Práctica I

Modelado y simulación de una máquina expendedora de refrescos

1.1. Objetivos.

En esta práctica vamos a modelar el comportamiento de un sistema descrito desde el nivel de transferencia de registros, a través de su diagrama ASM.

1.2. Ejemplo.

En primer lugar vamos a realizar un ejemplo del proceso completo. Este ejemplo será el sistema de control de una máquina expendedora de refrescos. Dicha áquina contará con las siguientes características:

- Únicamente dispone de un tipo de refresco, con un precio total de 32 unidades.
- Puede admitir monedas por un valor mínimo de 1 unidad, a través del puerto *moneda*.
- Las monedas son detectadas cuando se produce un cambio en una señal, denominada *moneda_ins*.
- Tiene la capacidad de devolver cambio, con un valor máximo de almacenamiento de 256 unidades.
- Dispone de un señal, denominada *botón*, para solicitar la expulsión del refresco y la devolución del cambio.
- Solamente se expulsará el refresco y devolverá el cambio (cuando la señal *refresco* tome el valor '1', y se devolverá la cantidad almacenada en *vuelta*), cuando se halla introducido una cantidad superior o igual al precio del refresco.
- Se tratará de un sistema síncrono, por lo que existirá una señal de reloj llamada *clk*.

El diagrama ASM corresopndiente a dicho sistema se muestra en la figura 1.1. En ella podemos apreciar tres bloques ASM (que se ejecutarán en ciclos de reloj diferentes) con las tres operaciones básicas que debe realizar el sistema:

- Almacenar la cantidad
- Obtener el valor de la vuelta
- En función de dicha vuelta, expulsar el refresco.

Para incluir este diagrama en el software de XILINX, hay que seguir los pasos comentados en las prácticas iniciales, es decir

- Crear un nuevo proyecto
- Añadir una nueva fuente, la cual tendrá los siguientes puertos:
  - Puerto de entrada clk
  - Puerto de entrada botón
  - Puerto de entrada moneda_ins
  - Bus de entrada moneda, con un tamaño de 8 bits (correspondiente al máximo de 256 unidades)
  - Puerto de salida refresco

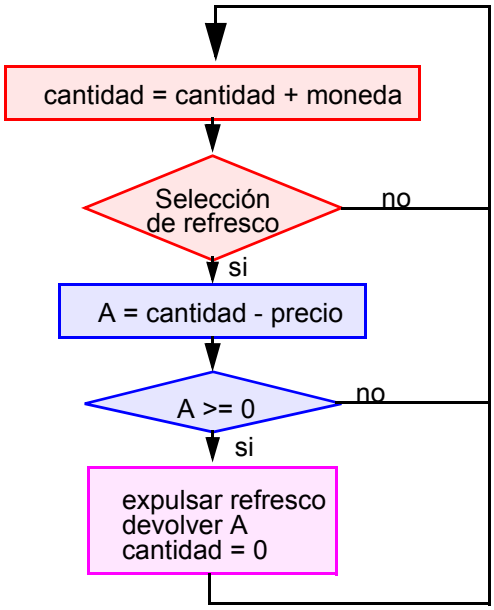

Figura 1.1.- Diagrama ASM correspondiente a la máquina expendedora de refrescos.

- Bus de salida vuelta, con un tamaño de 8 bits (correspondiente al máximo de 256 unidades).

- Crear un testbench para poder realizar las simulaciones de comprobación.
- Sintetizar el sistema.
- Implementar el sistema.

El código VHDL del sistema se muestra a continuación (en el que se han eliminado los comentarios que introduce el software por defecto; únicamente se han dejado los comentarios aclaratorios del código, en rojo):

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

entity refresco is
   Port ( clk, boton, moneda_ins : in  STD_LOGIC;
          moneda : in  STD_LOGIC_VECTOR (7 downto 0);
          refresco : out  STD_LOGIC;
          vuelta : out  STD_LOGIC_VECTOR (7 downto 0));
end refresco;

architecture Behavioral of refresco is
    type estado is (almacenamiento, seleccion, expulsion);
    signal ep, pe: estado;
-- En esta parte declarativa se han definido los tres estados por los que va a pasar el sistema
-- (correspondiente a las tres operaciones antes mencionadas). Estos estados han sido denominados:
-- almacenamiento, selección y expulsión (salvo por los acentos)
-- De la mismo forma, también han sido declaradas las señales del estado presente (ep) y próximo
-- estado (pe).
    signal cantidad: std_logic_vector(7 downto 0):="00000000";
-- Ha sido necesario declarar un bus interno para almacenar las cantidades insertadas en la máquina,
```

```
-- Dicho bus ha sido denominado cantidad.
    constant precio: std_logic_vector(7 downto 0):="00100000";
-- Por último, se ha incluido el precio del refresco como una constante con un valor igual a 32, es decir
-- a 00100000 en binario.
begin
-- Un proceso es el elemento VHDL para introducir instrucciones secuenciales, y por lo tanto,
-- poder describir algoritmos. Puede tener una lista de sensibilidad (lista de señales entre paréntesis
-- separadas por “,”), que al cambiar alguna de sus componentes provocará la ejecución del proceso.
-- Tendrá su parte declarativa (local al proceso) y su cuerpo.
--
-- Este proceso se ejecutará cada vez que cambie alguna de las siguientes señales: moneda_ins (inserción
-- de monedas), boton (solicitud de refrescos) o ep (estado presente del sistema)
-- En función del estado presente, deberá realizar las operaciones indicadas en cada estado.
    P1:process(moneda_ins, boton, ep)
-- Parte declarativa
    begin
-- Cuerpo
            case ep is
            when almacenamiento =>
-- A la hora de dar valores a las señales hay que diferenciar entre valores simples (que van entre
-- comillas simples ‘) y buses (que van entre comillas dobles “)
                    refresco <= '0';
                    vuelta <= "00000000";
                    cantidad <= cantidad+moneda;
-- Habiendo cargado las librerías adecuadas podemos hacer uso de funciones aritméticas.
-- Dicha librería es IEEE.STD_LOGIC_ARITH.all, la cual fue cargada al principio del código.
                    if (boton = '1') then pe<=seleccion;
                    end if;
            when seleccion =>
                    vuelta <= cantidad-precio;
-- Para evitar problemas con los números negativos, se ha optado por la comparación entre la
-- cantidad almacenada y el precio, en lugar de la vuelta con el ‘0’.
                    if (cantidad >= precio) then pe <=expulsion;
                    else pe <= almacenamiento;
                    end if;
            when expulsion =>
                    refresco <= '1';
                    cantidad <= "00000000";
                    pe <= almacenamiento;
            end case;
    end process;
-- Con este proceso, actualizaremos el estado cuando suba la señal de reloj (clk), por lo que
-- únicamente deberá ejecutarse cuando cambie dicha señal.
    P2:process(clk)
    begin
            if (clk = '1' and clk'event) then ep <= pe; end if;
    end process;
end Behavioral;
```

Con respecto al testbench, podemos utilizar el siguiente:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY refresco_tb1_vhd IS
```

```
END refresco_tb1_vhd;

ARCHITECTURE behavior OF refresco_tb1_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT refresco
    PORT(
            clk : IN std_logic;
            boton : IN std_logic;
            moneda_ins: IN std_logic;
            moneda : IN std_logic_vector(7 downto 0);
            refresco : OUT std_logic;
            vuelta : OUT std_logic_vector(7 downto 0)
            );
    END COMPONENT;

    --Inputs
    SIGNAL clk :  std_logic := '0';
    SIGNAL boton, moneda_ins :  std_logic := '0';
    SIGNAL moneda :  std_logic_vector(7 downto 0) := (others=>'0');

    --Outputs
    SIGNAL refresco1 :  std_logic:='0';
    SIGNAL vuelta :  std_logic_vector(7 downto 0):="00000000";
-- Declaración de la referencia temporal como una constante.
    constant periodo: time:=40 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: refresco PORT MAP(
            clk => clk,
            boton => boton,
            moneda_ins => moneda_ins,
            moneda => moneda,
            refresco => refresco1,
            vuelta => vuelta
    );
-- Como las señales de reloj, inserción de monedas y expulsión de refrescos no tienen porque estar
-- sincronizadas, se han obtenido de forma separada, con un asignamiento concurrente para el reloj y
-- dos procesos.
-- No obstante hemos utilizado la misma referencia temporal para las tres señales.
    clk <= not clk after periodo/2;

    tb : PROCESS
    BEGIN
-- Para no complicar demasiado el testbench se ha supuesto que siempre se inserta el mismo tipo
-- de moneda, correspondiente a cuatro unidades
            moneda <= "00000100";
            moneda_ins <= not moneda_ins;
            wait for periodo;
    END PROCESS;

    tb1 : process
    begin
            boton <= '0';
            wait for periodo*1.2;
            boton <= '1';
```

```
            wait for periodo;
        end process;

    END;
```

En la figura 1.2 podemos ver las formas de onda correspondiente a la simulación de comportamiento. En dicha figura, además de mostrar los puertos de entrada y salida del sistema, también se muestran la señal interna con la cantidad almacenada y el estado presente del sistema. Pordemos apreciar que el estado presente y la cantidad se actualizan cada vez que cambian algunas de las señales de la lista de sensibilidad del proceso correspondiente. También se puede apreciar el comportamiento corecto: únicamente la señal refresco tomará el valor '1' cuando el valor de la vuelta sea positivo.

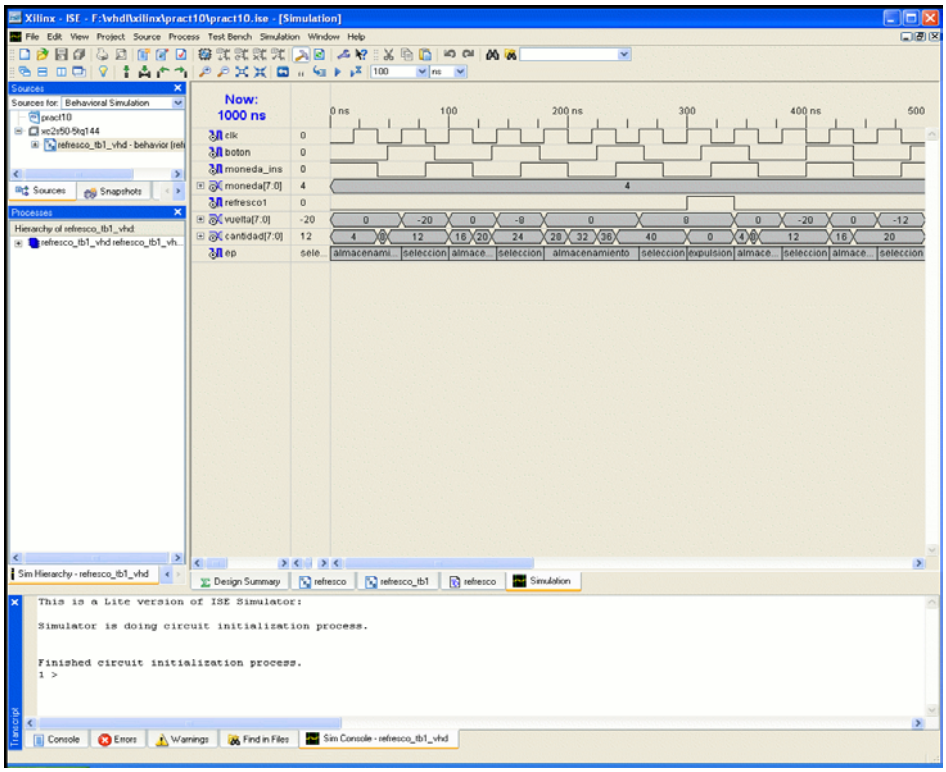

Figura 1.2.- Formas de onda correspondiente a la simulación de comportamiento.

Una vez que la simulación de comportamiento ha resultado satisfactoria, pasamos a la realización del proceso de síntesis. Una vez realizado éste, podemos ver la estructura del sistema sintetizado eligiendo la opción *View RTL schematic* de la tarea de síntesis. Esta opción nos mostrará una caja negra (realmente el color es gris) con los puertos de entrada y salida, con la cual podemos movernos por la jerarquía con un doble click o con las opciones *Push into selected Instance* (para descender) y *Pop to the calling Instance* (para ascender) desplegadas con el botón derecho del ratón. El primer nivel inferior se muestra en la figura 1.3. En esta figura podemos apreciar los diferentes módulos que ha utilizado. El esquema no tiene porqué ser plano (es decir, sin ningún nivel adicional de jerarquía); de hecho, en este esquema existe más de un nivel de jerarquíacomo podréis apreciar. Únicamente se podrá descender en la jerarquía en los bloques sintetizados, y no en los elementos de la librería de XILINX.

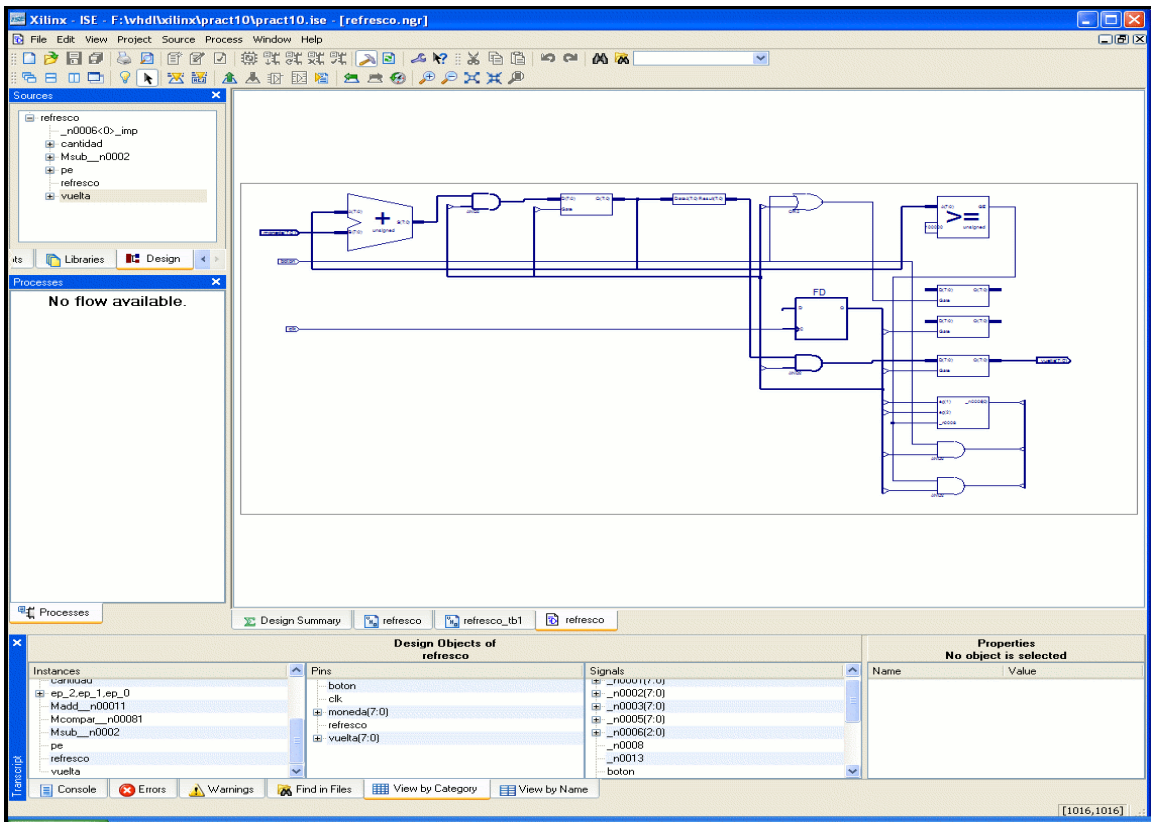

Figura 1.3.- Esquemático RTL del modelo VHDL del sistema.

Una vez realizada la implementación, podemos realizar una simulación post-rutado para verificar que el diseño funcionará correctamente. No obstante, puede que esta simulación no funcionará, seguramente debido a la inserción de los retrasos. Esta situación es la que se presenta en la figura 1.4. En este caso particular, el principal problema radica en que la cantidad almacenada no es correcta. Podemos ayudarnos del esquema RTL para tratar de solucionar el problema. (Aunque dicha solución no será realizada en este ejemplo).

También se puede obtener el par controlador-procesador del diagrama ASM anterior y modelarlo con el software de XILINX.De esta forma, el par controlador-procesador se muestra en la figura 1.5. En ella podemos distinguir el procesador y el controaldor, así como las señales de estado (señal comp) y de control (señales ep(2), ep(2) + ep(0)).

El modelo VHDL de esta representación se muestra a continuación:

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

entity refresco is
   Port ( clk, boton, moneda_ins : in  STD_LOGIC;
          moneda : in  STD_LOGIC_VECTOR (7 downto 0);
          refresco : out  STD_LOGIC;
          vuelta : out  STD_LOGIC_VECTOR (7 downto 0));
```

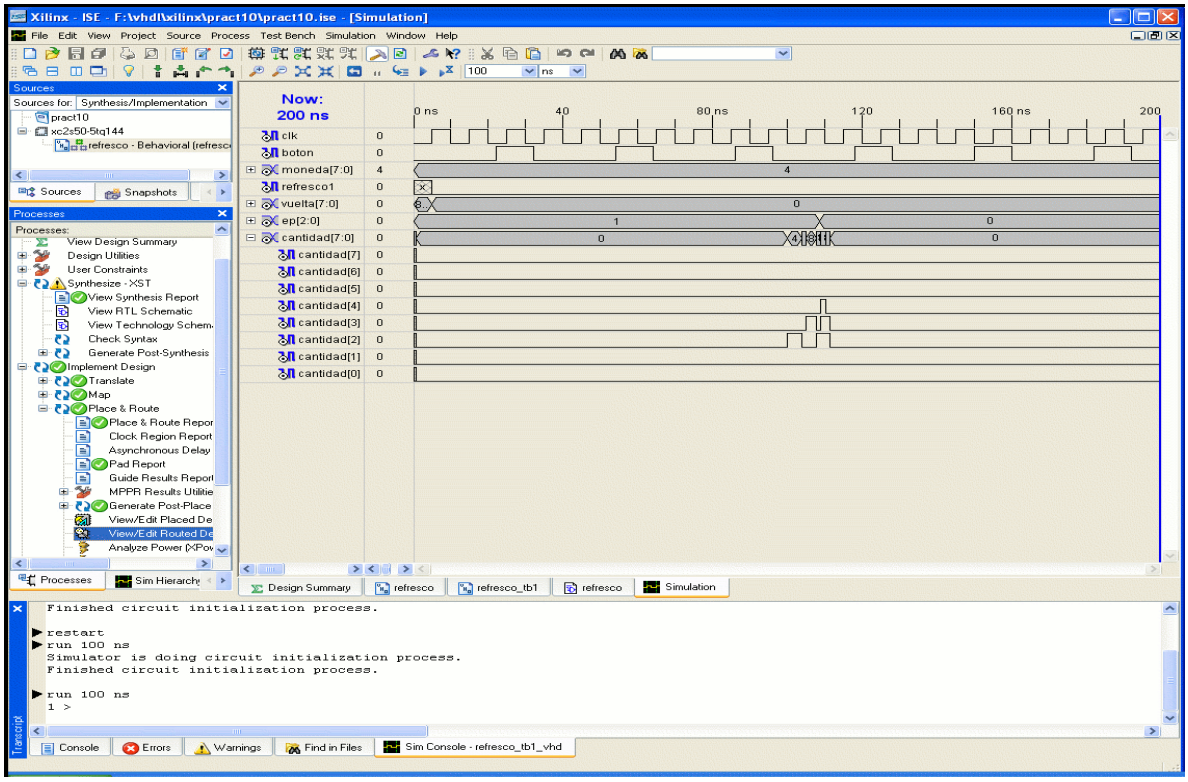

Figura 1.4.- Formas de onda de la simulación post-rutado.

```
end refresco;

architecture Behavioral1 of refresco is
    signal ep, pe: std_logic_vector(2 downto 0):="001";
    signal cantidad, cantidad1, vuelta1:std_logic_vector(7 downto 0):="00000000";
    signal comp: std_logic;

    constant precio: std_logic_vector(7 downto 0):="00100000";
begin
-- Controlador
    pe(0) <= (ep(0) and not boton) or (ep(1) and not comp) or ep(2);
    pe(1) <= (ep(0) and boton);
    pe(2) <= ep(1) and comp;
    p1:process (clk)
    begin
            if (clk = '1' and clk'event) then ep <= pe;
            end if;
    end process;
--
-- procesador
--
    -- sumador- acumulador
    cantidad <= cantidad1 + moneda;
    p2:process (clk)
    begin
            if (clk = '1' and clk'event) then
                    if (ep(2) = '1') then
                            cantidad1 <= "00000000";
                    elsif (ep(0) = '1' && moenda_ins'event) then
                            cantidad1 <= cantidad;
```

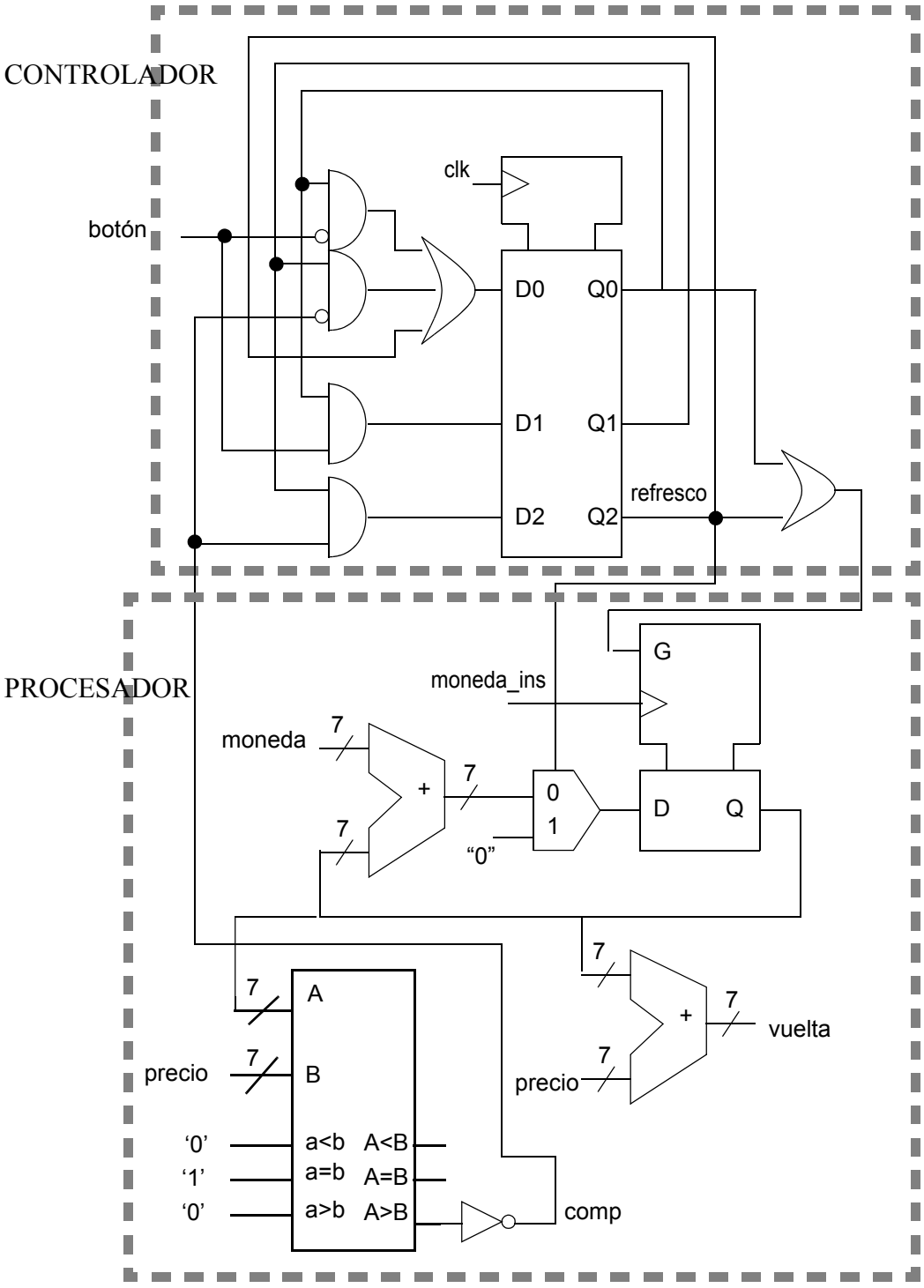

Figura 1.5.- Esquema a nivel de componentes del diagrama ASM.

```
                end if;
        end if;
end process;
-- restador
vuelta1 <= cantidad - precio;
-- comparador
```

```
    p3:process(vuelta1)
    begin
            if (cantidad >= precio) then comp <= '1';
            else comp <= '0';
            end if;
    end process;
    -- señales de salida
    vuelta <= vuelta1;
    refresco <= ep(2);
end Behavioral1;
```

En las figura 1.6 se muestran las formas de onda de las simulaciones de post-rutado, correspondiente al mismo testbench que se mostró anteriormente. Podemos observar que el comportamiento, en este caso, es correcto..

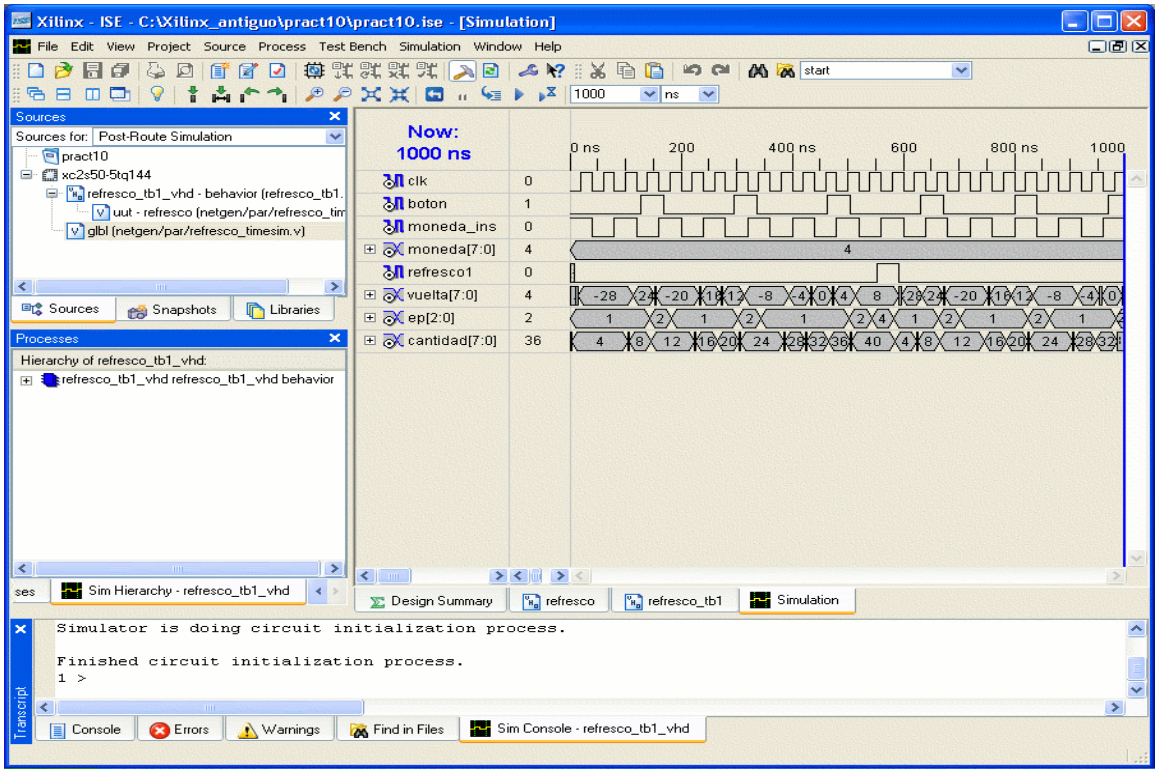

Figura 1.6.- Formas de onda de la simulación post-rutado.