Tutorial I

Diseño de una máquina de refrescos utilizando el WebPack de XILINX™

1.1. Introducción

Al ejecutar el software de Xilinx™ nos aparecerá una ventana similar a la mostrada en la figura 1.1. En esta ventana podemos distinguir cinco zonas bien diferenciadas:

- Barras de menús y botones
- Zona de fuentes, en donde aparecerán los nombres de las fuentes incluidas en el proyecto, así como el nombre del proyecto y el dispositivo sobre el que se implementará.
- Zona de procesos, en donde aparecerán los diferentes procesos que se pueden realizar con la fuente seleccionada (con todas las fuentes no tienen porqué poder ejecutarse los mismos procesos, por ejemplo, las simulaciones sólo se harán con fuentes de test-bench, la implementación sólo se hará con el Top Module, etc...)
- Zona de mensajes de consola
- Área de trabajo, zona donde se abrirán los editores para editar las fuentes, las formas de onda de simulación, etc..., y en general cualquier aplicación que se encuentre empotrada en el WebPack.

Por defecto, el WebPack abrirá el mismo proyecto utilizado en la última sesión, así como su resumen (al cual haremos alusión seguiedamente).

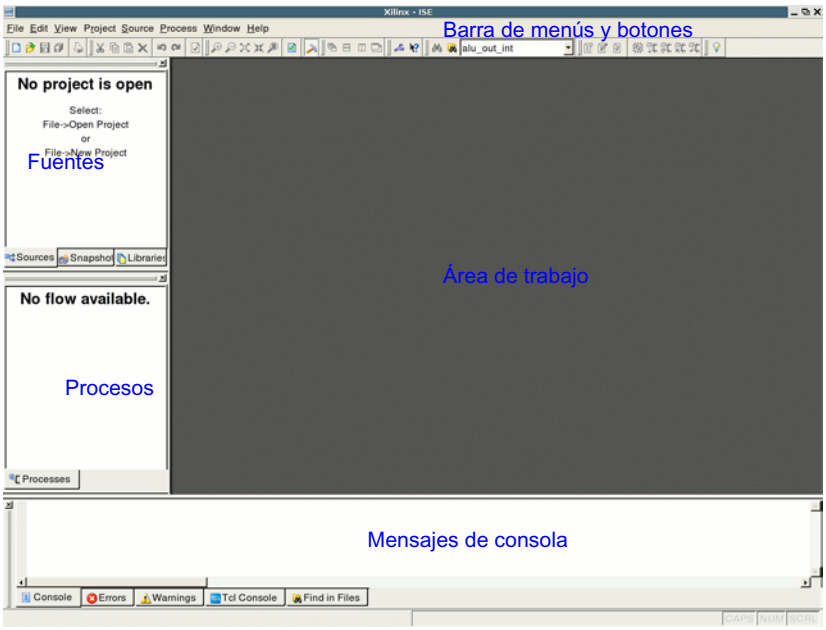


Figura 1.1.- Ventana del WebPack de Xilinx™.

Antes de empezar con el diseño propiamente dicho, vamos a inicializar la herramienta. Para poder empezar un diseño, en primer lugar tenemos que crear un proyecto. Para ello, seguimos los pasos indicados en la figura 1.2. En primer lugar se selecciona la opción de un nuevo

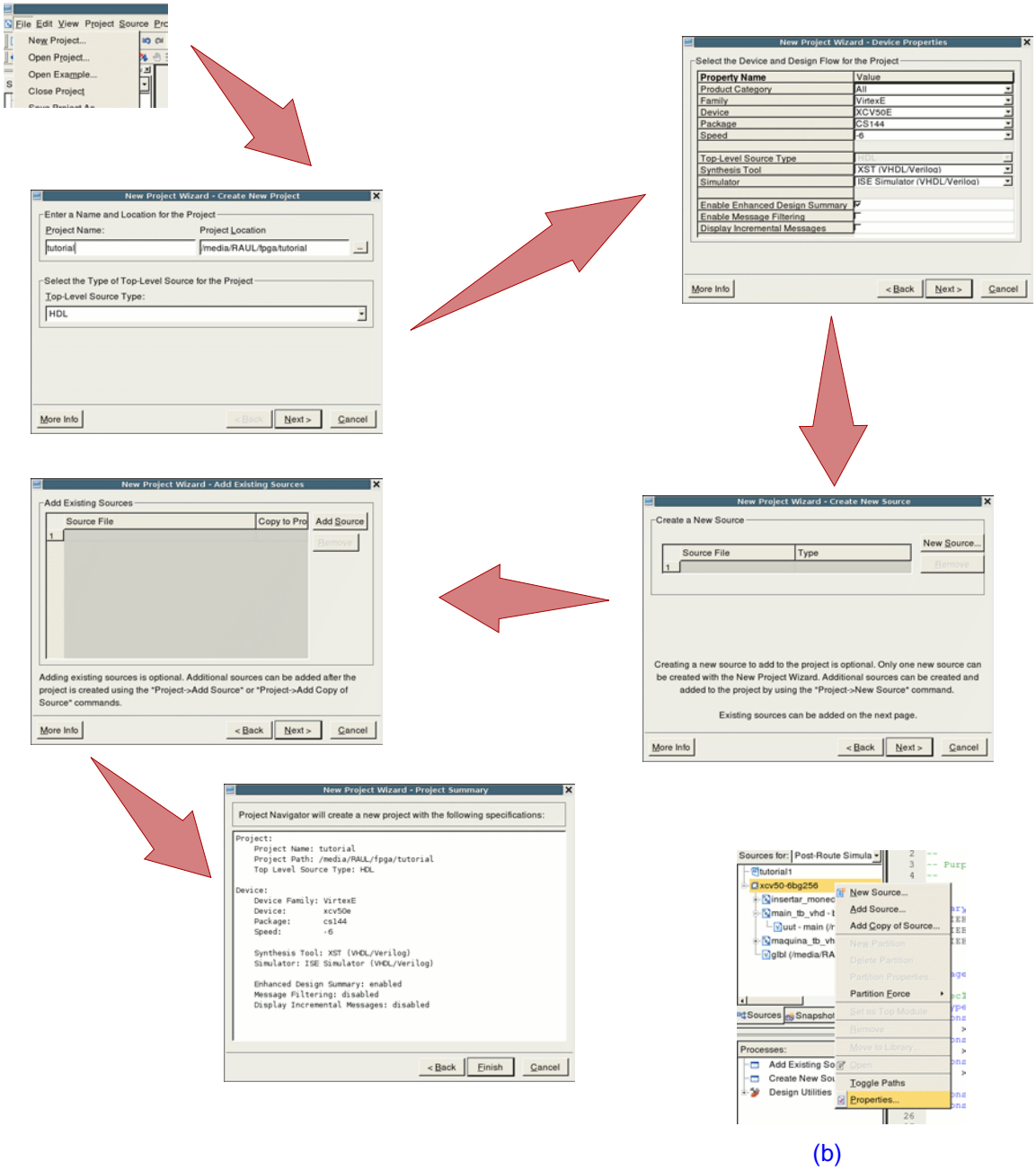


Figura 1.2.- Pasos para la creación de un proyecto.

proyecto con el menú File -> New Project. Esta opción nos llevará a la navegación a través de una serie de ventanas en la que indicaremos las características del proyecto. En la primera ventana nos pedirán el nombre, la ubicación (será un nuevo directorio en el camino indicado con el mismo nombre que el proyecto) y la forma en la que introduciremos la descripción del sistema principal (nosotros aquí siempre elegiremos HDL). Después se nos piden las características del dispositivo en el que se implementará el proyecto (nosotros siempre vamos a elegir un dispositivo FPGA como Virtex, por ejemplo; no obstante siguiendo los pasos de la figura 1.2b podemos cambiar este dispositivo así como sus características). Seguidamente se nos dará la posibilidad de incluir una nueva fuente (personalmente me gusta incluirlas después ya que sólo se nos permite incluir una, además, a priori es muy difícil conocer el número de fuentes que serán necesarias para el sistema final). En el siguiente paso, también se nos permite añadir

fuentes ya existentes (en este caso, a diferencia con el anterior es más cómodo añadirlas aquí, aunque también es posible añadirlas una vez abierto el proyecto). Finalmente se nos mostrará un resumen para comprobar los datos que hemos introducido.

Una vez abierto el proyecto, podemos crear nuevas fuentes, para lo cual debemos seguir los pasos indicados en la figura 1.3. En primer lugar seleccionamos de los procesos disponibles Create New Source (dicho proceso siempre estará accesible, siempre y cuando haya un proyecto abierto). Seguidamente nos aparecerá una ventana para indicar el nómbre, tipo y ubicación de la fuente; también nos aparece una pestaña para incluir dicha fuente en el proyecto (por defecto o no). En la ventana siguiente se nos pide los terminales de dicho módulo (los cuales pueden ser modificados en la edición si se estima oportuno), para después mostrarnos un resumen de la creación de la nueva fuente.

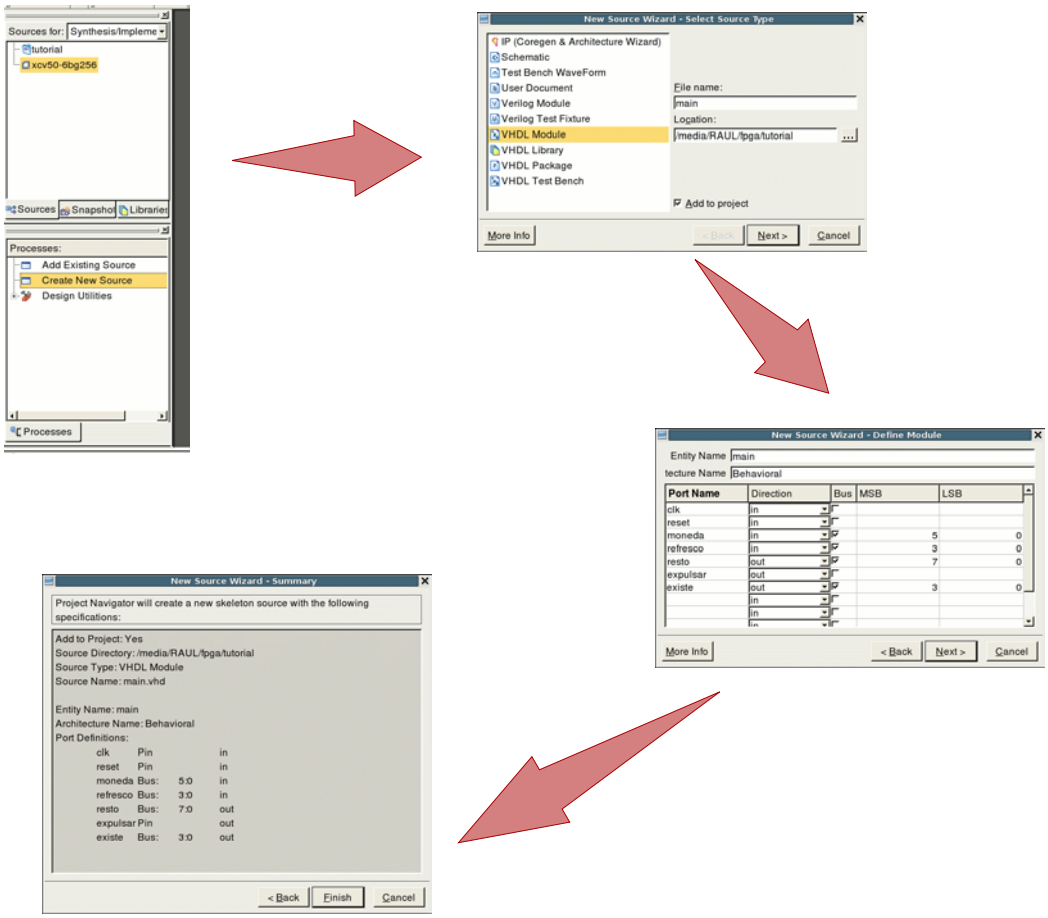
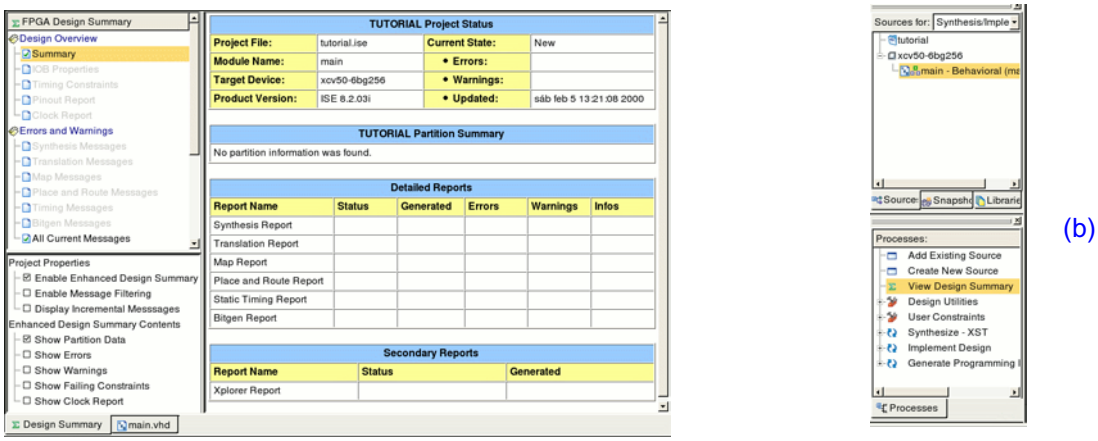


Figura 1.3.- Pasos a seguir para la creación de una nueva fuente.

El primer módulo que se crea se toma como Top Module, y se mostrará un resumen del diseño en el área de trabajo, como el mostrado en la figura 1.4. Este resumen irá cambiando a medida de que el proyecto se vaya modificando y/o completando. Si hemos cerrado la ventana de resumen y queremos verla otra vez deberemos seguir los siguientes pasos. En primer lugar, seleccionaremos el Top Module (dicho modulo es identificado por tres cuadrados (uno de ellos verdes, como se aprecia en la figura 1.4b). Uno de los procesos asociados a este módulo será View Design Summary, que activándolo nos aparecerá nuevamente la ventana del resumen actualizada..



(b)

Figura 1.4.- Resumen del diseño.

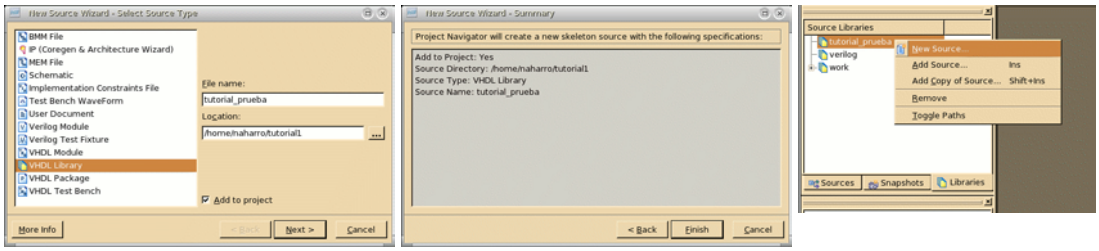
También nos aparecerá el esqueleto del código VHDL del módulo creado en un editor de textos. Dicho esqueleto se muestra a continuación.

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    13:37:47 02/05/2000
-- Design Name:
-- Module Name:    main - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity main is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           moneda : in  STD_LOGIC_VECTOR (5 downto 0);
           refresco : in  STD_LOGIC_VECTOR (3 downto 0);
           resto : out  STD_LOGIC_VECTOR (7 downto 0);
           expulsar : out  STD_LOGIC;
           existe : out  STD_LOGIC_VECTOR (3 downto 0));
end main;

architecture Behavioral of main is

begin


end Behavioral;
```

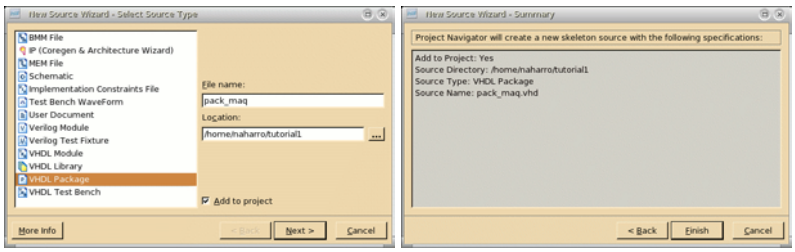
Podemos apreciar que la entidad ya estaría completa (si no queremos modificarla), y sólo habría que implementar la arquitectura y las propiedades de edición del módulo (estas propiedades serán útiles para que otro diseñador sepa en quién ha diseñado el módulo, sobre qué dispositivo ha sido implementado y una breve descripción de su funcionamiento).

Como podemos apreciar de la figura 1.3, además de módulos podemos crear otro tipo de fuentes, como Library (librerías), Package (paquetes) y Test Bench (patrones de test). Los pasos para la creación de estos nuevos tipos de fuentes son similares a los de la creación de un módulo, es decir, se pide una serie de información para finalmente dar un resumen, y son mostrados en la figura 1.5. Por lo tanto, únicamente vamos a comentar lo más destacado de ellos. En el caso de las librerias, su principal misión es agrupar fuentes que se puedan utiliza rne diferentes proyectos. Su creación no supondrá ninguna modificación aparente en la zona de fuentes, ya que en esa zona únicamente se muestran las fuentes de la librería de trabajo, es decir, work. Si queremos hacer uso de esa nueva librería debemos selecionar la pestaña Library en dicha zona, y ahi podemos visualizar todas las fuentes (excepto las IEEE que siempre están cargadas por defecto y no son, o deberían ser, modificadas) accesibles desde nuestro proyecto. En dicho caso pulsando con el botón derecho sobre la librería seleccionaremos la opción deseada, ya que en caso de utilizar la zona de procesos, todo lo que se haga estará almacenada en la librería de trabajo. En este caso no se creqrá ningún tipo de código.

Fuente del tipo librería



Fuente del tipo paquete



Fuente del tipo test bench

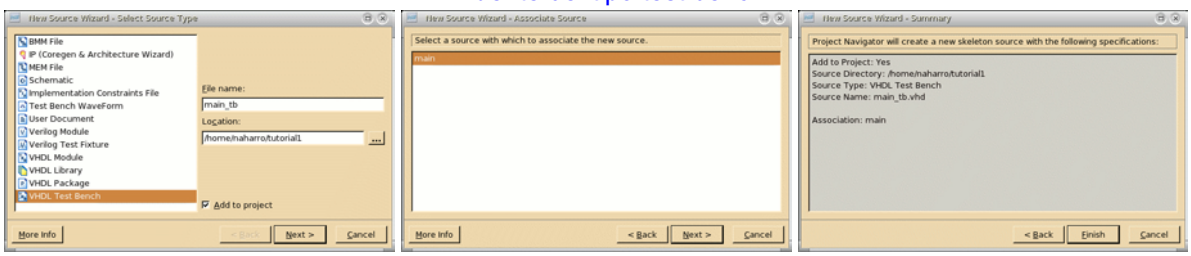


Figura 1.5.- Pasos para la creación de otros tipos de fuente.

En cuanto a los paquetes, su principal misión es agrupar definiciones que sean utilizadas por las fuentes del mismo proyecto (supuestamente por más de una fuente, ya que en caso contrario, dichas definiciones podrían ser locales a cada fuente). Puede que no sen visibles en la zona de fuentes (a pesar de que estén incluidos en la librería de trabajo), por lo que para hacerlos accesible procederemos como si estuviésemos en una librería diferente, pero visualizando la librería work. El código generado será de nuevo el esqueleto, mostrado a continuación:

```
--       Package File Template
--
--       Purpose: This package defines supplemental types, subtypes,
--                constants, and functions


library IEEE;
use IEEE.STD_LOGIC_1164.all;

package <Package_Name> is


  type <new_type> is
    record
       <type_name>        : std_logic_vector( 7 downto 0);
       <type_name>        : std_logic;
    end record;

-- Declare constants

  constant <constant_name>: time := <time_unit> ns;
  constant <constant_name>: integer := <value>;

-- Declare functions and procedure

  function <function_name>  (signal <signal_name> : in <type_declaration>) return
<type_declaration>;
  procedure <procedure_name>(<type_declaration> <constant_name>: in <type_declaration>);

end <Package_Name>;


package body <Package_Name> is

-- Example 1
  function <function_name>  (signal <signal_name> : in <type_declaration>  ) return
<type_declaration> is
    variable <variable_name>     : <type_declaration>;
  begin
    <variable_name> := <signal_name> xor <signal_name>;
    return <variable_name>;
  end <function_name>;


-- Example 2
  function <function_name>  (signal <signal_name> : in <type_declaration>;
         signal <signal_name>   : in <type_declaration>  ) return <type_declaration> is
  begin
    if (<signal_name> = '1') then
      return <signal_name>;
    else
      return 'Z';
    end if;
  end <function_name>;

-- Procedure Example
  procedure <procedure_name>  (<type_declaration> <constant_name>  : in <type_declaration>) is

  begin

  end <procedure_name>;
```

```
end <Package_Name>;
```

Podemos apreciar que aparecen los esqueletos de todas las posibles definiciones que pueden aparecer, es decir, tipos, constantes, funciones y subrutinas (procedures); obviamente sólo se utilizarán los que nos interese. También es de destacar que en el paquete hay dos partes:

- Definición del paquete (identificado por package), en el cual se llevará a cabo la declaración de todo lo que incluya el paquete.
- Cuerpo del paquete (identificado por package body), en el cual se explicitarán los cuerpos de las funciones y subrutinas, ya que los tipos y constantes sólo tienen su declaración.

En cuanto a los patrones de test, su principal misión es indicar los valores de las señales de entrada de un determinado sistema para su simulación y verificación. Una información que hay que suministrar es la fuente para la cual vamos a crear los patrones de test para su posterior simulación. El esqueleto de un patrón de test se muestra a continuación:

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   10:04:42 02/07/2007
-- Design Name:   main
-- Module Name:   G:/fpga/tutorial/pp.vhd
-- Project Name:  tutorial
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: main
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY pp_vhd IS
END pp_vhd;

ARCHITECTURE behavior OF pp_vhd IS

        -- Component Declaration for the Unit Under Test (UUT)
        COMPONENT main
        PORT(
                clk : IN std_logic;
                reset : IN std_logic;
                moneda : IN std_logic_vector(5 downto 0);
                refresco : IN std_logic_vector(3 downto 0);
                resto : OUT std_logic_vector(7 downto 0);
```

```
            existe : OUT std_logic_vector(3 downto 0);
            expulsar : OUT std_logic
            );
    END COMPONENT;

    --Inputs
    SIGNAL clk :  std_logic := '0';
    SIGNAL reset :  std_logic := '0';
    SIGNAL moneda :  std_logic_vector(5 downto 0) := (others=>'0');
    SIGNAL refresco :  std_logic_vector(3 downto 0) := (others=>'0');

    --Outputs
    SIGNAL resto :  std_logic_vector(7 downto 0);
    SIGNAL existe :  std_logic_vector(3 downto 0);
    SIGNAL expulsar :  std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: main PORT MAP(
            clk => clk,
            reset => reset,
            moneda => moneda,
            refresco => refresco,
            resto => resto,
            existe => existe,
            expulsar => expulsar
    );

    tb : PROCESS
    BEGIN

            -- Wait 100 ns for global reset to finish
            wait for 100 ns;

            -- Place stimulus here

            wait; -- will wait forever
    END PROCESS;

END;
```

En este caso, podemos destacar que la entidad está completa al ser un módulo especial puesto que los patrones de test no disponen de puertos, únicamente de señales internas para poder ser manipuladas. La declaración de la arquitectura también estaría completa ya que tenemos declarado el componente que vamos a utilizar, así como sus respectivas señales (necesidad por lo cual había que dar dicho dato). En principio, los patrones de test están pensados que se incorporen a un proceso (process) con la razón de que cuando el proceso termina vuelve a empezar desde el principio y la simulación no tendrá un tiempo final, sino que será repetitiva; no obstante, podesmo elegir cambiar este tipo de patrones, o lo que es más común, mezclar los estilos, es decir, poner asignamientos de señal fuera de procesos y uno o varios procesos. Así por ejemplo, las señales de reset suelen ser definidas fuera de procesos para evitar que el sistema se resetee antes de tiempo, mientras que las señales periódicas se suelen incluir en procesos para implementar de forma implícita la periodicidad.

Otra opción que podemos elegir para incluir fuentes en nuestro proyecto es añadir fuentes existentes (mostrada en la figura 1.6), con lo cual podemos reutilizar el trabajo de otros diseñadores (con su permiso previo). En este caso, seleccionamos el proceso Add existing source (que como en el caso anterior siempre aparecerá). Tras realizar dicho paso, aparecerá un gestor de ficheros para indicar las fuentes que queremos incluir. Una vez que hayamos seleccionado las fuentes en el gestor, dichas fuentes son compiladas para determinar el tipo de

aplicación para la que son válidas: imlpementación + simulación o sólo simulación. Una vez hecho esto, nos aparecerán las fuentes seleccionadas en la zona de fuentes. Si algunas de las fuentes incluidas son utilizadas en la jerarquía de algún módulo, dicha fuente aparecerá en su orden de jerarquía (que se podrá alcanzar descendiendo en los niveles de la fuente de mayor nivel, que aparecerá un signo +. En el caso de que el Top Module cambie, podemos reasignar dicha fuente seleccionándola y en el menú desplegado al pulsar el botón derecho del ratón, elegiremos la opción Set as Top Module, por lo que el identificativo aparecerá en este módulo y desparecerá del anterior.

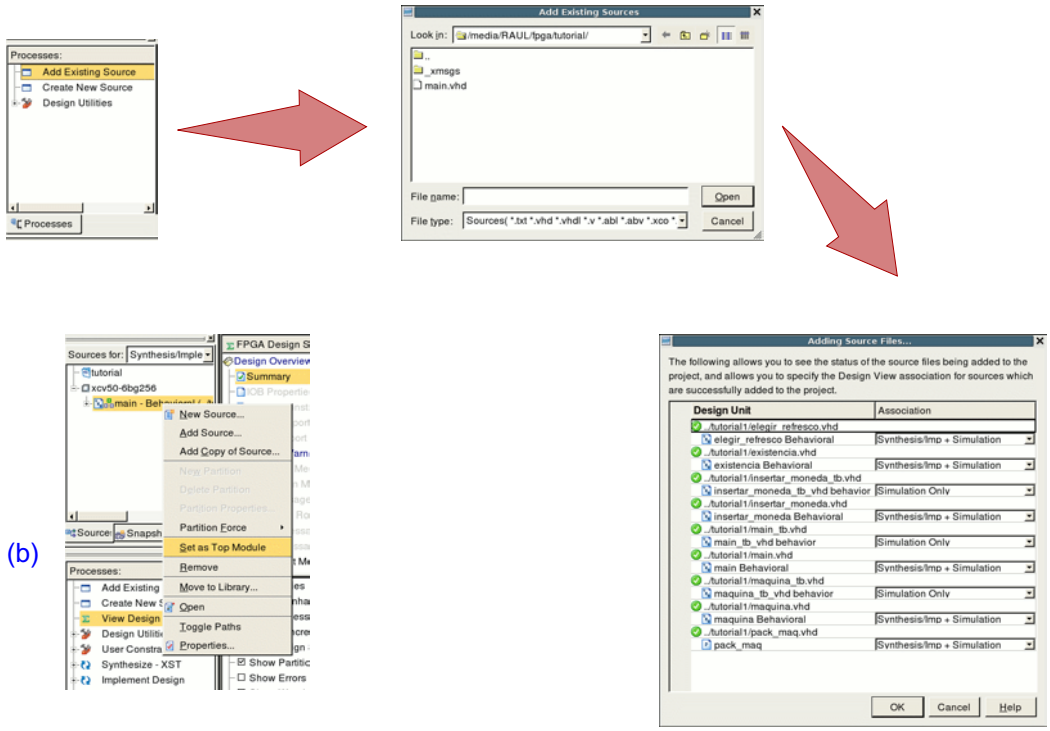


Figura 1.6.- Pasos a seguir para añadir nuevas fuentes.

1.2. Descripción de una máquina expendedora.

El diseño que vamos a abordar es el de una máquina expendedora con las siguientes características:

- Admite monedas de 5c, 10c, 20c, 50c, 1euro y 2 euros.
- Dispone de cuatro productos diferentes con los precios almacenados en una matriz constante
- Devuelve cambio hasta llegar a 12euros con 75 céntimos.
- Dispone de un indicador de existencia de cada uno de los productos.
- Dispone de un pulsador de reset con el que se devuelve la cantidad almacenada.

El diagrama ASM de dicho sistema se muestra en la figura 1.7. En él podemos apreciar que tenemos cinco cajas de estado, es decir, cinco estados del controlador, que llevarán a cabo

algún tipo de procesado o únicamente una función de espera de un ciclo de operación. No obstante para la realización del modelo VHDL de esta máquina no necesitamos conocer la implementación hardware de la misma, ya que VHDL nos permite modelar el comportamiento algorítmico.

Un posible modelo de dicha máquina sería el siguiente, denominado maquina.vhd:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    19:50:59 02/03/2007
-- Design Name:
-- Module Name:    maquina - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.pack_maq.all;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity maquina is
    Port ( clk, reset : in  STD_LOGIC;
           moneda : in  STD_LOGIC_VECTOR (3 downto 0);
           refresco : in  STD_LOGIC_VECTOR (2 downto 0);
                       resto: out std_logic_vector (7 downto 0);
                       existe: in std_logic_vector (3 downto 0);
                       desborde: out std_logic;
           expulsar : out  STD_LOGIC);
end maquina;

architecture Behavioral of maquina is
-- Posibles estados de la másquina de refresco
        type estado is (inicio, suma, eleccion, refresco1, expulsion);
        signal ep, pe: estado;
-- Señales internas
        signal cantidad: std_logic_vector (8 downto 0);
        signal cantidad_tmp: std_logic_vector(8 downto 0);
        signal prec             : std_logic_vector(7 downto 0);
begin
        p1: process(clk, reset)
        begin
                case ep is
                        when inicio =>
                                expulsar <= '0';
                                cantidad <= (others => '0');
                                cantidad_tmp <= (others => '0');
                                desborde <= '0';
                                resto <= (others => '0');
```

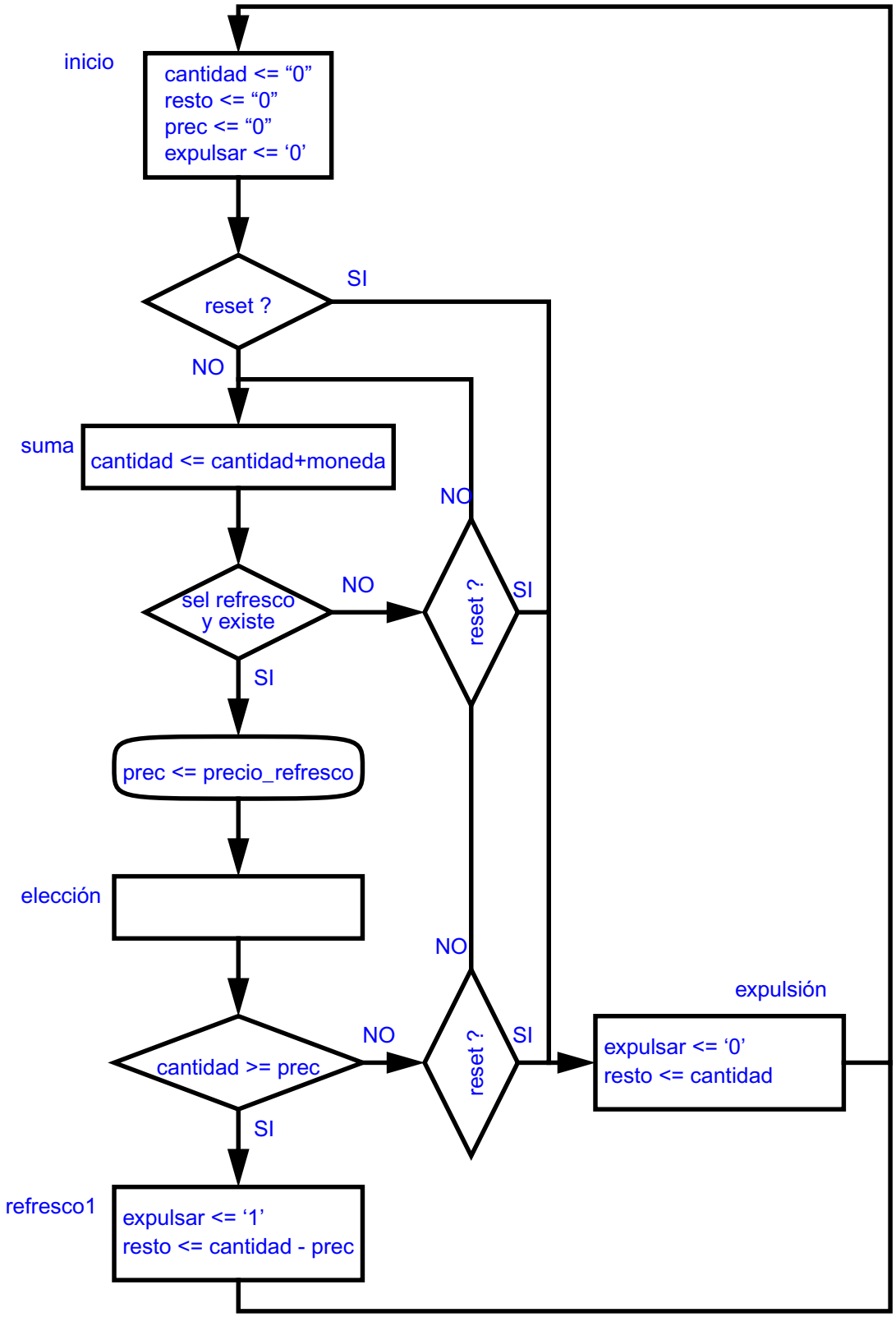


Figura 1.7.- Diagrama ASM de la máquina expendedora.

```
                        prec <= (others => '0');
                        if (reset = '1') then pe<= expulsion;
                        else pe <= suma;
                        end if;
                when suma =>
                        cantidad_tmp <= cantidad +
                                valor(CONV_INTEGER(moneda(2 downto 0)));
                        if (cantidad_tmp(8) = '1') then desborde <= '1';
                        else desborde <= '0';
                        end if;
                        if (refresco(2) = '0' and
                                existe(CONV_INTEGER(refresco(1 downto 0))) = '1') then
                                pe <= eleccion;
                                prec <= precio(CONV_INTEGER(refresco(1 downto 0)));
                        elsif (reset = '1') then pe <= expulsion;
                        end if;
                when eleccion =>
                        if (cantidad >= prec) then
                                pe <= refresco1;
                        elsif (reset = '1') then pe <= expulsion;
                        else pe <= suma;
                        end if;
                when refresco1 =>
                        expulsar <= '1';
                        resto <= cantidad(7 downto 0) - prec;
                        pe <= inicio;
                when expulsion =>
                        expulsar <= '0';
                        resto <= cantidad (7 downto 0);
                        pe <= inicio;
        end case;
        if (ep = suma and cantidad_tmp(8) = '0') then
                if (clk = '1' and clk'event) then
                        cantidad <= cantidad_tmp;
                end if;
        end if;
        if (clk = '1' and clk'event) then ep<=pe; end if;
    end process;
end Behavioral;
```

En este modelo vamos a comentar diferentes cuestiones sobre el lenguaje VHDL:

- Cualquier declaración acaba con un “;”, de hecho este signo es el que identifica los saltos de instrucción y no un retorno de carro. También podemos apreciar que no todas las declaraciones llevan “;”, como es el caso de la entidad (entity). Esto es debido a que la declaración de la entidad acaba con end <nombre>;. Los “;” que existen entre la palabras claves anteriores indican la declaración de otro tipo de elementos, como pueden ser una lista de puertos o de todos los puertos en sí.
- Todo lo que vaya situado después de “--” (dos guiones) será entendido como un comentario hasta encontrar un retorno de carro, y por lo tanto no será tenido en cuenta. La misión de los comentarios es aclarar el código, como en cualquier lenguaje de programación.
- Si se desean utilizar operaciones aritméticas, se deberá tener cargada la librería aritmética, es decir, deberán aparecer las líneas (que suelen venir por defecto en el esqueleto de los módulos):

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

- En este caso, se ha definido un tipo local denominado estado que dará nombre a cada uno de los estados del sistema. Esta definición siempre es local ya que el nombre y cantidad de estados es típico de cada sistema.

- Los algoritmos, es decir, el comportamiento secuencial, se deben incluir dentro de un proceso. Dicho proceso puede tener una lista de sensibilidad o no. La lista de sensibilidad indica el momento en que el proceso se ejecutará, siempre cuando alguna señal de la lista cambie. Si un proceso no tiene lista de sensibilidad (como en el caso de los patrones de test), el tiempo en el que cambiar se debe indicar explícita o implícitamente a través de sentencias wait.

Para cumplir con las especificaciones de diseño hay que añadir una serie de módulos extras. La razón de esta necesidad se encuentra en las decisiones tomadas a la hora de describir el sistema anterior, como son las siguientes:

- Se han considerado una serie de constantes definidas en algún otro lugar, que será un paquete. Estas constantes son los valores de valor (que se coresponde con la codificación de cada moneda) y precio (de cada uno de los productos).
- Se trabaja directamente con la codificación de las monedas y del refresco elegido.
- Se tiene en cuenta la existencia de productos pero no se actualiza.
- Por último, hay que unirlo todo en un sistema de jerarquía superior.

Luego, a continuación mostramos los códigos adicionales que hemos utilizado. El código corespondiente al paquete, que se ha llamado pack_maq.vhd será el siguiente:

```
--      Package File Template
--
--      Purpose: This package defines supplemental types, subtypes,
--               constants, and functions


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


package pack_maq is

-- Declare constants
        type moneda_valor is array (integer range <>) of std_logic_vector(7 downto 0);
        constant valor: moneda_valor (0 to 7) := (
                x"01", x"02", x"04", x"0A", x"14", x"28", x"00", x"00");
        constant precio: moneda_valor (0 to 3) := (
                x"0A", x"28", x"A0", x"0B");
        constant cantidad: moneda_valor (0 to 3) := (
                x"0A", x"0A", x"0A", x"0A");

        constant tamano_moneda: integer := 3;
        constant tamano_precio: integer := 2;

-- Declare functions and procedure

        function elecc  (signal refresco : in std_logic_vector)
        return integer;

end pack_maq;


package body pack_maq is

        function elecc  (signal refresco : in std_logic_vector)
        return integer is
```

```
        variable ref: integer;
    begin
        F1: for i in refresco'range loop
            if (refresco(i) = '1') then return i;
            end if;
        end loop;
        return refresco'length;
    end elecc;

end pack_maq;
```

En este paquete se define una matriz sin rango definido de buses de 8 bits, denominado moneda_valor, para determinar el valor de cada moneda, el precio de cada producto y el número máximo de productos que puede albergar la máquina expendedora. También se han definido una serie de constantes, como las anteriores y el tamaño del bus en el que se almacenarán dichos valores para su posterio procesado. Por último se ha definido una función que fuenciona como un codificador con prioridad devolviendo el índice del bit seleccionado.

El código utilizado para determinar la codificación de la moneda, denominado insertar_moneda.vhd, se muestra a continuación:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:38:03 02/04/2007
-- Design Name:
-- Module Name:    insertar_moneda - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.pack_maq.all;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

-- Mantendrá seleccionada la moneda insertada durante dos ciclos de operación
entity insertar_moneda is
    Port (clk : in std_logic;
                        moneda : in  STD_LOGIC_VECTOR (5 downto 0);
                        moneda_out : out  STD_LOGIC_VECTOR (3 downto 0));
end insertar_moneda;

architecture Behavioral of insertar_moneda is
        signal salida: integer;
        signal moneda_tmp: std_logic_vector (3 downto 0);
begin
```

```
        salida <= elecc(moneda);
        moneda_tmp(2 downto 0) <= CONV_STD_LOGIC_VECTOR(salida, tamano_moneda);
        moneda_tmp(3) <= '1' when salida >= moneda'length else '0';
        P1: process (clk)
        begin
                if (clk = '1' and clk'event) then moneda_out <= moneda_tmp; end if;
        end process;
end Behavioral;
```

En este caso hacemos uso de la función definida en el paquete, cuyo resultado convertimos a un bus con el tamaño definido anteriormente. Este bus tiene un bit más de lo necesario que será utilizado para determinar si s ha echado alguna moneda (el bit adicional toma el valor '0') o no se ha echado ninguna (toma el valor '1'). Esta decisión se puede tomar gracias a que la función cuando no se ha seleccionado ninguna moneda devuelve la longitud del bus de moneda, es decir, un valor mayor al que puede codificar.

El código utilizado para determinar la codificación de la moneda, denominado elegir_refresco.vhd, se muestra a continuación:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    16:01:17 02/04/2007
-- Design Name:
-- Module Name:    elegir_refresco - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.pack_maq.all;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity elegir_refresco is
    Port ( clk  : in std_logic;
                                refresco : in  STD_LOGIC_VECTOR (3 downto 0);
                                refresco_out : out  STD_LOGIC_VECTOR (2 downto 0));
end elegir_refresco;

architecture Behavioral of elegir_refresco is
        signal salida: integer;
        signal q1, q2: std_logic_vector(1 downto 0);
        signal no_refresco: std_logic;
begin
        salida <= elecc(refresco);
        q1(1 downto 0) <= CONV_STD_LOGIC_VECTOR(salida, tamano_precio);
```

```
        no_refresco <= '1' when salida >= refresco'length else '0';

        refresco_out <= no_refresco & q1;
end Behavioral;
```

Este código realiza la misma función que el anterior, pero con la selección del producto. Debemos utilizar un módulo diferente porque el tamaño de los datos de entrada y salida son diferentes.

El código utilizado para determinar la codificación de la moneda, denominado existencia.vhd, se muestra a continuación:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:40:14 02/04/2007
-- Design Name:
-- Module Name:    existencia - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.pack_maq.all;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

-- Indica cuando hay existencia de algún tipo de producto
entity existencia is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
                       expulsar: in std_logic;
           refresco : in  STD_LOGIC_VECTOR (1 downto 0);
           existe : out  STD_LOGIC_VECTOR (3 downto 0));
end existencia;

architecture Behavioral of existencia is
        signal cantidad_maquina: moneda_valor(0 to 3):= cantidad;
begin
        P1:process (clk)
        begin
                if (reset = '1') then
                        existe <= "1111";
                        cantidad_maquina <= cantidad;
                elsif (expulsar = '1') then
                        if (cantidad_maquina(CONV_INTEGER(refresco)) = X"00") then
                                existe(CONV_INTEGER(refresco)) <= '0';
                        elsif (clk = '1' and clk'event) then
```

```
                        cantidad_maquina(CONV_INTEGER(refresco)) <=
                            cantidad_maquina(CONV_INTEGER(refresco))-1;
                    end if;
                end if;
            end process;
    end Behavioral;
```

En este caso, vamos a actualizar la existencia de los productos de la máquina. Para lo cual, cuando se detecta que se ha expulsado un deteminado producto, se reduce en uno la cantidad de ese producto. Obviamente, para ser operativos, el usuario final únicamente verá si hay o no hay existencia de un determinado producto.

Por último, la unión de todos estos códigos en el mayor nivel de la jerarquia se muestra a continuación, denominado main.vhd:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    15:09:40 02/04/2007
-- Design Name:
-- Module Name:    main - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity main is
Port ( clk, reset : in  STD_LOGIC;
        moneda : in  STD_LOGIC_VECTOR (5 downto 0);
        refresco : in  STD_LOGIC_VECTOR (3 downto 0);
        resto  : out std_logic_vector (7 downto 0);
        existe : out std_logic_vector (3 downto 0);
        desborde : out std_logic;
        expulsar : out  STD_LOGIC);
end main;

architecture Behavioral of main is
        component insertar_moneda
        Port (  clk : in std_logic;
                moneda : in  STD_LOGIC_VECTOR (5 downto 0);
                moneda_out : out  STD_LOGIC_VECTOR (3 downto 0));
        end component;
        component elegir_refresco
        Port (clk: in std_logic;
                refresco : in  STD_LOGIC_VECTOR (3 downto 0);
                refresco_out : out  STD_LOGIC_VECTOR (2 downto 0));
```

```
        end component;
        component maquina
        Port ( clk, reset : in  STD_LOGIC;
               moneda : in  STD_LOGIC_VECTOR (3 downto 0);
               refresco : in  STD_LOGIC_VECTOR (2 downto 0);
                 resto: out std_logic_vector (7 downto 0);
                 existe: in std_logic_vector (3 downto 0);
                 desborde :out std_logic;
               expulsar : out  STD_LOGIC);
        end component;
        component existencia
        Port ( clk : in  STD_LOGIC;
               reset : in  STD_LOGIC;
                expulsar: in std_logic;
               refresco : in  STD_LOGIC_VECTOR (1 downto 0);
               existe : out  STD_LOGIC_VECTOR (3 downto 0));
        end component;

        signal moneda_out: std_logic_vector(3 downto 0);
        signal refresco_out: std_logic_vector(2 downto 0);
        signal expulsar1: std_logic;
        signal existe1: std_logic_vector(3 downto 0) := (others => '1');

begin
        C1: insertar_moneda port map (clk, moneda, moneda_out);
        C2: elegir_refresco port map (clk, refresco, refresco_out);
        C3: maquina port map (clk, reset, moneda_out, refresco_out,
               resto, existe1, desborde, expulsar1);
        C4: existencia port map (clk, reset, expulsar1,
               refresco_out(1 downto 0), existe1);

        expulsar <= expulsar1;
        existe <= existe1;

end Behavioral;
```

En este caso únicamente hemos conectado los diferentes bloques siguiendo el esquemos mostrado en la figura 1.8.

1.3. Simulación de comportamiento

Una vez que hemos descrito el sistema, debemos realizar una simulación para verificar el comportamiento correcto. Aunque aquí sólo vamos a realizar la comprobación del nivel superior de la jerarquía, se debería realizar de cada módulo para verificar que posibles fallos no se encuentran en el interior de los diferentes módulos.

A medida que la complejidad del sistema aumenta, es más dificil comprobar todo el comportamiento con un solo fichero de patrones. En este caso particular vamos a considerar dos ficheros de patrones de test diferentes, en el que estarán incluidos los principales comportamientos que se deben verificar (si existe algún comportamiento no verificado, hay que generar más patrones de test para completar la comprobación, es más también habría que comprobar situaciones anómalas como insertar moneda y elegir producto de forma simultánea).

En el primer fichero de patrones vamos a considerar las operaciones más comunes que deberá llevar a cabo la máquina expendedora. Dicho fichero (main_tb.vhd) se muestra a continuación:

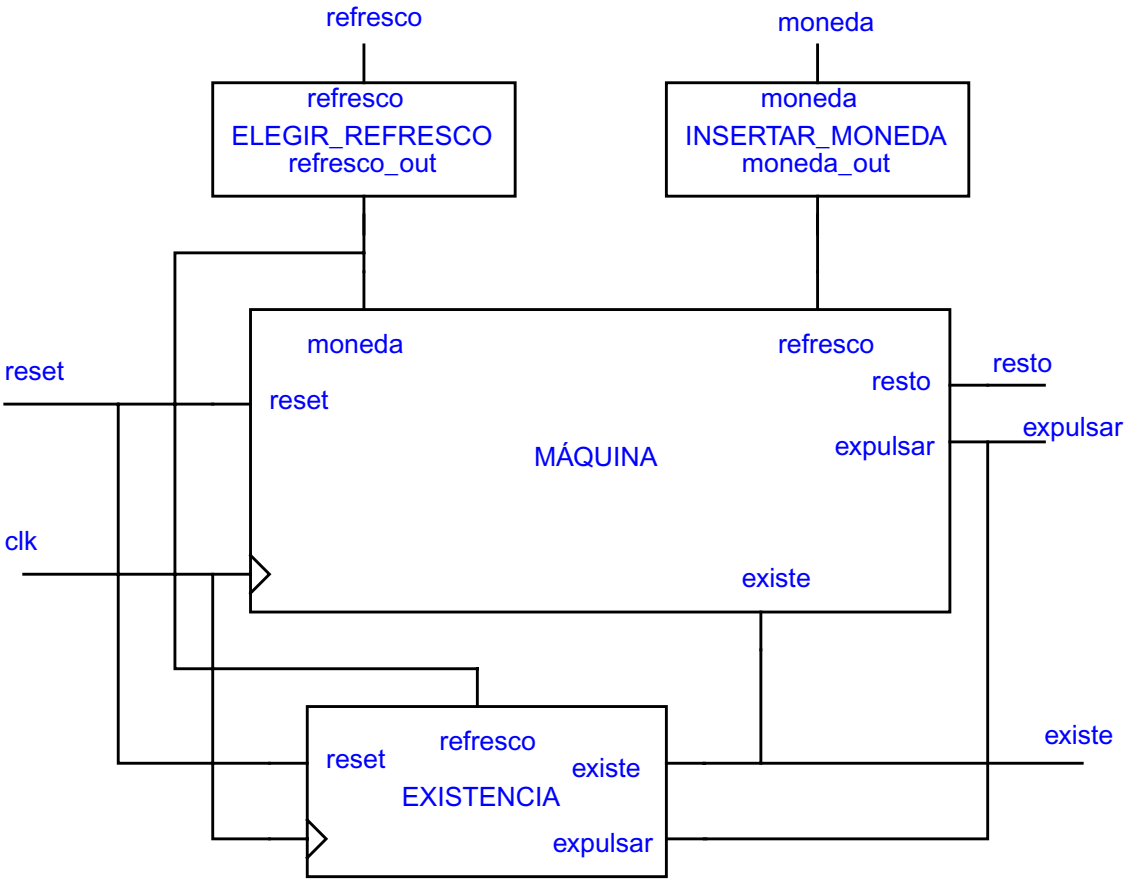


Figura 1.8.- Esquema del nivel superior de la jerarquía de la máquina expendedora.

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   21:35:49 02/03/2007
-- Design Name:   maquina
-- Module Name:   E:/fpga/tutorial1/maquina_tb.vhd
-- Project Name:  tutorial1
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: maquina
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

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

use work.pack_maq.all;

ENTITY main_tb_vhd IS
END main_tb_vhd;

ARCHITECTURE behavior OF main_tb_vhd IS

	-- Component Declaration for the Unit Under Test (UUT)
	COMPONENT main
	PORT(
		clk : IN std_logic;
		reset : IN std_logic;
		moneda : IN std_logic_vector(5 downto 0);
		refresco : IN std_logic_vector(3 downto 0);
		resto : OUT std_logic_vector(7 downto 0);
		existe : OUT std_logic_vector (3 downto 0);
		desborde : out std_logic;
		expulsar : OUT std_logic
		);
	END COMPONENT;

	--Inputs
	SIGNAL clk :  std_logic := '0';
	SIGNAL reset :  std_logic := '0';
	SIGNAL moneda :  std_logic_vector(5 downto 0) := (others=>'0');
	SIGNAL refresco :  std_logic_vector(3 downto 0) := (others=>'0');
	SIGNAL existe :  std_logic_vector(3 downto 0) := (others=>'0');

	--Outputs
	SIGNAL resto :  std_logic_vector(7 downto 0);
	SIGNAL expulsar :  std_logic;
	signal desborde : std_logic;

	constant td : time := 10 ns;

BEGIN

	-- Instantiate the Unit Under Test (UUT)
	uut: main PORT MAP(
		clk => clk,
		reset => reset,
		moneda => moneda,
		refresco => refresco,
		resto => resto,
		existe => existe,
		desborde => desborde,
		expulsar => expulsar
	);

	tbreset : process
	begin
		reset <= '1';
		wait for 3*td/2;
		reset <= '0';
		wait for 201*td/2;
	end process;

	tbclk : PROCESS
	BEGIN
		wait for td;
		clk <= not clk;
	END PROCESS;

	tbmoneda: process
	begin
		moneda <= "000000";
		wait for 10*td;
```

```
            moneda <= "000001";
            wait for 2*td;
            moneda <= "000000";
            wait for 10*td;
            moneda <= "000010";
            wait for 2*td;
            moneda <= "000000";
            wait for 10*td;
            moneda <= "000100";
            wait for 2*td;
            moneda <= "000000";
            wait for 10*td;
            moneda <= "001000";
            wait for 2*td;
            moneda <= "000000";
            wait for 10*td;
            moneda <= "010000";
            wait for 2*td;
            moneda <= "000000";
            wait for 10*td;
            moneda <= "100000";
            wait for 2*td;
        end process;

        tbrefresco: process
        begin
            refresco <= "0000";
            wait for 18*td;
            refresco <= "0001";
            wait for 2*td;
            refresco <= "0000";
            wait for 18*td;
            refresco <= "0010";
            wait for 2*td;
            refresco <= "0000";
            wait for 18*td;
            refresco <= "0100";
            wait for 2*td;
            refresco <= "0000";
            wait for 18*td;
            refresco <= "1000";
            wait for 2*td;
        end process;

    END;
```

Con estos patrones de test obtenemos las formas de onda de la figura 1.10. En este caso se comprobarán los siguientes casos:

- acumulación de monedas,
- elección de productos con los casos de no devolución (la cantidad almacenada es inferior al precio) y de devolución (cantidad almacenada superior o igual al precio),
- reset, con devolución de lacantidad almacenada

En dicha figura, además de los puertos de entrada/salida, se muestra la cantidad almacenada y los estados por los que pasa la máquina. Se puede apreciar la acumulación de las monedas insertadas. Cada vez que se realiza una selección de productos, es decir, la señal refresco toma un valor diferente de 0, se comprueba si la cantidad almacenada es lo suficientemente alta, situación que sólo se cumple en la cuarta selección. En este caso se expulsa el producto y se devuelve la vuelta. También, se muestra que al resetear el sistema, se devuelve la cantidad almacenada.

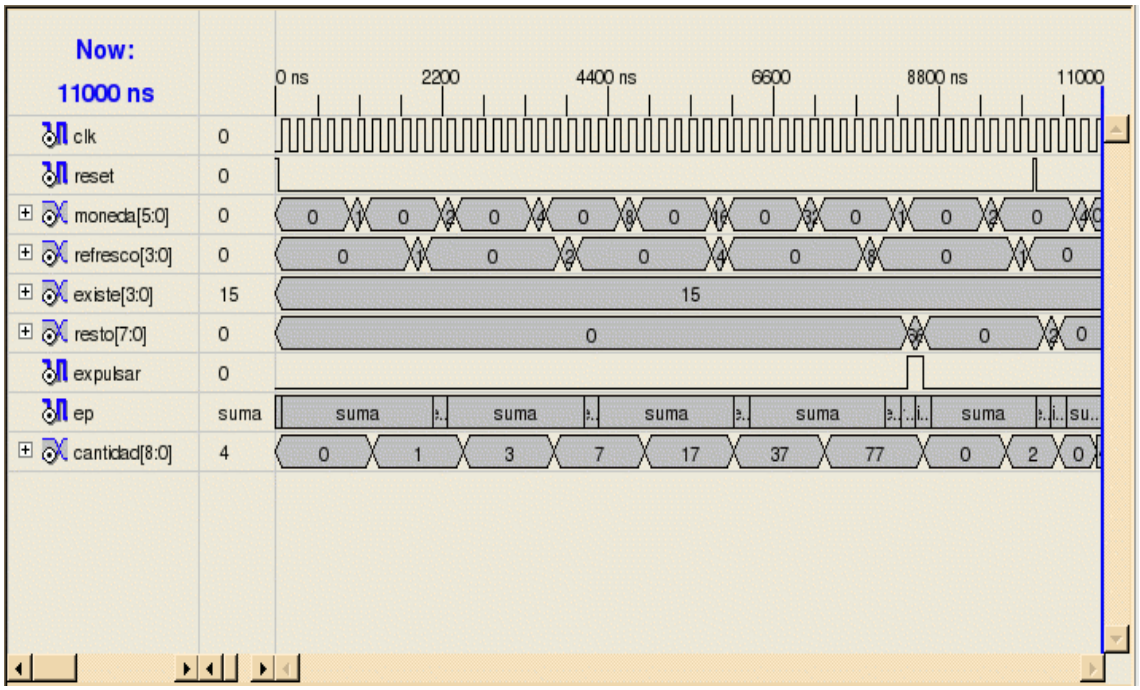


Figura 1.9.- Formas de onda de simulación.

El segundo fichero de patrones (main_tb1.vhd), con el que se completará la comprobación del sistema, se muestra a continuación:

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   10:02:30 02/09/2007
-- Design Name:   main
-- Module Name:   main_tb1.vhd
-- Project Name:  tutorial1
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: main
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
```

```
use work.pack_maq.all;

ENTITY main_tb1_vhd IS
END main_tb1_vhd;

ARCHITECTURE behavior OF main_tb1_vhd IS

	-- Component Declaration for the Unit Under Test (UUT)
	COMPONENT main
	PORT(
		clk : IN std_logic;
		reset : IN std_logic;
		moneda : IN std_logic_vector(5 downto 0);
		refresco : IN std_logic_vector(3 downto 0);
		resto : OUT std_logic_vector(7 downto 0);
		existe : OUT std_logic_vector(3 downto 0);
		desborde : OUT std_logic;
		expulsar : OUT std_logic
		);
	END COMPONENT;

	--Inputs
	SIGNAL clk :  std_logic := '0';
	SIGNAL reset :  std_logic := '0';
	SIGNAL moneda :  std_logic_vector(5 downto 0) := (others=>'0');
	SIGNAL refresco :  std_logic_vector(3 downto 0) := (others=>'0');

	--Outputs
	SIGNAL resto :  std_logic_vector(7 downto 0);
	SIGNAL existe :  std_logic_vector(3 downto 0);
	SIGNAL desborde :  std_logic;
	SIGNAL expulsar :  std_logic;

	constanttd    : time := 100ns;

BEGIN

	-- Instantiate the Unit Under Test (UUT)
	uut: main PORT MAP(
		clk => clk,
		reset => reset,
		moneda => moneda,
		refresco => refresco,
		resto => resto,
		existe => existe,
		desborde => desborde,
		expulsar => expulsar
	);

	tbreset : process
	begin
		reset <= '1';
		wait for 3*td/2;
		reset <= '0';
		wait for 300*td;
	end process;

	tbclk : PROCESS
	BEGIN
		wait for td;
		clk <= not clk;
	END PROCESS;

	tbmoneda: process
	begin
		moneda <= "000000";
		wait for 10*td;
		moneda <= "100000";
		wait for 2*td;
```

```
        end process;

        tbrefresco: process
        begin
                refresco <= "0000";
                wait for 18*td;
                refresco <= "0001";
                wait for 2*td;
        end process;

END;
```

Las formas de onda relativas al fichero de patrones anterior se muestran en la figura 1.10. En este caso, además de la acumulación, se comprueba la variación de la existencia de cada uno de los productos y el límite máximo de acumulación que determinará un límite en la vuelta.

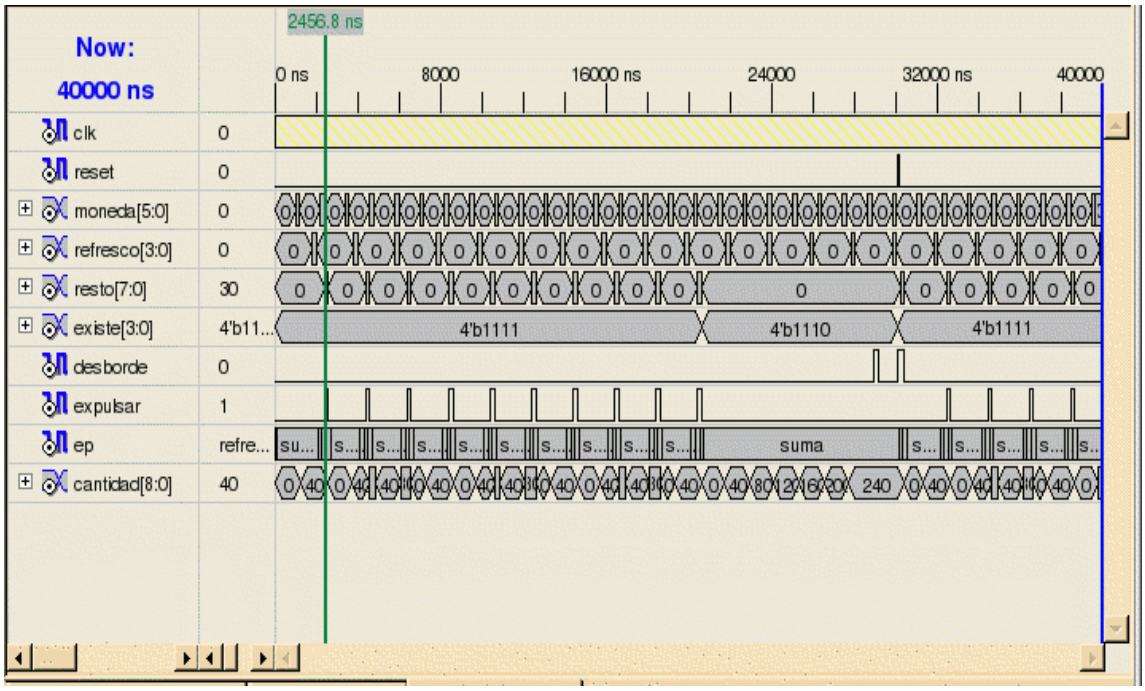


Figura 1.10.- Formas de onda de simulación.

1.4. Síntesis e implementación del sistema

Una vez que se ha comprobado el comportamiento correcto del sistema diseñado, debemos obtener una implementación física del sistema. Dicha implementación será llevada a cabo en dos pasos:

- Síntesis, en el cual se pasará de una descripción de comportamiento a una descripción de estructura. Por lo tanto, se determinarán los elementos necesarios para poder implementar el comportamiento del sistema.
- Implementación, en el cual se mapearán los elementos anteriores con elementos del dispositivo que se ha seleccionado para el proyecto.

Para llevar a cabo el proceso de síntesis e implementación, únicamente hay que ejecutar los procesos correspondientes, mostrados en la figura 1.11 (hay que comentar que dichos procesos sólo estarán disponibles en el Top Module, que será el implementado). Los resultados de dichos procesos se pueden visualizar a través de los visores/editores de las vistas correspondientes.

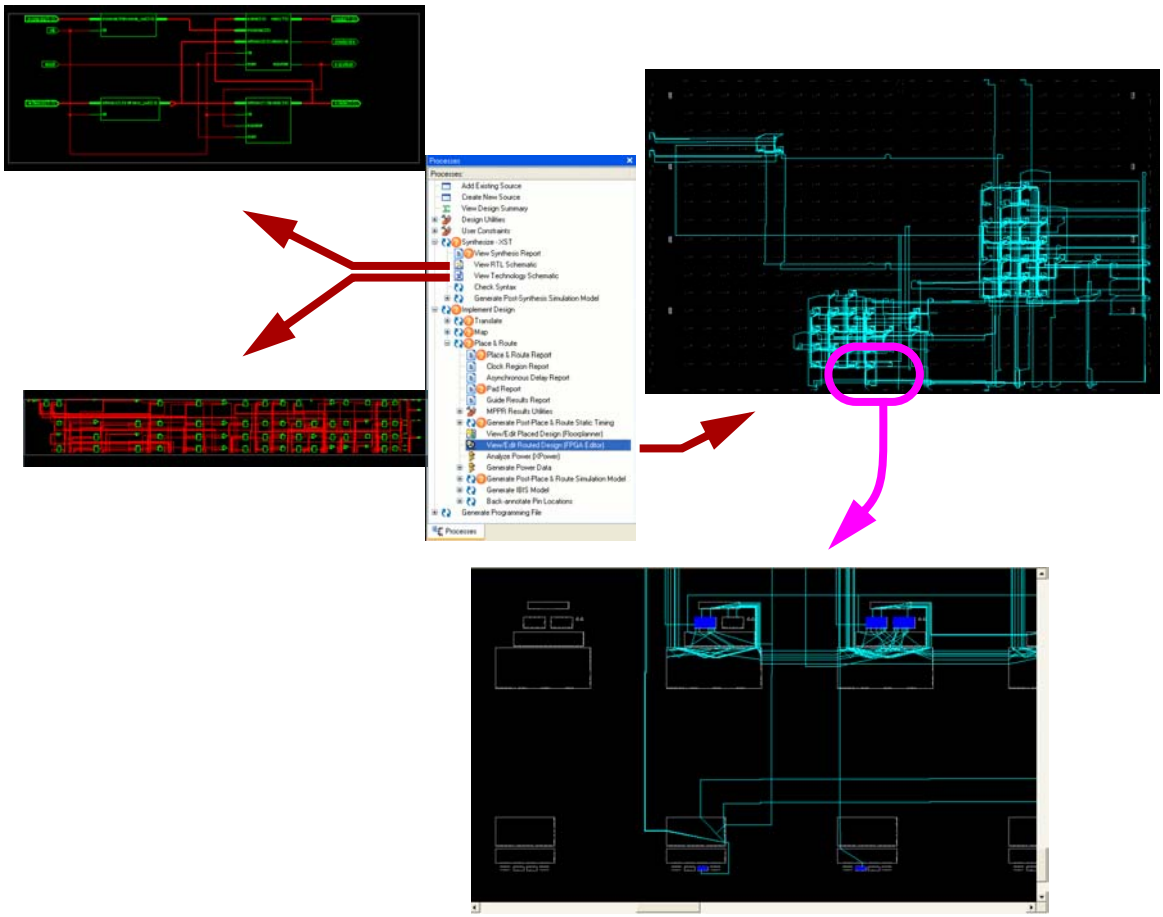


Figura 1.11.- Síntesis e implementación de la máquina expendedora.

En el caso de la síntesis, podemos visualizar el esquema RTL, en el cual los bloques de menor nivel (primitivas) mostrados serían bloques lógicos, como puertas, flip-flops, latches y algún que otro dispositivo de mayor nivel existente en la librería de Xilinx (como puede ser un sumador); o el equema tecnológico, en el cual los bloques mostrados son los bloques existentes en el dispositivo físico como pueden ser las LUT (Look-Up Table, que se encuentran dentro de los CLB, Computation Logic Block).

Tanto en una visualización como en otra, se utiliza la propiedad de jerarquía para que los esquemas sean manejables. Para viajar a través de la jerarquía, basta con hacer un doble click en el módulo al cual se quiere descender; y únicamente se podrá descender en los bloques que no sean primitivas.

En el caso de la implementación, veremos el interior del dispositivo, así como todos los elementos utilizados. Si observamos el zoom, podemos distinguir los elementos utilizados (rellenos de color azul) de los no utilizados (no rellenos). Si hacemos un doble click sobre uno de ellos, podemos entrar en él y visualizar su contenido.

1.5. Simulación de post-rutado

La simulación de comportamiento no basta para asegurar el correcto funcionamiento de un sistema, ya que en dicha simulación no se consideran elementos debido a la implementación como pueden ser la distancia existente entre un módulo y otro, lo cual es traducido en un mayor o menor retraso. De hecho, dependiendo de la complejidad del sistema, no es extraño que la simulación de post-rutado sea diferente a la de comportamiento (no sólo por un aumento de retraso, sino también de forma cualitativa). Por lo tanto, la simulación que más se parecerá al comportamiento real del sistema implementado es la simulación post-rutado, la cual sí se puede dar por definitiva.

En nuestro caso particular, las simulaciones post-rutado del sistema se muestran en la figura 1.12. En estas figuras podemos apreciar que el comportamiento no es el esperado ya que no se produce ninguna acumulación (la señal cantidad no cambia) y no se obtienen la secuencia de estados esperada (de hecho, se llega a un estado muerto del cual no se sale).

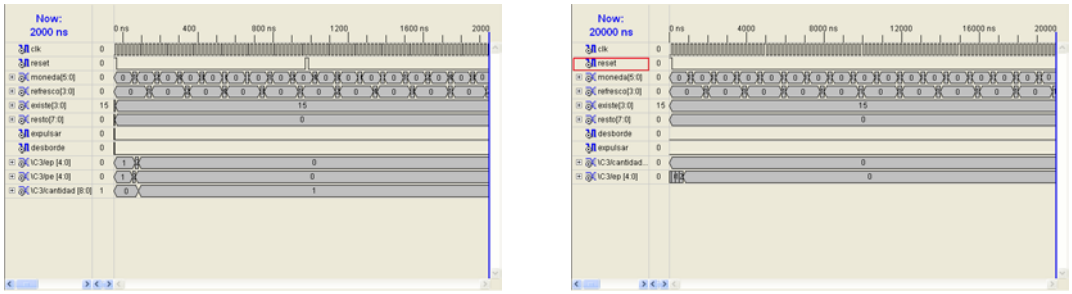


Figura 1.12.- Sismulaciones post-rutado para el caso main_tb.vhd (a) y main_tb1.vhd (b).

En el caso de que la simulación post-rutado no muestre el comportamiento correcto, hay que dirigir el proceso de síntesis para llegar al comportamiento correcto. Hay varias opciones que podemos adoptar como son la modificación de las propiedades del proceso de síntesis , sobre todo la síntesis de las máquinas de estado (FSM). Dichas opciones se muestran en la figura 1.13.

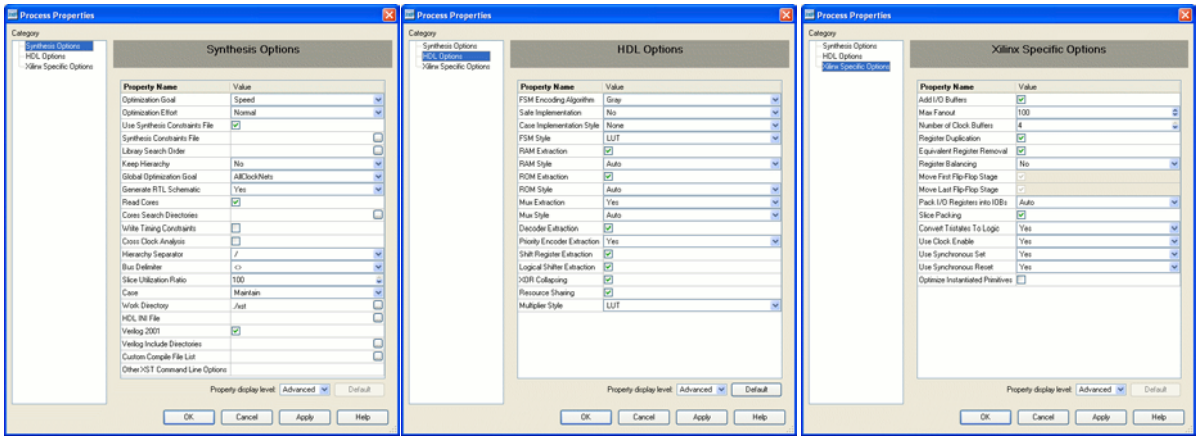
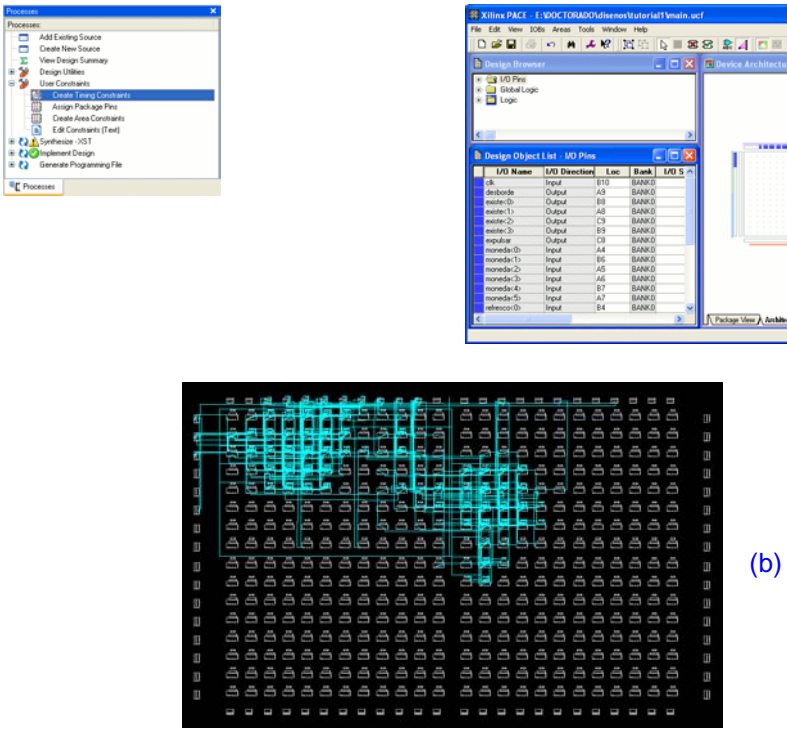


Figura 1.13.- Opciones del proceso de síntesis

Otra opción es imponer una serie de restricciones al diseño (dentro de la pestaña User Constraints). Dichas restricciones pueden ser temporales (Create timing Constraints), relativas al área (Create Area Constraints) o relativas a la colocación de los pines(Assign Package Pins), como se muestra en la FIGURA. Las restricciones más utilizadas suelen ser la colocación de los pines, ya que afecta indirectamente a las otras dos restricciones:



(b)

Figura 1.14.- Restricciones del diseño. Colocación de los pines. (b) Implementación con la nueva colocación.

- Si los pines son colocados en una misma región (todos cercanos entre sí), los módulos se colocarán cerca de los pines, y por lo tanto cerca unos de otros.
- La colocación cercana de los pines, que implca la colocación cercana de los módulos, provoca que las conexiones no sea excesivamente largas por lo que el retraso debido a ellas será menor.

Por lo tanto, hemos optado en primer lugar por la asignación de pines. En este caso hemos agrupado los pines en la esquina superior izquierda, por lo que la nueva implementación (figura 1.14) ha resultado ser más compacta que la inicial (figura 1.11). Dicha restricción se correspondería con el siguiente fichero, denominado main.ucf:

```
#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
NET "clk"  LOC = "B10"  ;
NET "desborde"  LOC = "A9"  ;
NET "existe<0>"  LOC = "B8"  ;
NET "existe<1>"  LOC = "A8"  ;
NET "existe<2>"  LOC = "C9"  ;
NET "existe<3>"  LOC = "B9"  ;
NET "expulsar"  LOC = "C8"  ;
NET "moneda<0>"  LOC = "A4"  ;
NET "moneda<1>"  LOC = "B6"  ;
NET "moneda<2>"  LOC = "A5"  ;
```

```
NET "moneda<3>"  LOC = "A6"  ;
NET "moneda<4>"  LOC = "B7"  ;
NET "moneda<5>"  LOC = "A7"  ;
NET "refresco<0>"  LOC = "B4"  ;
NET "refresco<1>"  LOC = "A3"  ;
NET "refresco<2>"  LOC = "C6"  ;
NET "refresco<3>"  LOC = "B5"  ;
NET "reset"  LOC = "C5"  ;
NET "resto<0>"  LOC = "C2"  ;
NET "resto<1>"  LOC = "B1"  ;
NET "resto<2>"  LOC = "B2"  ;
NET "resto<3>"  LOC = "E3"  ;
NET "resto<4>"  LOC = "D2"  ;
NET "resto<5>"  LOC = "G3"  ;
NET "resto<6>"  LOC = "C1"  ;
NET "resto<7>"  LOC = "F3"  ;

#PACE: Start of PACE Area Constraints

#PACE: Start of PACE Prohibit Constraints

#PACE: End of Constraints generated by PACE
```

No obstante, la nueva simulación post-rutado ha dado el mismo resultado erróneo que antes de la colocación.

El siguiente paso que podemos tomar es tocar las opciones de implementación de la máquina de estados puesto que ese es uno de los elementos que falla al no obtenerse la secuencia correcta. Podemos elegir alguno de los diferentes criterios de codificación de estados que tenemos disponibles, como son Auto, One-Hot, Compact, Sequential, Gray, Johnson, Speed 1, User y None. No obstante, con estos cambios tampoco obtuvimos un comportamiento correcto.

Por último podemos dar una descripción más orientada a la estructura, para que la herramienta deba tomar las menos elecciones posibles.

1.6. Rediseño del sistema

Para adecuar la descripción a la estructura, vamos a sustituir la máquina de estado más compleja (la correspondiente al fichero maquina.vhd) por su descripción como flujo de datos utilizando una codificación one-hot. No obstante vamos a mantener la descripción anterior, para lo cual generaremos una nueva arquitectura. El nuevo fichero maquina.vhd será el siguiente:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    19:50:59 02/03/2007
-- Design Name:
-- Module Name:    maquina - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
```

```
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use work.pack_maq.all;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity maquina is
    Port ( clk, reset : in  STD_LOGIC;
           moneda : in  STD_LOGIC_VECTOR (3 downto 0);
           refresco : in  STD_LOGIC_VECTOR (2 downto 0);
                        resto: out std_logic_vector (7 downto 0);
                        existe: in std_logic_vector (3 downto 0);
                        desborde: out std_logic;
           expulsar : out  STD_LOGIC);
end maquina;

architecture Behavioral of maquina is
-- Posibles estados de la m·squina de refresco
        type estado is (inicio, suma, eleccion, refresco1, expulsion);
        signal ep, pe: estado;
-- SeÒales internas
        signal cantidad: std_logic_vector (8 downto 0);
        signal cantidad_tmp: std_logic_vector(8 downto 0);
        signal prec             : std_logic_vector(7 downto 0);
begin
        p1: process(clk, reset)
        begin
                case ep is
                        when inicio =>
                                expulsar <= '0';
                                cantidad <= (others => '0');
                                cantidad_tmp <= (others => '0');
                                desborde <= '0';
                                resto <= (others => '0');
                                prec <= (others => '0');
                                if (reset = '1') then pe<= expulsion;
                                else pe <= suma;
                                end if;
                        when suma =>
                                cantidad_tmp <= cantidad +
                                        valor(CONV_INTEGER(moneda(2 downto 0)));
                                if (cantidad_tmp(8) = '1') then desborde <= '1';
                                else desborde <= '0';
                                end if;
                                if (refresco(2) = '0' and
                                        existe(CONV_INTEGER(refresco(1 downto 0))) = '1') then
                                        pe <= eleccion;
                                        prec <= precio(CONV_INTEGER(refresco(1 downto 0)));
                                elsif (reset = '1') then pe <= expulsion;
                                end if;
                        when eleccion =>
                                if (cantidad >= prec) then
                                        pe <= refresco1;
                                elsif (reset = '1') then pe <= expulsion;
                                else pe <= suma;
                                end if;
                        when refresco1 =>
                                expulsar <= '1';
                                resto <= cantidad(7 downto 0) - prec;
                                pe <= inicio;
```

```
                when expulsion =>
                        expulsar <= '0';
                        resto <= cantidad (7 downto 0);
                        pe <= inicio;
        end case;
        if (ep = suma and cantidad_tmp(8) = '0') then
                if (clk = '1' and clk'event) then
                        cantidad <= cantidad_tmp;
                end if;
        end if;
        if (clk = '1' and clk'event) then ep<=pe; end if;
    end process;
end Behavioral;


architecture flujo of maquina is
    signal ctrl     : std_logic_vector (1 downto 0);
    signal estado, estadop: std_logic_vector (4 downto 0);
    signal prec: std_logic_vector (7 downto 0);
    signal cantidad_tmp: std_logic_vector (8 downto 0);
    signal cantidad: std_logic_vector (8 downto 0);
begin

    ctrl(0) <= not refresco(2) and existe(CONV_INTEGER(refresco(1 downto 0)));
    ctrl(1) <= '1' when cantidad >= prec else '0';

    estadop(0) <= estado(3) or estado(4);
    estadop(1) <= (estado(0) and not reset) or
                                    (estado(1) and not ctrl(0) and not reset) or
                                    (estado(2) and not ctrl(1) and not reset);
    estadop(2) <= estado(1) and ctrl(0);
    estadop(3) <= estado(2) and ctrl(1);
    estadop(4) <= reset and (estado(0) or estado(1) or estado(2));

    expulsar <= '1' when estado(3)='1' else '0';
    cantidad_tmp <= (others => '0') when estado(0) = '1' else
                                            cantidad + valor(CONV_INTEGER(moneda(2 downto
0)));
    desborde <= cantidad_tmp(8);
    resto <= cantidad(7 downto 0) - prec when estado(3) = '1' else
                    cantidad(7 downto 0) when estado(4) = '1' else
                    (others => '0');
    prec <= precio(CONV_INTEGER(refresco(1 downto 0)))
                when (ctrl(0) = '1' and estado(1)='1');

    P1:process (clk)
    begin
        if (clk = '1' and clk'event) then
                if (cantidad_tmp(8) = '0') then
                        cantidad <= cantidad_tmp;
                end if;
                if (reset = '1') then
                        estado <= "10000";
                else estado <= estadop;
                end if;
        end if;
    end process;

end flujo;
```

La nueva arquitectura ha sido denominada flujo. El único proceso existente será para la actualización de las señales de estado y la cantidad almacenada.

Para que la unidad superior en jerarquía sepa qué arquitectura debe utilizar se debe utilizar una línea como la siguiente en la parte declarativa de su propia arquitectura.

```
for all:maquina use entity work.maquina(flujo);
```

Donde el nombre de la arquitectura a utilizar va entre paréntesis. Por lo tanto, la porción declarativa del fichero main.vhd quedaría de la siguiente forma.

```
architecture Behavioral of main is
        component insertar_moneda
        Port (  clk : in std_logic;
                moneda : in  STD_LOGIC_VECTOR (5 downto 0);
                moneda_out : out  STD_LOGIC_VECTOR (3 downto 0));
        end component;
        component elegir_refresco
        Port (clk: in std_logic;
                refresco : in  STD_LOGIC_VECTOR (3 downto 0);
                refresco_out : out  STD_LOGIC_VECTOR (2 downto 0));
        end component;
        component maquina
        Port ( clk, reset : in  STD_LOGIC;
                moneda : in  STD_LOGIC_VECTOR (3 downto 0);
                refresco : in  STD_LOGIC_VECTOR (2 downto 0);
                resto  : out std_logic_vector (7 downto 0);
                existe : in std_logic_vector (3 downto 0);
                desborde :out std_logic;
                expulsar : out  STD_LOGIC);
        end component;

        for all:maquina use entity work.maquina(flujo);

        component existencia
        Port ( clk : in  STD_LOGIC;
                reset : in  STD_LOGIC;
                  expulsar: in std_logic;
                refresco : in  STD_LOGIC_VECTOR (1 downto 0);
                existe : out  STD_LOGIC_VECTOR (3 downto 0));
        end component;

        signal moneda_out: std_logic_vector(3 downto 0);
        signal refresco_out: std_logic_vector(2 downto 0);
        signal expulsar1: std_logic;
        signal existe1: std_logic_vector(3 downto 0) := (others => '1');

begin
```

Debido al cambio de la arquitectura, debemos volver a comprobar el comportamiento para asegurarnos que no ha cambiado. En la figura 1.15 mostramos las formas de onda de las simulaciones de comportamiento de la nueva arquitectura, y podemos observar que el comportamiento no ha cambiado. La única diferencia se encuentra en que las señales ep son cambiadas por las señales estados (nombre con el que se han declarado en la nueva arquitectura), y que ahora toman valores numéricos (y no nombres de estado como sucedía en la arquitectura anterior).

Tras esta comprobación, obtenemos la implementación del nuevo sistema. Para esta implementación vamos a mantener las restricciones del apartado anterior y aprovechar, de este modo, sus ventajas. Esta nueva implementación se muestra en la figura 1.16. En esta nueva implementación podemos apreciar que la distribución de los diferentes módulos es similar a la anterior. Esta situación es motivada por la utilización de la misma colocación de pines, como se ha comentado previamente.

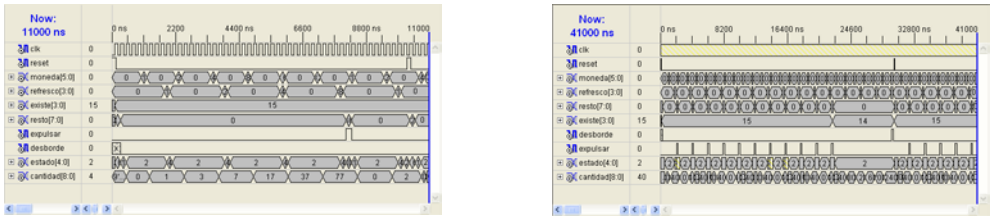


Figura 1.15.- Formas de onda de la simulación de comportamiento de la arquitectura flujo.

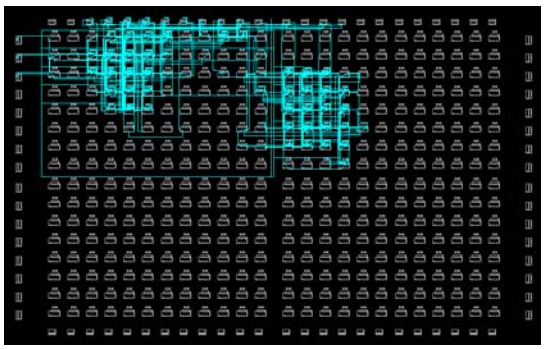
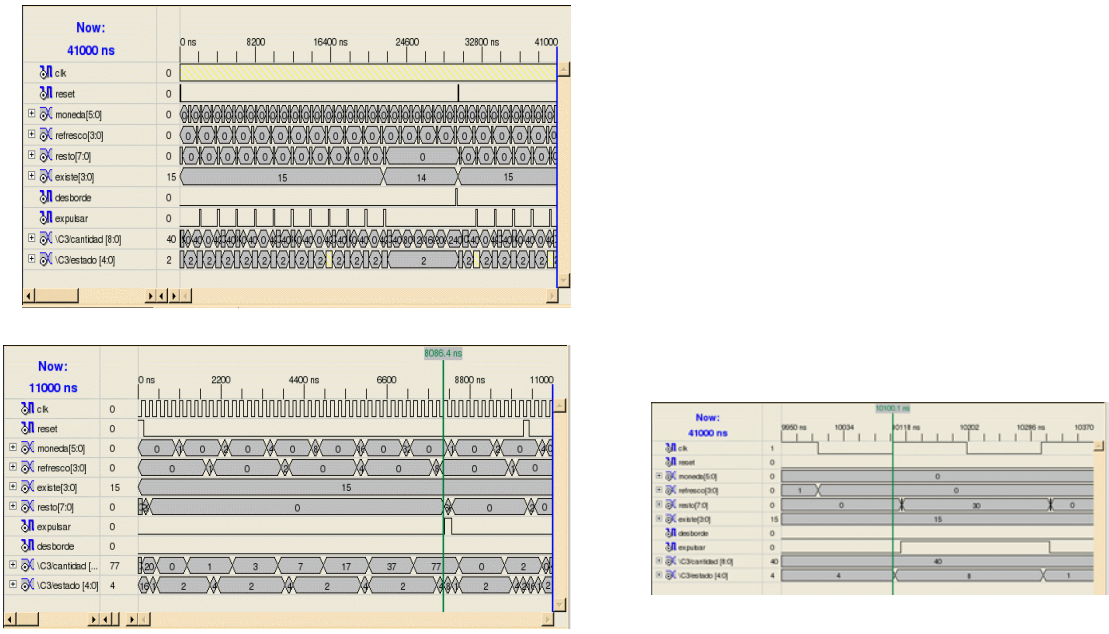


Figura 1.16.- Implementación del sistema utilizando la arquitectura flujo.

El diseño será completado con la simulación post-rutado. En estas simulaciones podemos comprobar que se obtiene el comportamiento correcto. También se muestra un zoom de una de las simulaciones para mostrar las diferencias temporales entre las diferentes señales fruto de los retrasos debido a la implementación física del sistema.



(b)

Figura 1.17.- Formas de onda de la simulación post-rutado de la arquitectura flujo. (b) Zoom de una de las simulaciones.

1.7. Caracterización del sistema

Una vez que tenemos un prototipo con un funcionamiento correcto en la simulación post-rutado, podemos abordar la tarea de caractreización del sistema. Aunque dicha tarea suele ser llevada a cabo de forma física, el entorno de diseño tiene una serie de herramientas que nos proporcionan una estimación fiable. Vamos a considerar como caracterización los siguientes parámetros: área ocupada, velocidad de operación y consumo de potencia.

1.7.1. Área ocupada

La medida de este parámetro va a ser dada en el número de puertas equivalentes. A continuación mostramos parte de los mensajes de consola obtenidos cuando se ejecuta el proceso de síntesis e implementación.

```
                    ...

Process "Translate" completed successfully
Using target part "v50bg256-6".
Mapping design into LUTs...
Running directed packing...
Running delay-based LUT packing...
Running related packing...

Design Summary:
Number of errors:      0
Number of warnings:    2
Logic Utilization:
  Total Number Slice Registers:      57 out of  1,536    3%
    Number used as Flip Flops:                49
    Number used as Latches:                    8
  Number of 4 input LUTs:          123 out of  1,536    8%
Logic Distribution:
    Number of occupied Slices:                      74 out of    768    9%
    Number of Slices containing only related logic:     74 out of     74  100%
    Number of Slices containing unrelated logic:        0 out of     74    0%
       *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:         131 out of  1,536    8%
      Number used as logic:                   123
      Number used as a route-thru:              8
   Number of bonded IOBs:           25 out of    180   13%
     IOB Latches:                          1
   Number of GCLKs:                 1 out of      4   25%
   Number of GCLKIOBs:               1 out of      4   25%

Total equivalent gate count for design:  1,511
Additional JTAG gate count for IOBs:  1,248
Peak Memory Usage:  123 MB

                    ...

Device Utilization Summary:

   Number of GCLKs                   1 out of 4      25%
   Number of External GCLKIOBs         1 out of 4      25%
     Number of LOCed GCLKIOBs         1 out of 1     100%

   Number of External IOBs           25 out of 180    13%
     Number of LOCed IOBs            25 out of 25    100%

   Number of SLICEs                  74 out of 768     9%

                    ...
```

De esta información podemos identificar el número de puertas equivalentes del sistema implementado, en este caso particular es de 1511 puertas. También muestra el grado de ocupación de los diferentes componentes del dispositivo seleccionado.

Si continuamos observando los mensajes, también podemos observar algunas medidas temporales, aunque podemos obtener información más precisa con la utilidad TRACE.

```
                              ...

**************************
Generating Clock Report
**************************

+----------------- + ---------------- + ---------------- + ---------------- + ------------------------- + ----------------------
| Clock Net   | Resource   | Locked       | Fanout       | Net Skew(ns)       | Max Delay(ns)     |
+----------------- + ---------------- + ---------------- + ---------------- + ------------------------- + ----------------------
| clk_BUFGP| GCLKBUF3| No           |   28           |  0.055               |  0.358               |
+----------------- + ---------------- + ---------------- + ---------------- + ------------------------- + ----------------------
| C3/ctrl<0>  | Low-Skew  |              | 5              |  0.161               |  3.688               |
+----------------- + ---------------- + ---------------- + ---------------- + ------------------------- + ----------------------
| C4/_not0001| Local       |              | 2              | 1.471                | 2.734                |
+----------------- + ---------------- + ---------------- + ---------------- + ------------------------- + ----------------------

* Net Skew is the difference between the minimum and maximum routing
only delays for the net. Note this is different from Clock Skew which
is reported in TRCE timing report. Clock Skew is the difference between
the minimum and maximum path delays which includes logic delays.

Timing Score: 0

Asterisk (*) preceding a constraint indicates it was not met.
   This may be due to a setup or hold violation.

--------------------------------------------------------------------------------------------------------------------------------
  Constraint                                | Requested | Actual| Logic | Absolute |Nº of
                                            |           |       | Levels | Slack   |errors
--------------------------------------------------------------------------------------------------------------------------------
  Autotimespec constraint for clock net clk | N/A       | 7.431ns | 2 | N/A      | N/A
  _BUFGP                                    |           |      ....|   |          |
--------------------------------------------------------------------------------------------------------------------------------
  Autotimespec constraint for clock net C4/ | N/A       | 3.633ns | 1 | N/A      | N/A
  _not0001                                  |           |     .....|   |          |
--------------------------------------------------------------------------------------------------------------------------------


All constraints were met.
INFO:Timing:2761 - N/A entries in the Constraints list may indicate that the
   constraint does not cover any paths or that it has no requested value.

                              ...
```

En la figura 1.18 podemos apreciar donde se encuentran cada una de las palicaciones para obtener una estimación de las prestaciones que tendría nuestro sistema.

1.7.2. Velocidad de operación

Al ejecutar la apliación de parámetros temporales, se obtiene el siguiente report, en el cual se muestra todas las medidas temporales de las diferentes señales y caminos. Entre estas medidas podemos destacar los retrasos máximos de cada una de las señales, y los retrasos que se deben verificar para que el almacenamiento en los biestables sea correcto.

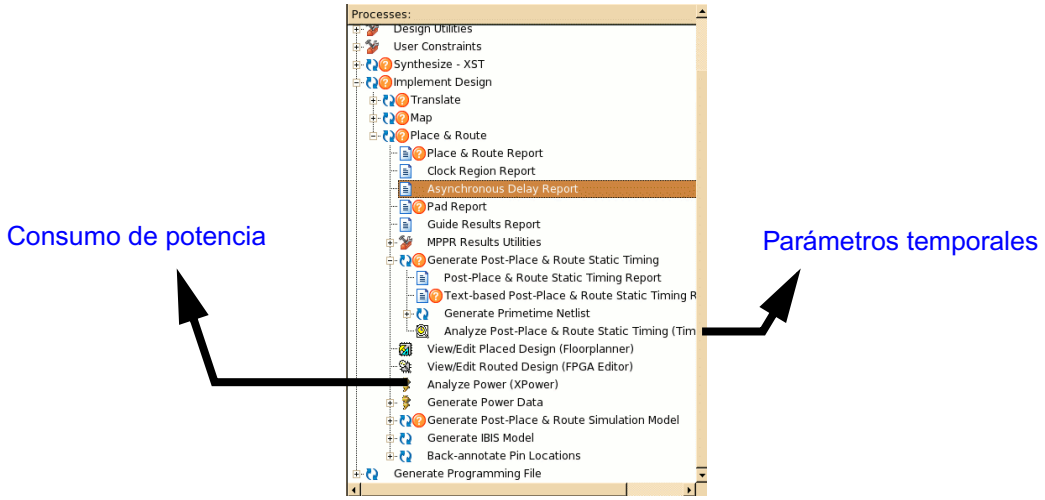


Figura 1.18.- Ubicación de las aplicaciones de caracterización.

```
--------------------------------------------------------------------------------
Release 8.2.03i Trace
Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.

C:\Xilinx\bin\nt\trce.exe -ise E:/DOCTORADO/disenos/tutorial1/tutorial1.ise
-intstyle ise -e 3 -l 3 -s 6 -xml main main.ncd -o main.twr main.pcf -ucf
main.ucf


Design file:              main.ncd
Physical constraint file: main.pcf
Device,speed:             xcv50,-6 (FINAL 1.123 2006-08-18)
Report level:             error report

Environment Variable      Effect
------------------------  ------
NONE                      No environment variables were set
--------------------------------------------------------------------------------

INFO:Timing:2698 - No timing constraints found, doing default enumeration.
INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
   option. All paths that are not constrained will be reported in the
   unconstrained paths section(s) of the report.


Data Sheet report:
------------------
All values displayed in nanoseconds (ns)

Setup/Hold to clock clk
------------+------------+------------+------------------+--------+
            | Setup to   | Hold to    |                  | Clock  |
Source      | clk (edge) | clk (edge) |Internal Clock(s) | Phase  |
------------+------------+------------+------------------+--------+
moneda<0>   |    2.655(R)|   -0.422(R)|clk_BUFGP         |   0.000|
moneda<1>   |    2.673(R)|   -0.200(R)|clk_BUFGP         |   0.000|
moneda<2>   |    2.808(R)|   -0.313(R)|clk_BUFGP         |   0.000|
moneda<3>   |    2.986(R)|   -0.427(R)|clk_BUFGP         |   0.000|
moneda<4>   |    2.634(R)|   -0.208(R)|clk_BUFGP         |   0.000|
moneda<5>   |    2.329(R)|    0.138(R)|clk_BUFGP         |   0.000|
reset       |    3.227(R)|    0.454(R)|clk_BUFGP         |   0.000|
------------+------------+------------+------------------+--------+

Clock clk to Pad
```

Destination	clk (edge) to PAD		Internal Clock(s)	Clock Phase
desborde	13.312(R)	clk_BUFGP		0.000
expulsar	9.093(R)	clk_BUFGP		0.000
resto<0>	14.009(R)	clk_BUFGP		0.000
resto<1>	14.565(R)	clk_BUFGP		0.000
resto<2>	14.390(R)	clk_BUFGP		0.000
resto<3>	14.423(R)	clk_BUFGP		0.000
resto<4>	14.212(R)	clk_BUFGP		0.000
resto<5>	13.706(R)	clk_BUFGP		0.000
resto<6>	14.388(R)	clk_BUFGP		0.000
resto<7>	13.861(R)	clk_BUFGP		0.000

Clock refresco<0> to Pad

Destination	clk (edge) to PAD		Internal Clock(s)	Clock Phase
resto<0>	17.256(F)	C3/ctrl<0>		0.000
resto<1>	17.933(F)	C3/ctrl<0>		0.000
resto<2>	18.449(F)	C3/ctrl<0>		0.000
resto<3>	18.482(F)	C3/ctrl<0>		0.000
resto<4>	18.271(F)	C3/ctrl<0>		0.000
resto<5>	17.765(F)	C3/ctrl<0>		0.000
resto<6>	18.447(F)	C3/ctrl<0>		0.000
resto<7>	17.920(F)	C3/ctrl<0>		0.000

Clock refresco<1> to Pad

Destination	clk (edge) to PAD		Internal Clock(s)	Clock Phase
existe<0>	19.005(R)	C4/_not0001		0.000
existe<1>	18.864(R)	C4/_not0001		0.000
existe<2>	17.534(R)	C4/_not0001		0.000
existe<3>	17.160(R)	C4/_not0001		0.000
resto<0>	22.217(F)	C3/ctrl<0>		0.000
resto<1>	22.894(F)	C3/ctrl<0>		0.000
resto<2>	23.410(F)	C3/ctrl<0>		0.000
resto<3>	23.443(F)	C3/ctrl<0>		0.000
resto<4>	23.232(F)	C3/ctrl<0>		0.000
resto<5>	22.726(F)	C3/ctrl<0>		0.000
resto<6>	23.408(F)	C3/ctrl<0>		0.000
resto<7>	22.881(F)	C3/ctrl<0>		0.000

Clock refresco<2> to Pad

Destination	clk (edge) to PAD		Internal Clock(s)	Clock Phase
existe<0>	18.684(R)	C4/_not0001		0.000
existe<1>	18.543(R)	C4/_not0001		0.000
existe<2>	17.213(R)	C4/_not0001		0.000
existe<3>	16.839(R)	C4/_not0001		0.000
resto<0>	21.614(F)	C3/ctrl<0>		0.000
resto<1>	22.291(F)	C3/ctrl<0>		0.000
resto<2>	22.807(F)	C3/ctrl<0>		0.000
resto<3>	22.840(F)	C3/ctrl<0>		0.000
resto<4>	22.629(F)	C3/ctrl<0>		0.000
resto<5>	22.123(F)	C3/ctrl<0>		0.000
resto<6>	22.805(F)	C3/ctrl<0>		0.000
resto<7>	22.278(F)	C3/ctrl<0>		0.000

Clock refresco<3> to Pad

Destination	clk (edge) to PAD	Internal Clock(s)	Clock Phase
existe<0>	18.885(R)	C4/_not0001	0.000
existe<1>	18.744(R)	C4/_not0001	0.000
existe<2>	17.414(R)	C4/_not0001	0.000
existe<3>	17.040(R)	C4/_not0001	0.000
resto<0>	22.097(F)	C3/ctrl<0>	0.000
resto<1>	22.774(F)	C3/ctrl<0>	0.000
resto<2>	23.290(F)	C3/ctrl<0>	0.000
resto<3>	23.323(F)	C3/ctrl<0>	0.000
resto<4>	23.112(F)	C3/ctrl<0>	0.000
resto<5>	22.606(F)	C3/ctrl<0>	0.000
resto<6>	23.288(F)	C3/ctrl<0>	0.000
resto<7>	22.761(F)	C3/ctrl<0>	0.000

Clock to Setup on destination clock clk

Source Clock	Dest:Rise	Src:Fall Dest:Rise	Src:Rise Dest:Fall	Src:Fall Dest:Fall
clk	7.431			
refresco<0>	4.385	5.899		
refresco<1>	9.346	9.346		
refresco<2>	8.743	8.743		
refresco<3>	9.226	9.226		

Clock to Setup on destination clock refresco<0>

Source Clock	Dest:Rise	Src:Fall Dest:Rise	Src:Rise Dest:Fall	Src:Fall Dest:Fall
clk			3.728	
refresco<1>			-1.319	-1.319
refresco<2>			-1.089	-1.089
refresco<3>			-0.967	-0.967

Clock to Setup on destination clock refresco<1>

Source Clock	Dest:Rise	Src:Fall Dest:Rise	Src:Rise Dest:Fall	Src:Fall Dest:Fall
clk	3.750		3.728	
refresco<1>	3.633	-3.450	-1.407	-1.407
refresco<2>	3.633	-3.039	-1.178	-1.178
refresco<3>	3.633	-2.320	-1.055	-1.055

Clock to Setup on destination clock refresco<2>

Source Clock	Dest:Rise	Src:Fall Dest:Rise	Src:Rise Dest:Fall	Src:Fall Dest:Fall
clk	3.750		3.728	
refresco<1>	3.633	-2.976	-1.316	-1.316
refresco<2>	3.633	-2.565	-1.087	-1.087
refresco<3>	3.633	-1.846	-0.965	-0.965

Clock to Setup on destination clock refresco<3>

Source Clock	Dest:Rise	Src:Fall Dest:Rise	Src:Rise Dest:Fall	Src:Fall Dest:Fall
clk	3.750		3.728	
refresco<1>	3.633	-2.798	-1.036	-1.036

```
refresco<2>   |   3.633|  -2.387|  -0.807|  -0.807|
refresco<3>   |   3.633|  -1.668|  -0.684|  -0.684|
--------------+--------+--------+--------+--------+

Analysis completed Thu Feb 15 18:08:16 2007
--------------------------------------------------------------------------------


Peak Memory Usage: 88 MB
```

Para más información sobre los datos reportados, acudir a los manuales de la herramienta.

1.7.3. Consumo de potencia

Al ejecutar la aplicación de estimar el consumo de potencia, se obtiene el siguiente report. En dicha información conviene destacar que se desglosan las contribuciones de los bloques I/O (Vcco) y el resto (Vccint).

```
------------------------------------------------------------------
Release 8.2.03i - XPower SoftwareVersion:I.34
Copyright (c) 1995-2006 Xilinx, Inc.  All rights reserved.
Design:       E:\DOCTORADO\disenos\tutorial1\main.ncd
Preferences:  main.pcf
Part:         v50bg256-6
Data version: PRODUCTION,v1.0,05-28-03

XPower and Datasheet may have some Quiescent Current differences.
This is due to the fact that the quiescent numbers in XPower are based on
measurements of real designs with active functional elements reflecting real
world design scenarios.


Power summary:                    I(mA)    P(mW)
------------------------------------------------------------------
Total estimated power consumption:              27
                    ---
                Vccint 2.50V:       8       20
                Vcco33 3.30V:       2        7
                    ---
                  Clocks:       0       0
                  Inputs:       0       0
                   Logic:       0       0
                 Outputs:
                   Vcco33       0       0
                 Signals:       0       0
                    ---
      Quiescent Vccint  2.50V:       8       20
      Quiescent Vcco33  3.30V:       2        7

Thermal summary:
------------------------------------------------------------------
  Estimated junction temperature:               26C
            Ambient temp:  25C
              Case temp:  26C
              Theta J-A:  30C/W

Decoupling Network Summary:      Cap Range (uF)      #
------------------------------------------------------------
```

```
Capacitor Recommendations:
Total for      Vccint :                    12
                        470.0  - 1000.0 :    1
                        0.470  -  2.200 :    1
                       0.0470  - 0.2200 :    2
                       0.0100  - 0.0470 :    3
                       0.0010  - 0.0047 :    5
                           ---
Invalid Program Mode
Total for      Vcco33 :                     8
                        470.0  - 1000.0 :    1
                       0.0470  - 0.2200 :    1
                       0.0100  - 0.0470 :    2
                       0.0010  - 0.0047 :    4

Analysis completed: Thu Feb 15 22:00:15 2007
----------------------------------------------------------------------
```

Para más información sobre los datos reportados, acudir a los manuales de la herramienta.