 constituyen la parte declarativa, donde se definen los tipos de datos y las señales intermedias que se necesitarán para conectar los diversos componentes del multiplicador, y cuyo significado se explicará al analizar cada uno de los módulos.

En el cuerpo de la arquitectura (después del *begin*) se definen los componentes a nivel RTL que se han definido en la Figura 3. El primer proceso (líneas 28–42) define los registros C y A de forma conjunta, utilizando para ello la señal CA. Se ha tomado esta decisión puesto que ambos registros tienen las mismas señales de control (*init*, *ld* y *sh*) y el mismo comportamiento, que es diferente sin embargo en el caso de los registros Q y M. Su comportamiento es el siguiente: en cada flanco de reloj, si *init* está activo, CA se inicializa a '0'; en caso contrario, si *ld* está activo, se carga en ambos registros el valor en *pp*, el producto parcial proveniente del sumador; finalmente, si *ld* no está activo se comprueba si *sh* lo está, en cuyo caso se produce un desplazamiento a la derecha de CA.

El proceso siguiente (44–56) define al registro Q de forma similar. Sin embargo, en este caso no existe la entrada *init* de puesta a cero, y la entrada de habilitación de carga *weQ* es una entrada externa, no gobernada por la Unidad de Control. La señal *sh* controla el desplazamiento, que ahora se realizará cargando $CA(0)$ como bit más significativo de Q, para conseguir el desplazamiento conjunto de ambos registros.

A continuación se define el registro M para el multiplicando. Este es el más sencillo de todos, puesto que únicamente tiene una entrada de control, *weM* para la carga desde el bus de datos del valor correspondiente.

El sumador está descrito de forma muy compacta en una sola línea (71), y por ello precisa cierta aclaración. En primer lugar, dado que es necesario disponer del bit de acarreo saliente del sumador, se deben expandir los operandos A y M de 8 a 9 bits, concatenándoles un '0' como bit más significativo a ambos. De esta forma en la señal de salida del sumador *pp*, también de 9 bits, se obtiene la suma (en los 8 bits menos significativos) y el acarreo saliente si lo hubiese (en $pp(8)$), para almacenarlos en el registro CA. En segundo lugar, los operandos expandidos deben ser convertidos a un tipo de datos que soporte las operaciones aritméticas, como el UNSIGNED antes de realizar la suma, y posteriormente el resultado debe ser de nuevo convertido a STD_LOGIC_VECTOR para poder asignarlo a la señal *pp* que es de este tipo de datos.

La línea 74 define la salida, formada por los 8 bits menos significativos de CA y los 8 bits de Q, que es asignada al puerto *outbus*.

El proceso siguiente (77–88) define un contador descendente con inicialización a "111" y cuenta descendente controlada por *sh*.

Finalmente se describe la Unidad de Control del multiplicador, en forma de máquina de estados finitos, utilizando dos procesos. Los estilos de descripción de FSM son muy variados, aunque lo más normal, por la facilidad que representa para el software de síntesis a la hora de reconocer el tipo de sistema, es dividir el comportamiento en dos procesos concurrentes. El primer proceso se encargará de la parte secuencial, infiriendo el Registro de Estado que describe las asignaciones del estado siguiente al estado actual, sincronizadas con el flanco de reloj. Se utiliza un segundo y único proceso combinacional para describir la función que calcula el Estado Siguiente y las Salidas correspondientes para cada uno de los estados.

El código mostrado resulta suficientemente explicativo para comprender esta técnica. En el proceso combinacional, las salidas que sólo dependen del Estado Actual serán salidas de Moore, mientras que las que dependen además del valor de las entradas (I, $Q(0)$ o Z en nuestro caso) serán salidas de Mealy. De esta forma, VHDL permite definir fácilmente máquinas de estados de arquitectura mixta (Moore/Mealy). En caso de que las salidas no dependan de las entradas, sino que sean sólo función del estado actual, deberán situarse fuera de la sentencia IF, quedando únicamente dentro de éste la lógica de decisión que determina el estado siguiente.

Listado 1: Multiplicador secuencial según el algoritmo de "lápiz y papel".

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity mult_sec is
6      Port ( inbus  : in std_logic_vector(7 downto 0);
7             outbus : out std_logic_vector(15 downto 0);
8             I      : in std_logic;
9             weQ    : in std_logic;
10            weM    : in std_logic;
11            rst    : in std_logic;
12            clk    : in std_logic);
13 end mult_sec;
14
15 architecture Behavioral of mult_sec is
16
17     type type_state is (INICIO, SUMACU, DESDEC);
18     signal state, nextstate: type_state;
19
20     signal CA, pp       : std_logic_vector(8 downto 0);
21     signal Q, M         : std_logic_vector(7 downto 0);
22     signal init, ld, sh: std_logic;
23     signal CNT          : unsigned (2 downto 0);
24     signal Z            : std_logic;
25
26 begin
27
28     --registros C y A:
29     process(rst, clk)
30     begin
31         if (rst='1') then
32             CA <= (others=>'0');
33         elsif rising_edge(clk) then
34             if (init='1') then
35                 CA <= (others=>'0');
36             elsif (ld='1') then
37                 CA <= pp;
38             elsif (sh='1') then
39                 CA <= '0' & CA(8 downto 1);
40             end if;
41         end if;
42     end process;
43
44     --registro Q:
45         --
46         --
47         --
48         --
49         --
50         --
51         --
52         --
```

```
53          --
54          --
55          --
56          --
57
58      --registro M:
59          --
60          --
61          --
62          --
63          --
64          --
65          --
66          --
67          --
68          --
69
70      --sumador:
71      pp <=std_logic_vector(unsigned('0'&CA(7 downto 0))+unsigned('0'&M));
72
73      --salida:
74      outbus <= CA(7 downto 0) & Q;
75
76      --contador:
77          --
78          --
79          --
80          --
81          --
82          --
83          --
84          --
85          --
86          --
87          --
88          --
89
90      --detector de cero:
91      process(cnt)
92      begin
93          if (CNT="000") then
94              Z <= '1';
95          else
96              Z <= '0';
97          end if;
98      end process;
99
100     --Unidad de Control:
101     process(--)
102     begin
103         --
104         --
105         --
106         --
107         --
108     end process;
109
110     process(--)
111     begin
112         init <= '0'; ld <= '0'; sh <= '0';
113         case state is
114             when INICIO =>
115                 --
116                 --
117                 --
118                 --
119                 --
120                 --
121                 --
122             when SUMACU =>
123                 --
```

```
124                     --
125                     --
126                     --
127                     --
128                     --
129                 when DESDEC =>
130                     --
131                     --
132                     --
133                     --
134                     --
135                     --
136             end case;
137         end process;
138
139     end Behavioral;
```

Los estados posibles se listan en un tipo enumerado para mayor legibilidad, aunque se podría haberles asignado una señal codificada del tipo STD_LOGIC_VECTOR. En el primer caso, al no especificar el tipo de codificación que se debe utilizar para asignar una combinación de valores de salida de cada biestable a cada estado, las herramientas de síntesis suelen colocarse en modo "automático" y utilizar la codificación (binaria, gray, *one–hot*, etc.) más adecuada para optimizar el circuito según los parámetros especificados (menor área o mayor velocidad, usualmente). Además de mayor legibilidad y automatización del proceso de síntesis, la utilización de un tipo enumerado para definir el conjunto de estados tiene la ventaja de que en la sentencia CASE no es necesario utilizar la cláusula WHEN OTHERS, ya que la señal cuyo valor se comprueba, *state*, sólo puede tomar los valores especificados ($INICIO$, $SUM\&ACU$ o $DES\&DEC$) por definición.