Universidad Politécnica de Madrid

Grado en Ingeniería Electrónica y Automática

Asignatura

SISTEMAS ELECTRÓNICOS DIGITALES

TRABAJO VHDL

Máquina Expendedora

Curso 2021-2022

Julio Augusto Arbizu García 52070

Adrián Rodríguez Fernández 54829

ÍNDICE

Contenido

INTRODUCCIÓN

En este documento se expone el proceso que hemos seguido para desarrollar el trabajo en VHDL. Nuestro equipo ha elegido como tema una máquina expendedora. Hemos hecho algunas modificaciones a la propuesta inicial. Por ejemplo, para intentar acercarnos a la realidad, hemos desechado la necesidad de introducir el importe exacto, y lo que hacemos es que simulamos que la máquina devuelve la diferencia entre el precio del producto y el importe introducido. A parte de esto, hemos considerado que la máquina no solo tiene refrescos, sino que tiene una gran variedad de productos, y que el usuario introduce mediante un código BCD propio de cada producto lo que quiere comprar.

Para ejecutar el programa utilizaremos los diversos periféricos que nos suministra la placa NEXUS 4 DDR, tales como LEDs, displays de 7 segmentos, switches y pulsadores. Más adelante se explicará la función de cada uno de ellos.

Respecto a la división del trabajo entre los distintos componentes del grupo, al ser solo dos no hemos visto necesario hacer el reparto de tareas como se aconseja (arquitecto del sistema, programador, responsable de calidad), sino que nos hemos ido dividiendo las tareas equitativamente, así que cada uno de los dos ha trabajado en los tres roles indistintamente, realizando los sources, testbenches, control de versiones, etc.

A continuación, se expone el link al repositorio git utilizado:

https://github.com/JulioArbizu/Trabajo-VHDL

DISEÑO DEL SISTEMA

Respecto al diseño utilizado, se realizaron una serie de bocetos que fueron evolucionando según fuimos entendiendo sus limitaciones y los objetivos que teníamos que cumplir. El primero de ellos fue el siguiente:

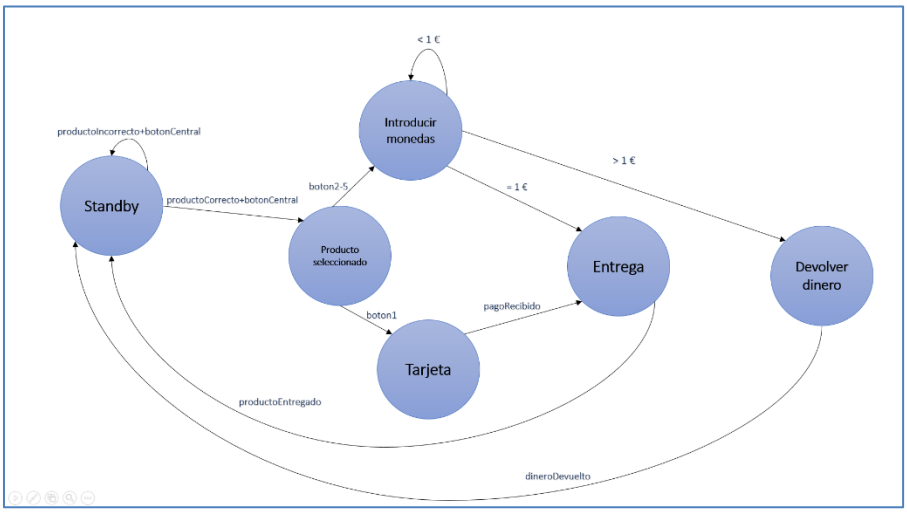

Ilustración 1. Primer boceto realizado

Según intentamos a empezar a trabajar con él nos dimos cuenta de que íbamos a crear una super máquina de estados que se encargaba de todas y cada una de las funcionalidades del diseño, por lo que no tardamos en descartarlo. Asimismo, también descartamos el pago con tarjeta.

Ya en el segundo diseño, y siguiendo las recomendación del profesor de dividir el sistema según sus funcionalidades creamos el siguiente diagrama:

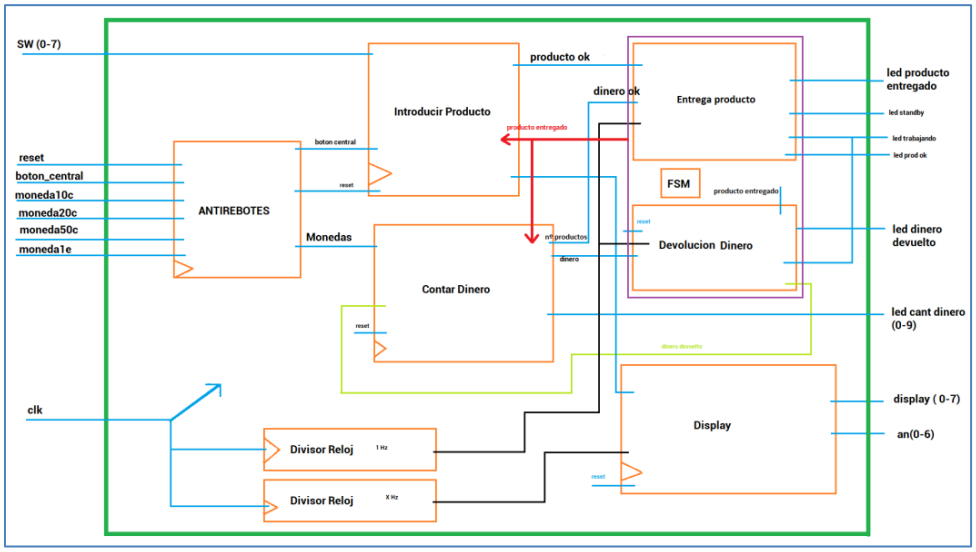

Ilustración 2. Boceto de diagrama de funcionalidades

En esta segunda iteración ya se ven unidades que se mantendrían hasta el diseño final como *Introducir Producto*, *Antirrebotes*, *Contar Dinero* y *Display*. A la derecha, recuadrado en morado, está una entidad que más tarde se llamó *máquina*, que funciona como el cerebro de la máquina expendedora, ya que en su interior estaría la máquina de estados y algunas funcionalidades más.

Como podemos observar, las distintas entidades se encargan de cumplir una función, y transmitir la información necesaria a las siguientes etapas. A modo de ejemplo, el bloque *Introducir Producto* se encarga de recibir la entrada de los *switches* y comprobar que el código *BCD* introducido es correcto, y si lo es, lo envía a el bloque morado que lo usará en la máquina de estados. Más adelante se estudiará cada uno de los bloques en profundidad.

Respecto al bloque *máquina*, se realizó un primer diagrama de bloques interno que es el siguiente:

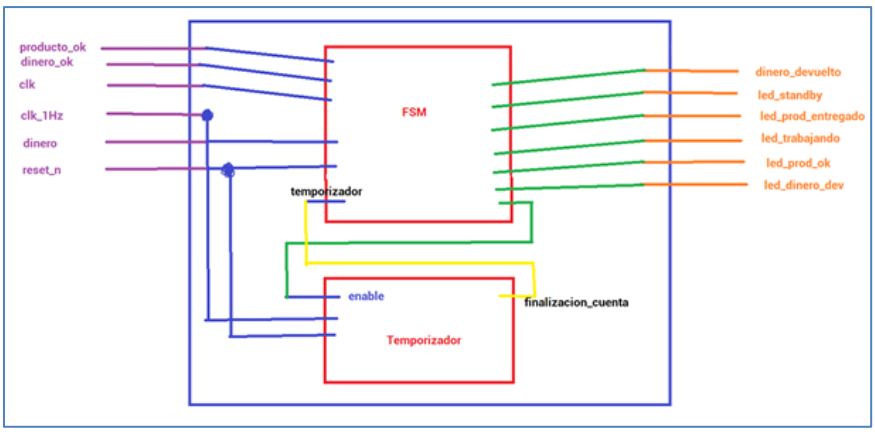

Ilustración 3 Bloque máquina

En él podemos observar las entradas y salidas del bloque, que después serían modificadas pero que nos sirvieron como primera aproximación. También es interesante ver cómo se utiliza el temporizador con un esquema *maestro-esclavo*, ya que el temporizador empezaría a contar solo cuando la máquina de estados se lo indica. Finalmente, en el proyecto final utilizamos dos temporizadores, uno de 2 segundos y otro de 4 segundos, pero gracias a la generalización sólo creamos una entidad, que personalizamos para cada tipo. En el esquema también observamos una salida de *producto_entregado* que finalmente no nos hizo falta utilizar.

Con respecto a la máquina de estados, realizamos el siguiente diagrama de estados, que representa la secuencia a realizar para sacar un producto de la máquina:

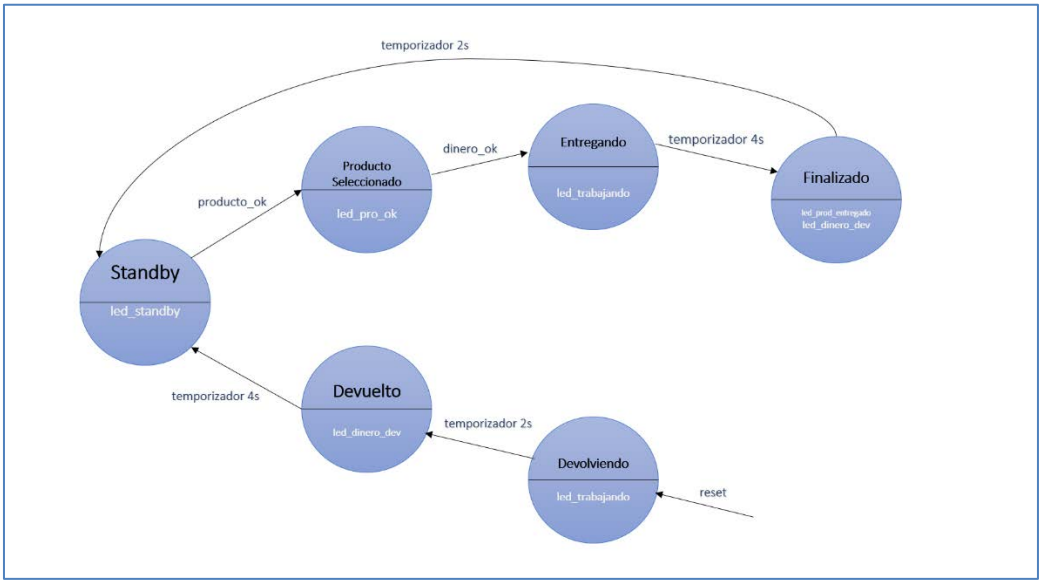

Ilustración 4. Máquina de Estados

También podemos ver que, si se pulsa el botón de *reset*, se devuelve el dinero que tuviera la máquina dentro, al igual que se devuelve el dinero que supere el precio del producto, aunque no se pulse el botón de *reset*. Finalmente, el diagrama de bloques quedó de la siguiente manera:

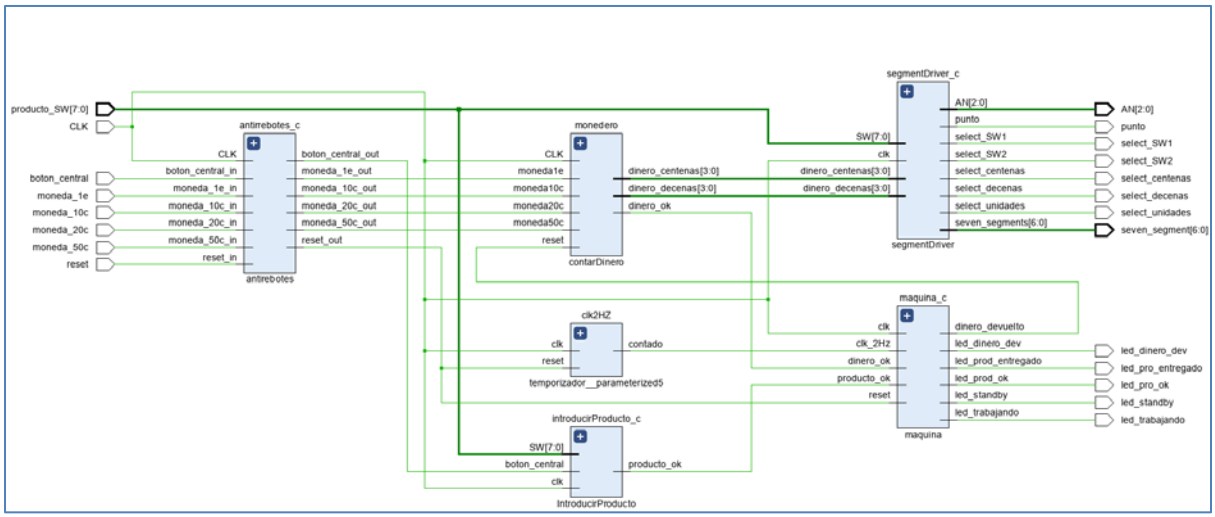

Ilustración 5. Esquematico del proyecto

Como podemos observar, se han mantenido prácticamente todos los bloques, pero se han modificado algunas entradas y salidas de estos que finalmente no fueron necesarias. Podemos observar

claramente las funcionalidades de cada bloque, por ejemplo, el bloque *display* recibe la información y la procesa para sacarla por los segmentos, o como los bloques de introducir producto o contar dinero le pasan la información a la máquina de estado de cuando el producto o el dinero es correcto.

CASOS DE USO

Los casos de uso son los siguientes:

- El sistema diseñado será una máquina expendedora de diversos productos.
- Esta máquina podrá tener hasta un máximo de 39 productos a elegir.
- La selección de estos productos se hará mediante 8 *SW* con los cuales se seleccionará el producto en cuestión con un formato numérico de entrada *BCD*.
- Una vez introducido un valor entre 01 y 39 se pulsará el botón seleccionar producto.
- A continuación, será el momento de introducir las monedas mediante los diferentes botones.
- Tanto el dinero introducido como el producto seleccionado se mostrará a través del *display* de 7 segmentos.
- Si el dinero es igual o superior al precio del producto la máquina entregará el producto.
- Una vez entregado el producto devolverá el dinero sobrante y se podrá seleccionar otro producto.

FUNCIONAMIENTO

La máquina se queda en *standby* hasta que se selecciona un producto correctamente, que utiliza los 8 *SW*, donde los 4 primeros serán para las decenas del número, y los 4 siguientes para las unidades. Este número deberá ser en formato *BCD* y comprendido entre 01 y 39. Una vez seleccionado el producto se tiene que pagar. Todos los productos valen un euro, pero se puede introducir más cantidad, pues el resto se devolverá. Se pueden introducir monedas de 10, 20, 50 céntimos y de 1 euro.

Una vez seleccionado el producto y realizado el pago, la máquina tarda 4 segundos en entregar el producto y devolver el dinero sobrante. Tras 2 segundos, la máquina vuelve al *standby*, donde espera al siguiente usuario.

Si se pulsa el botón de *reset* la máquina devolverá el dinero tras 2 segundos de trabajo. Una vez ahí, vuelve al *standby* tras 4 segundos.

La máquina consta de varios *LEDs*, que indican al usuario el estado de la máquina, tiene un *led* de *standby*, otro para indicar que se ha seleccionado bien el producto, otro para indicar que está trabajando (entregando producto o devolviendo dinero) y otros dos para mostrar que se ha entregado el producto y/o devuelto el dinero.

CONSTRAINTS

En el archivo *contraints* de nuestro proyecto indicaremos las *E/S* que utilizaremos e inicializamos estos puertos para poder ser usados por nuestro programa. Estos puertos son los siguientes:

1. CLOCK SIGNAL

- CLK: Reloj interno de la placa.

2. SWITCHES

- producto_SW [3 downto 0]: Estos switches se usarán para seleccionar la entrada del dígito unidades del producto seleccionado.
- producto_SW [7 downto 4]: Estos switches se usarán para seleccionar la entrada del dígito decenas del producto seleccionado.

3. LEDs

- led_standby: Indica que la máquina está en el punto de *standby*.
- led_pro_ok: Indica que el valor de producto seleccionado con los *SW* está dentro de los valores correctos.
- led_trabajando: Indica que la máquina está trabajando, es decir, esta entregando el producto y/o el dinero.
- led_pro_entregado: Indica que el producto seleccionado ha sido entregado.
- led_dinero_dev: Indica que el cambio del dinero introducido se ha devuelto y el contador del monedero está a 0.

La disposición de estos LEDs en la placa es la mostrada en la siguiente imagen.

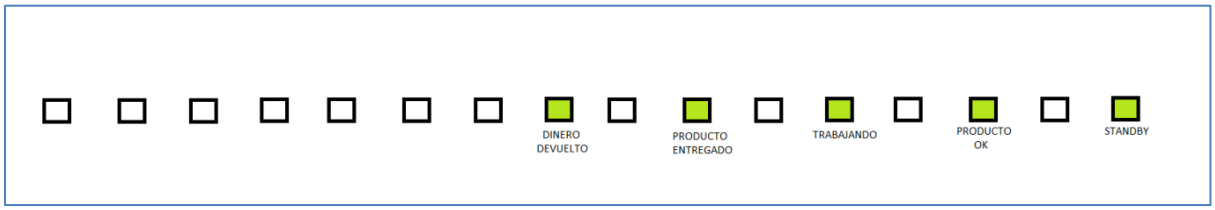

Ilustración 6. Disposición de los LEDs

4. SEGMENT DISPLAY

- seven_segment [7 downto 0]: Segmentos del *display*.
- Punto: Punto del display que indica el punto decimal en el valor del monedero.
- select_unidades: Ánodo del valor de unidades de céntimos, este *display* siempre mostrará un 0.
- select_decenas: Ánodo del valor de decenas de céntimos.
- select_centenas: Ánodo del valor de unidades de céntimos.
- AN[2 downto 0]: Ánodos que desactivan los *displays* que no se utilizan.
- select_SW1: Ánodo del segundo dígito (unidades) del producto seleccionado en BCD.
- select_SW2: Ánodo del primer dígito (decenas) del producto seleccionado en BCD.

La posición de estos en la placa se muestra en la siguiente imagen.

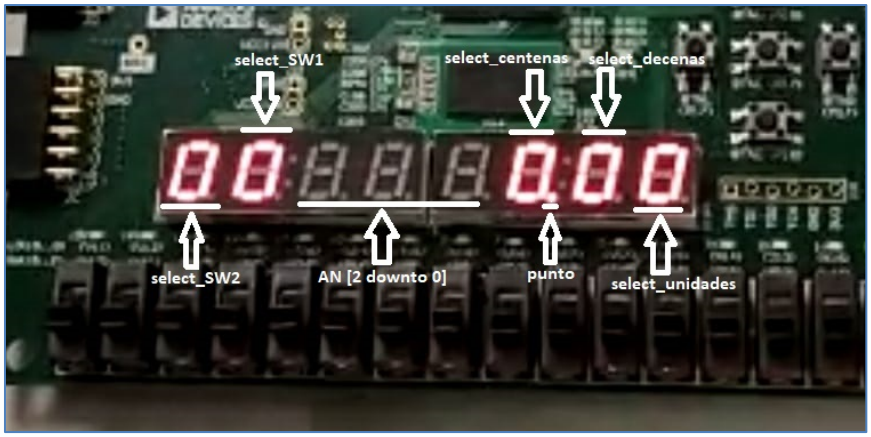

Ilustración 7. Distribución de los displays

5. BUTTONS

- reset: Pone la maquina al estado devolviendo que devuelve el dinero introducido antes de volver al *standby*.
- boton_central: Si se pulsa se comprueba que el producto seleccionado es el correcto.
- moneda_10c: Introduce una moneda de 10 céntimos.
- moneda_20c: Introduce una moneda de 20 céntimos.
- moneda_50c: Introduce una moneda de 50 céntimos.
- moneda_1e: Introduce una moneda de 1 euro.

BLOQUES FUNCIONALES

1. Bloque Introducir Producto

El código completo es de la entidad es:

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

entity IntroducirProducto is
    Port (
    clk : in std_logic;                                 -- Señal de reloj
    boton_central : in std_logic;                       -- Boton central, señal que
llega desde el bloque antirebotes
    SW : in std_logic_vector(7 downto 0);               -- Switches
    producto_ok : out std_logic                        -- 1 si el producto es
correcto, 0 si no es codigo BCD correcto
    );
end IntroducirProducto;

architecture Behavioral of IntroducirProducto is
    signal comprobacion_ok : std_logic;
begin

    comprobacion : process(sw)
        variable comprobacion1, comprobacion2 : boolean := false;
    begin
        --4 primeros bits--
        case(SW(7 downto 4)) is
            -- Comprobamos si los primeros cuatro digitos corresponden a un
numero en BCD --
```

```
            when "0000" | "0001" | "0010" | "0011" =>
                comprobacion1 := true;
            when others =>
                comprobacion1 := false;
        end case;

        --4 siguientes bits--
        case(SW(3 downto 0)) is
            -- Comprobamos si los siguientes cuatro digitos corresponden a un
numero en BCD --
            when "0000" | "0001" | "0010" | "0011" | "0100" | "0101" | "0110" |
"0111" | "1000" | "1001" =>
                comprobacion2 := true;
            when others =>
                comprobacion2 := false;
        end case;
        -- Si ambas comprobaciones son correctas el codigo BCD introducido es
válido
        if (comprobacion1 and comprobacion2) then
            if (SW /= "00000000") then
            comprobacion_ok <= '1';
            end if;
        else
            comprobacion_ok <= '0';
        end if;
    end process;

    salidas : process (clk,boton_central)
    begin
        --Hacemos la logica de la salida producto_ok--
            if (boton_central = '1' and comprobacion_ok = '1') then
            producto_ok <= '1';
            else
            producto_ok <= '0';
            end if;
    end process;

end Behavioral;
```

El bloque tiene que comprobar si la combinación introducida en los *switches* es un código *BCD* válido, para ello, hacemos un proceso con dos variables booleanas. Primero comprobamos si los 4 primeros *bits* son correctos con un case, en este caso, supusimos que la máquina sólo tendría 39 productos, por lo que introducir un 4 o superior se da como código incorrecto. En caso de que sea válido, la variable *comprobacion1* se pone a *true*. Se repite este mismo mecanismo para los 4 siguientes bits, esta vez considerando que se puede llegar hasta el número 9, y se pone el resultado en la variable *comprobacion2*. Por último, si ambas comprobaciones son correctas se pone a '1' la señal *comprobacion_ok*.

Después en otro proceso, si la comprobación es correcta y se pulsa el botón central, se pone a '1' la salida del bloque *producto_ok*.

2. Bloque Antirrebotes

Este bloque se encarga de gestionar todas las entradas asíncronas del sistema, básicamente los pulsadores. Este bloque tiene una arquitectura estructural, ya que se encarga de unir las dos entidades que forman el bloque. El código es el siguiente:

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


entity antirebotes is
    port(
    ------------------------------ ENTRADAS ---------------------------------------
        CLK                      : in std_logic;                      -- Señal de
reloj
        reset_in             : in std_logic;                    -- Entrada reset
activa a nivel bajo
        boton_central_in       : in std_logic;                      -- Utilaza para
seleccion del producto y modo de pago con tarjeta
        moneda_10c_in          : in std_logic;                      -- Moneda 10
centimos
        moneda_20c_in          : in std_logic;                      -- Moneda 20
centimos
        moneda_50c_in          : in std_logic;                      -- Moneda 50
centimos
        moneda_1e_in           : in std_logic;                      -- Moneda 1 euro
        --------------------------- SALIDAS ----------------------------------------
-
        reset_out             : out std_logic;
        boton_central_out       : out std_logic;
        moneda_10c_out          : out std_logic;
        moneda_20c_out          : out std_logic;
        moneda_50c_out          : out std_logic;
        moneda_1e_out           : out std_logic

    );
end antirebotes;

architecture Structural of antirebotes is
    COMPONENT EDGEDTCTR is
        port (
            CLK : in std_logic;
            SYNC_IN : in std_logic;
            EDGE : out std_logic
     );
    end COMPONENT;

    COMPONENT SYNCHRNZR is
    port (
        CLK : in std_logic;
        ASYNC_IN : in std_logic;
        SYNC_OUT : out std_logic
    );
    end COMPONENT;
    signal reset, boton_central, moneda_10c, moneda_20c, moneda_50c, moneda_1e :
std_logic;
begin
    snc_reset : SYNCHRNZR PORT MAP (clk=>clk, ASYNC_IN=>reset_in, SYNC_OUT=>reset);
    edge_reset : EDGEDTCTR PORT MAP(clk=>clk, SYNC_IN=>reset, EDGE=>reset_out);

    snc_btn_cen : SYNCHRNZR PORT MAP (clk=>clk, ASYNC_IN=>boton_central_in,
SYNC_OUT=>boton_central);
    edge_btc_cen : EDGEDTCTR PORT MAP(clk=>clk, SYNC_IN=>boton_central,
EDGE=>boton_central_out);
```

```
    snc_m10c : SYNCHRNZR PORT MAP (clk=>clk, ASYNC_IN=>moneda_10c_in,
SYNC_OUT=>moneda_10c);
    edge_m10c : EDGEDTCTR PORT MAP(clk=>clk, SYNC_IN=>moneda_10c,
EDGE=>moneda_10c_out);

    snc_m20c : SYNCHRNZR PORT MAP (clk=>clk, ASYNC_IN=>moneda_20c_in,
SYNC_OUT=>moneda_20c);
    edge_m20c : EDGEDTCTR PORT MAP(clk=>clk, SYNC_IN=>moneda_20c,
EDGE=>moneda_20c_out);

    snc_m50c : SYNCHRNZR PORT MAP (clk=>clk, ASYNC_IN=>moneda_50c_in,
SYNC_OUT=>moneda_50c);
    edge_m50c : EDGEDTCTR PORT MAP(clk=>clk, SYNC_IN=>moneda_50c,
EDGE=>moneda_50c_out);

    snc_m1e : SYNCHRNZR PORT MAP (clk=>clk, ASYNC_IN=>moneda_1e_in,
SYNC_OUT=>moneda_1e);
    edge_m1e : EDGEDTCTR PORT MAP(clk=>clk, SYNC_IN=>moneda_1e,
EDGE=>moneda_1e_out);
end Structural;
```

Como podemos ver, este bloque tiene como entradas los pulsadores, que entran a los sincronizadores correspondientes, y la salida de estos es la entrada de los detectores de flanco, cuya salida ya se corresponde con la salida del bloque principal.

Sincronizador

El código es el siguiente:

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

entity SYNCHRNZR is
    port (
        CLK : in std_logic;
        ASYNC_IN : in std_logic;
        SYNC_OUT : out std_logic
 );
end SYNCHRNZR;

architecture BEHAVIORAL of SYNCHRNZR is
    signal sreg : std_logic_vector(1 downto 0);
begin
    process (CLK)
    begin
        if rising_edge(CLK) then
            sync_out <= sreg(1);
            sreg <= sreg(0) & async_in;
        end if;
    end process;
end BEHAVIORAL;
```

Como su nombre indica, este bloque sincroniza la entrada asíncrona con un reloj, para poder tratarla como síncrona utilizando un registro de desplazamiento

Detector de flancos

Su código es el siguiente:

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

entity EDGEDTCTR is
    port (
        CLK : in std_logic;
        SYNC_IN : in std_logic;
        EDGE : out std_logic
 );
end EDGEDTCTR;

architecture BEHAVIORAL of EDGEDTCTR is
    signal sreg : std_logic_vector(2 downto 0);
begin
    process (CLK)
    begin
        if rising_edge(CLK) then
            sreg <= sreg(1 downto 0) & SYNC_IN;
        end if;
    end process;

    with sreg select
    EDGE <= '1' when "100",
    '0' when others;
end BEHAVIORAL;
```

El detector de flancos se encarga de que no se produzcan rebotes, y solo detectar el primer flanco de subida del pulsador. Para ello también lo hace con un registro de desplazamiento que va registrando la entrada y que hace que la salida sea '1' solo cuando este vale "100", es decir, en el primer flanco de subida de la señal.

3. Bloque Máquina

El bloque máquina es de tipo estructural, y se encarga de unir los diferentes componentes, en este caso, una máquina de estados y dos temporizadores (de 2 y 4 segundos) que son esclavos de esta, ya que tienen como entrada *reset* una salida de la *fsm*. El código de la entidad completa es el siguiente:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

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

entity maquina is
    Port (
    producto_ok : in std_logic;
    dinero_ok : in std_logic;
    clk : in std_logic;
    clk_2Hz : in std_logic;
    reset : in std_logic;
    dinero_devuelto : out std_logic;
    led_standby : out std_logic;
    led_prod_entregado : out std_logic;
    led_trabajando : out std_logic;
```

```
    led_prod_ok : out std_logic;
    led_dinero_dev : out std_logic
    );
end maquina;

architecture Structural of maquina is
      signal temporizador2s, temporizador4s : std_logic;
      signal reset_temp2s, reset_temp4s : std_logic;
      signal dinero_devuelto_o : std_logic;
      COMPONENT temporizador is
      generic(
        MODULO : positive
      );
      Port (
        clk       : in std_logic;  --Reloj
        reset   : in std_logic;  --Entrada reset asíncrona
        contado   : out std_logic  --Salida del temporizador, 1 cuando se acaba
la cuenta
      );
    end COMPONENT;

    COMPONENT fsm is
    port (
        CLK                     : in std_logic;                   -- Señal de
reloj
        reset                 : in std_logic;                  -- Entrada reset
activa a nivel bajo
        producto_ok             : in std_logic;                   -- Entrada de
que el producto se ha seleccionado correctamente
        dinero_ok               : in std_logic;                   -- Entrada de
que se ha superado 1 euro
        temporizador2s          : in std_logic;                   -- Entrada del
temporizador
        temporizador4s          : in std_logic;                   -- Entrada del
temporizador
        dinero_devuelto         : out std_logic;                  -- Salida que
reprenta que se ha devuelto el producto
        reset_temporizador2s    : out std_logic;                  -- Salida para
activar el temporizador de 2 segundos
        reset_temporizador4s    : out std_logic;                  -- Salida para
activar el temporizador de 4 segundos
        led_pro_entregado       : out std_logic;                  -- true
producto entregado, 0 producto no entregado
        led_pro_ok              : out std_logic;                  -- true
producto elegido correctamente
        led_trabajando          : out std_logic;                  -- true si la
maquina está procesando, o bien esperando al pago, devolviendo o entregando
producto
        led_dinero_dev          : out std_logic;                  -- true si se
ha devuelto correctamente el pago
        led_standby             : out std_logic                   -- true si se
esta en el estado de standby
     );
    end COMPONENT;

begin

    temporizador2segundos : temporizador
    generic map(
        MODULO => 5
    )
    port map (
        clk => clk_2Hz,
        reset => reset_temp2s,
        contado => temporizador2s
    );
    temporizador4segundos : temporizador
    generic map(
```

```
        MODULO => 9
    )
    port map (
        clk => clk_2Hz,
        reset => reset_temp4s,
        contado => temporizador4s
    );
    fsm_maquina : fsm port map (
        CLK => clk,
        reset => reset,
        producto_ok => producto_ok,
        dinero_ok => dinero_ok,
        temporizador2s => temporizador2s,
        temporizador4s => temporizador4s,
        dinero_devuelto => dinero_devuelto,
        reset_temporizador2s => reset_temp2s,
        reset_temporizador4s => reset_temp4s,
        led_pro_entregado => led_prod_entregado,
        led_pro_ok => led_prod_ok,
        led_trabajando => led_trabajando,
        led_dinero_dev => led_dinero_dev,
        led_standby => led_standby
     );

end;
```

FSM

Respecto a la máquina de estados, su código es el siguiente:

```
library IEEE;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_1164.all;

entity fsm is
    port (
        CLK                     : in std_logic;                 -- Señal de
reloj
        reset                   : in std_logic;                 -- Entrada
reset activa a nivel alto
        producto_ok             : in std_logic;                 -- Entrada de
que el producto se ha seleccionado correctamente
        dinero_ok               : in std_logic;                 -- Entrada de
que se ha superado 1 euro
        temporizador2s          : in std_logic;                 -- Entrada del
temporizador
        temporizador4s          : in std_logic;                 -- Entrada del
temporizador
        dinero_devuelto         : out std_logic;                -- Salida que
reprenta que se ha devuelto el producto
        reset_temporizador2s    : out std_logic;                -- Salida para
activar el temporizador de 2 segundos
        reset_temporizador4s    : out std_logic;                -- Salida para
activar el temporizador de 4 segundos
        led_pro_entregado       : out std_logic;                -- true
producto entregado, 0 producto no entregado
        led_pro_ok              : out std_logic;                -- true
producto elegido correctamente
        led_trabajando          : out std_logic;                -- true si la
maquina está procesando, o bien esperando al pago, devolviendo o entregando
producto
        led_dinero_dev          : out std_logic;                -- true si se
ha devuelto correctamente el pago
        led_standby             : out std_logic                 -- true si se
esta en el estado de standby
```

```
 );
end fsm;

architecture behavioral of fsm is
    type STATES is (S0_Standby, S1_ProdSelecc, S2_Entregando, S3_Finalizado,
S4_Devolviendo, S5_Devuelto);
    signal current_state: STATES := S0_Standby;
    signal next_state: STATES;
begin
    state_register: process (reset, CLK)
    begin
        if reset = '1' then
            current_state <= S4_Devolviendo;
        elsif rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process;

    nextstate_decod: process (producto_ok,dinero_ok,temporizador2s,
temporizador4s, current_state)
    begin
        next_state <= current_state;
        case current_state is
           when S0_Standby =>
             if producto_ok = '1' then
                next_state <= S1_ProdSelecc;
             end if;
           when S1_ProdSelecc =>
             if dinero_ok = '1' then
                next_state <= S2_Entregando;
             end if;
           when S2_Entregando =>
             if temporizador4s = '1' then
                next_state <= S3_Finalizado;
             end if;
           when S3_Finalizado =>
             if temporizador2s = '1' then
                next_state <= S0_Standby;
             end if;
            when S4_Devolviendo =>
             if temporizador2s = '1' then
                next_state <= S5_Devuelto;
             end if;
            when S5_Devuelto =>
             if temporizador4s = '1' then
                next_state <= S0_Standby;
             end if;
           when others =>
            next_state <= S4_Devolviendo;
        end case;
    end process;

    output_decod: process (current_state)
    begin
        case current_state is
         when S0_Standby =>
            dinero_devuelto <= '0';
            reset_temporizador2s <= '1';
            reset_temporizador4s <= '1';
            led_pro_entregado <= '0';
            led_pro_ok <= '0';
            led_trabajando <= '0';
            led_dinero_dev <= '0';
            led_standby <= '1';
         when S1_ProdSelecc =>
            dinero_devuelto <= '0';
            reset_temporizador2s <= '1';
            reset_temporizador4s <= '1';
```

```
            led_pro_entregado <= '0';
            led_pro_ok <= '1';
            led_trabajando <= '0';
            led_dinero_dev <= '0';
            led_standby <= '0';
         when S2_Entregando =>
            dinero_devuelto <= '0';
            reset_temporizador2s <= '1';
            reset_temporizador4s <= '0';
            led_pro_entregado <= '0';
            led_pro_ok <= '0';
            led_trabajando <= '1';
            led_dinero_dev <= '0';
            led_standby <= '0';
         when S3_Finalizado =>
            dinero_devuelto <= '1';
            reset_temporizador2s <= '0';
            reset_temporizador4s <= '1';
            led_pro_entregado <= '1';
            led_pro_ok <= '0';
            led_trabajando <= '0';
            led_dinero_dev <= '1';
            led_standby <= '0';
         when S4_Devolviendo =>
            dinero_devuelto <= '0';
            reset_temporizador2s <= '0';
            reset_temporizador4s <= '1';
            led_pro_entregado <= '0';
            led_pro_ok <= '0';
            led_trabajando <= '1';
            led_dinero_dev <= '0';
            led_standby <= '0';
        when S5_Devuelto =>
            dinero_devuelto <= '1';
            reset_temporizador2s <= '1';
            reset_temporizador4s <= '0';
            led_pro_entregado <= '0';
            led_pro_ok <= '0';
            led_trabajando <= '0';
            led_dinero_dev <= '1';
            led_standby <= '0';
         when others =>
            dinero_devuelto <= '1';
            reset_temporizador2s <= '1';
            reset_temporizador4s <= '1';
            led_pro_entregado <= '1';
            led_pro_ok <= '1';
            led_trabajando <= '1';
            led_dinero_dev <= '1';
            led_standby <= '1';
        end case;
    end process;
end behavioral;
```

Esta sigue el esquema básico, con tres procesos, el primero de ellos actualiza el estado en los flancos de subida del reloj, o cuando se pulsa el botón de *reset*, que pone el estado de *S4_Devolviendo* ya que la máquina expendedora devuelve el dinero cuando se pulsa el botón de *reset*.

El siguiente proceso es el decodificador de siguiente estado, que evalúa cuando se produce la transición al siguiente estado desde el actual. Por ejemplo, si se está en *standby*, si llega la señal desde el bloque Introducir Producto de que se ha seleccionado producto correctamente, se pasa al siguiente estado *S1_ProdSelecc,* y así sucesivamente.

Por último, se encuentra el decodificador de las salidas de la máquina de estados, que activa unas u otras según el estado actual. Cabe destacar como salidas los *reset_temporizadorXs*, que se

encargan de desactivar los temporizadores correspondientes en los respectivos estados, y los activan en caso de que sean necesarios.

4. Temporizador

El código del temporizador es el siguiente:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

entity temporizador is
  generic(
  MODULO : positive
  );
  Port (
  clk       : in std_logic;  --Reloj
  reset    : in std_logic;  --Entrada reset asíncrona
  contado   : out std_logic  --Salida del temporizador, 1 cuando se acaba la
cuenta
  );
end temporizador;

architecture Behavioral of temporizador is

begin
process(reset, clk)
    subtype count_t is natural range 0 to MODULO - 1;
    variable count : count_t;
begin
    if reset = '1' then
        count := 0;
    elsif rising_edge (clk) then
        count := (count+1)mod MODULO;
    end if;

    -- Si la cuenta llega a MODULO - 1 significa que ha terminado de contar los
ciclos de reloj necesarios --
    if (count = MODULO - 1) then
        contado <= '1';
    else
        contado <= '0';
    end if;
end process;
end Behavioral;
```

En él podemos ver como básicamente es un contador que suma uno en cada flanco de subida del reloj, y cuando se llega a cierta cantidad, se pone a 0 el contador y la salida vale uno. Programando adecuadamente el módulo de la cuenta, se puede generar una señal cada X segundos. En nuestro caso, la utilizamos para crear un reloj de 2 Hz que hace de entrada al bloque máquina, para ser utilizada por dos temporizadores internos de forma síncrona para crear los tiempos de espera de 2 y 4 segundos. Por otra parte, también fue usada para refrescar el *display* a una frecuencia de 10 kHz.

5. Bloque segmentDriver

Este bloque está formado por la entidad superior y otros componentes. Los componentes internos son los siguientes:

clock10kHz

Divisor de reloj a 10 kHz. Se utiliza la entidad temporizador anteriormente mencionada. Ponemos el MODULE a 10000 para generar esa frecuencia.

decoder7segmentes

Es un decodificador tradicional cuya función es decidir qué segmentos del *display* enciende y cuáles no para mostrar valores numéricos en base decimal.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY decoder7segments IS
    PORT (
        code : IN std_logic_vector(3 DOWNTO 0);
        led : OUT std_logic_vector(6 DOWNTO 0)
    );
END ENTITY decoder7segments;

ARCHITECTURE dataflow OF decoder7segments IS
BEGIN
    WITH code SELECT
        led <=  "1000000" WHEN "0000",
                "1111001" WHEN "0001",
                "0100100" WHEN "0010",
                "0110000" WHEN "0011",
                "0011001" WHEN "0100",
                "0010010" WHEN "0101",
                "0000010" WHEN "0110",
                "1111000" WHEN "0111",
                "0000000" WHEN "1000",
                "0010000" WHEN "1001",
                "0111111" WHEN others;
END ARCHITECTURE dataflow;
```

Entidad segmentDriver

Su código se muestra en la siguiente imagen:

```
entity segmentDriver is
  Port (
  SW : in std_logic_vector(7 downto 0);
  dinero_decenas : in std_logic_vector(3 downto 0);
  dinero_centenas : in std_logic_vector(3 downto 0);
  clk : in std_logic;
  seven_segments : out std_logic_vector(6 downto 0);
  punto : out std_logic;
  select_SW1 : out std_logic;
  select_SW2 : out std_logic;
  select_decenas : out std_logic;
  select_centenas : out std_logic;
  select_unidades : out std_logic;
```

```
  AN : out std_logic_vector(2 DOWNTO 0)
  );
end segmentDriver;

architecture Behavioral of segmentDriver is

    COMPONENT decoder7segments IS
        PORT (
            code : IN std_logic_vector(3 DOWNTO 0);
            led : OUT std_logic_vector(6 DOWNTO 0)
        );
    END COMPONENT decoder7segments;

    COMPONENT temporizador is
      generic(
      MODULO : positive
      );
      Port (
      clk       : in std_logic;  --Reloj
      reset     : in std_logic;  --Entrada reset asíncrona
      contado   : out std_logic  --Salida del temporizador, 1 cuando se acaba la
cuenta
      );
    end COMPONENT;

    signal temperary_data : std_logic_vector(3 downto 0);
    signal clk_10kHz : std_logic;

begin

    decoder7segments_uut : decoder7segments port map(
        code => temperary_data,
        led => seven_segments
    );

    clock10kHz : temporizador generic map(
        MODULO => 10000
    )
    port map(
        clk => clk,
        reset => '0',
        contado => clk_10kHz
    );

    process(clk_10kHz)
        variable seleccion_display : std_logic_vector(2 downto 0);
    begin
        if rising_edge(clk_10kHz) then
            case seleccion_display is

            when "000" => temperary_data <= SW(3 downto 0);
                select_SW1 <= '0';
                select_SW2 <= '1';
                select_decenas <= '1';
                select_centenas <= '1';
                select_unidades <= '1';
                punto <= '1';
                AN <= "111";
               seleccion_display := seleccion_display + '1';

            when "001" => temperary_data <= SW(7 downto 4);
                select_SW1 <= '1';
                select_SW2 <= '0';
                select_decenas <= '1';
                select_centenas <= '1';
                select_unidades <= '1';
                punto <= '1';
                AN <= "111";
```

```
                seleccion_display := seleccion_display + '1';

            when "010" => temperary_data <= "0000";
                select_SW1 <= '1';
                select_SW2 <= '1';
                select_decenas <= '1';
                select_centenas <= '1';
                select_unidades <= '0';
                punto <= '1';
                AN <= "111";
               seleccion_display := seleccion_display + '1';

            when "011" => temperary_data <= dinero_decenas;
                select_SW1 <= '1';
                select_SW2 <= '1';
                select_decenas <= '0';
                select_centenas <= '1';
                select_unidades <= '1';
                punto <= '1';
                AN <= "111";
               seleccion_display := seleccion_display + '1';

            when "100" => temperary_data <= dinero_centenas;
                select_SW1 <= '1';
                select_SW2 <= '1';
                select_decenas <= '1';
                select_centenas <= '0';
                select_unidades <= '1';
                punto <= '0';
                AN <= "111";
               seleccion_display := "000";

            when others => temperary_data <= "1111";
                select_SW1 <= '0';
                select_SW2 <= '0';
                select_decenas <= '0';
                select_centenas <= '0';
                select_unidades <= '0';
                punto <= '1';
                AN <= "111";
             seleccion_display := "000";
          end case;
        end if;
    end process;

end Behavioral;
```

Este bloque se encarga de activar o desactivar los ánodos de los diferentes SW. Realmente lo que hace es apagar todos menos uno e ir cambiando el *display* a encender de forma sucesiva, sin embargo, como la velocidad de reloj es de 10 kHz el ojo humano cree que todos están encendidos a la vez. Es un buen sistema para controlarlos individualmente y poder enviar valores distintos por cada uno de ellos.

6. Bloque ContarDinero

El código es el siguiente:

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

```
entity contarDinero is
       port (
      CLK:        in std_logic;
      reset:      in std_logic;
      moneda10c:  in std_logic;
      moneda20c:  in std_logic;
      moneda50c:  in std_logic;
      moneda1e:   in std_logic;
         dinero_ok:  out std_logic;
      dinero_decenas:      out std_logic_vector (3 downto 0);
      dinero_centenas:     out std_logic_vector (3 downto 0)
    );

end contarDinero;

architecture BEHAVIORAL of contarDinero is

begin



   process(reset,CLK)
    variable cont : integer :=0;
    variable centenas  : integer :=0;
    variable decenas  : integer :=0;
    variable aux : integer  :=0;
       begin
            if reset='1' then
            cont := 0;
            elsif rising_edge(CLK) then
                if moneda10c = '1' then
                cont := cont +1;
                elsif  moneda20c = '1' then
                cont := cont +2;
                elsif  moneda50c = '1' then
                cont := cont +5;
                elsif  moneda1e = '1' then
                cont := cont +10;
                end if;
            end if;

               if cont > 10 then
               dinero_ok <= '1';
               else
               dinero_ok <= '0';
               end if ;

   aux := cont;
   decenas := aux mod 10;
   aux := aux/10;
   centenas := aux mod 10;

    dinero_decenas   <= std_logic_vector(to_unsigned(decenas
,dinero_decenas'length));
    dinero_centenas  <=
std_logic_vector(to_unsigned(centenas,dinero_centenas'length));
    end process;

end BEHAVIORAL;
```

Este bloque se encarga de contar el dinero introducido a través de cuatro variables. En función del valor de la moneda introducida incrementa un contador ese valor partido de diez, una vez introducidos varios valores divide contador entre diez para separar centenas y decenas y almacena estos valores. Además, si el valor del contador es mayor que 10 envía la señal *dinero_ok*.

7. Bloque TOP

El bloque top se encarga de unir todos los componentes tanto con el exterior como con otros bloques por medio de señales. Se han realizado las conexiones correspondientes al diagrama de bloques, y el código resultante es el siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
--library UNISIM;
--use UNISIM.VComponents.all;

entity top is
    Port (
        CLK                 : in std_logic;                     -- Señal de reloj
        reset               : in std_logic;                   -- Entrada reset
activa a nivel alto
        boton_central       : in std_logic;                     -- Utilaza para
seleccion del producto y modo de pago con tarjeta
        moneda_10c          : in std_logic;                     -- Moneda 10
centimos
        moneda_20c          : in std_logic;                     -- Moneda 20
centimos
        moneda_50c          : in std_logic;                     -- Moneda 50
centimos
        moneda_1e           : in std_logic;                     -- Moneda 1 euro
        producto_SW         : in std_logic_vector(7 downto 0);  -- Switches para
seleccion numero de producto en BCD
        seven_segment       : out std_logic_vector(6 downto 0); -- Salida a los
displays
        punto               : out std_logic;                    -- Salida para
representar el punto decimal en el dinero
        select_SW1          : out std_logic;                    -- Salida para
representar las unidades del producto
        select_SW2          : out std_logic;                    -- Salida para
representar las decenas del producto
        select_decenas      : out std_logic;                    -- Salida para
representar las decenas (centimos) del dinero
        select_centenas     : out std_logic;                    -- Salida para
representar las centenas (euros) del dinero
        select_unidades     : out std_logic;                    -- Salida para
representar las unidades (centimos) del dinero
        AN                  : out std_logic_vector(2 downto 0); -- Anodos que
queremos apagados
        led_pro_entregado   : out std_logic;                    -- true producto
entregado, 0 producto no entregado
        led_pro_ok          : out std_logic;                    -- true producto
elegido correctamente
        led_trabajando      : out std_logic;                    -- true si la
maquina está procesando, o bien esperando al pago, devolviendo o entregando producto
        led_dinero_dev      : out std_logic;                     -- true si se ha
devuelto correctamente el pago
        led_standby         : out std_logic                     -- true si se esta
en el estado de standby
     );
end top;

architecture Behavioral of top is

--------------- COMPONENTES --------------
COMPONENT antirebotes is
    port(
    ------------------------------ ENTRADAS ---------------------------------------
        CLK                     : in std_logic;                     -- Señal de
reloj
```

```
        reset_in            : in std_logic;                     -- Entrada reset
activa a nivel bajo
        boton_central_in      : in std_logic;                     -- Utilaza para
seleccion del producto y modo de pago con tarjeta
        moneda_10c_in         : in std_logic;                     -- Moneda 10
centimos
        moneda_20c_in         : in std_logic;                     -- Moneda 20
centimos
        moneda_50c_in         : in std_logic;                     -- Moneda 50
centimos
        moneda_1e_in          : in std_logic;                     -- Moneda 1 euro
        --------------------------- SALIDAS ----------------------------------------
-
        reset_out             : out std_logic;
        boton_central_out      : out std_logic;
        moneda_10c_out         : out std_logic;
        moneda_20c_out         : out std_logic;
        moneda_50c_out         : out std_logic;
        moneda_1e_out          : out std_logic

    );
end COMPONENT;

COMPONENT maquina is
    Port (
    producto_ok : in std_logic;
    dinero_ok : in std_logic;
    clk : in std_logic;
    clk_2Hz : in std_logic;
    reset : in std_logic;
    dinero_devuelto : out std_logic;
    led_standby : out std_logic;
    led_prod_entregado : out std_logic;
    led_trabajando : out std_logic;
    led_prod_ok : out std_logic;
    led_dinero_dev : out std_logic
    );
end COMPONENT;

COMPONENT IntroducirProducto is
    Port (
    clk : in std_logic;                                 -- Señal de reloj
    boton_central : in std_logic;                       -- Boton central, señal que
llega desde el bloque antirebotes
    SW : in std_logic_vector(7 downto 0);               -- Switches
    producto_ok : out std_logic                        -- 1 si el producto es correcto,
0 si no es codigo BCD correcto
    );
end COMPONENT;

COMPONENT temporizador is
  generic(
  MODULO : positive
  );
  Port (
  clk       : in std_logic;  --Reloj
  reset     : in std_logic;  --Entrada reset asíncrona
  contado   : out std_logic  --Salida del temporizador, 1 cuando se acaba la cuenta
  );
end COMPONENT;

component  contarDinero is
      port (
      CLK:        in std_logic;
      reset:      in std_logic;
      moneda10c:  in std_logic;
      moneda20c:  in std_logic;
      moneda50c:  in std_logic;
```

```
      moneda1e:   in std_logic;
         dinero_ok:  out std_logic;
      dinero_decenas:      out std_logic_vector (3 downto 0);
      dinero_centenas:     out std_logic_vector (3 downto 0)
    );
end component ;

COMPONENT segmentDriver is
  Port (
  SW : in std_logic_vector(7 downto 0);
  dinero_decenas : in std_logic_vector(3 downto 0);
  dinero_centenas : in std_logic_vector(3 downto 0);
  clk : in std_logic;
  seven_segments : out std_logic_vector(6 downto 0);
  punto : out std_logic;
  select_SW1 : out std_logic;
  select_SW2 : out std_logic;
  select_decenas : out std_logic;
  select_centenas : out std_logic;
  select_unidades : out std_logic;
  AN              : out std_logic_vector(2 downto 0)
  );
end COMPONENT;

-------------- SEÑALES -------------------
    signal reset_i, boton_central_i, moneda_10c_i, moneda_20c_i, moneda_50c_i,
moneda_1e_i : std_logic;
    signal productoOK_i, dineroOK_i : std_logic;
    signal dinero_decenas_i,dinero_centenas_i : std_logic_vector(3 downto 0);
    signal dinero_devuelto_i : std_logic;
    signal clk_2Hz_i : std_logic;

begin
antirrebotes_c : antirebotes PORT MAP(
        CLK => CLK,
        reset_in => reset,
        boton_central_in => boton_central,
        moneda_10c_in => moneda_10c,
        moneda_20c_in => moneda_20c,
        moneda_50c_in => moneda_50c,
        moneda_1e_in => moneda_1e,
        --------------------------- SALIDAS ---------------------------------------
-
        reset_out => reset_i,
        boton_central_out => boton_central_i,
        moneda_10c_out => moneda_10c_i,
        moneda_20c_out => moneda_20c_i,
        moneda_50c_out => moneda_50c_i,
        moneda_1e_out  => moneda_1e_i
);

introducirProducto_c : IntroducirProducto PORT MAP(
    clk => CLK,
    boton_central => boton_central_i,
    SW => producto_SW,
    producto_ok => productoOK_i
);

monedero : contarDinero PORT MAP(
      CLK => CLK,
      reset => dinero_devuelto_i,
      moneda10c => moneda_10c_i,
      moneda20c => moneda_20c_i,
      moneda50c => moneda_50c_i,
      moneda1e => moneda_1e_i,
      dinero_ok => dineroOK_i,
      dinero_decenas => dinero_decenas_i,
      dinero_centenas => dinero_centenas_i
```

```
);

segmentDriver_c : segmentDriver Port map (
  SW => producto_SW,
  dinero_decenas => dinero_decenas_i,
  dinero_centenas => dinero_centenas_i,
  clk => clk,
  seven_segments => seven_segment,
  punto => punto,
  select_SW1 => select_SW1,
  select_SW2 => select_SW2,
  select_decenas => select_decenas,
  select_centenas => select_centenas,
  select_unidades  => select_unidades,
  AN => AN
  );

maquina_c : maquina PORT MAP(
    producto_ok => productoOK_i,
    dinero_ok => dineroOK_i,
    clk => CLK,
    clk_2Hz => clk_2Hz_i,
    reset => reset_i,
    dinero_devuelto => dinero_devuelto_i,
    led_standby => led_standby,
    led_prod_entregado => led_pro_entregado,
    led_trabajando => led_trabajando,
    led_prod_ok => led_pro_ok,
    led_dinero_dev => led_dinero_dev
);

clk2HZ : temporizador GENERIC MAP (
    MODULO => 50000000
    )
    PORT MAP(
    clk => CLK,
    reset => reset_i,
    contado => clk_2Hz_i
    );


end Behavioral;
```

TESTBENCHES

*Los códigos de los testbench se incluyen en un anexo final ya que ocupan mucho espacio y así se facilita la lectura de la memoria.

1. Bloque Introducir Producto

En este testbench comprobamos que los códigos BCD introducidos son o no correctos. Para ello, si el botón está pulsado y se produce un flanco de reloj (tanto ascendente como descendente) y si el número del producto está en el rango 00-39 la salida *producto_ok* se pone a '1'. En el *test* comprobamos tanto códigos correctos como incorrectos en ambos dígitos.

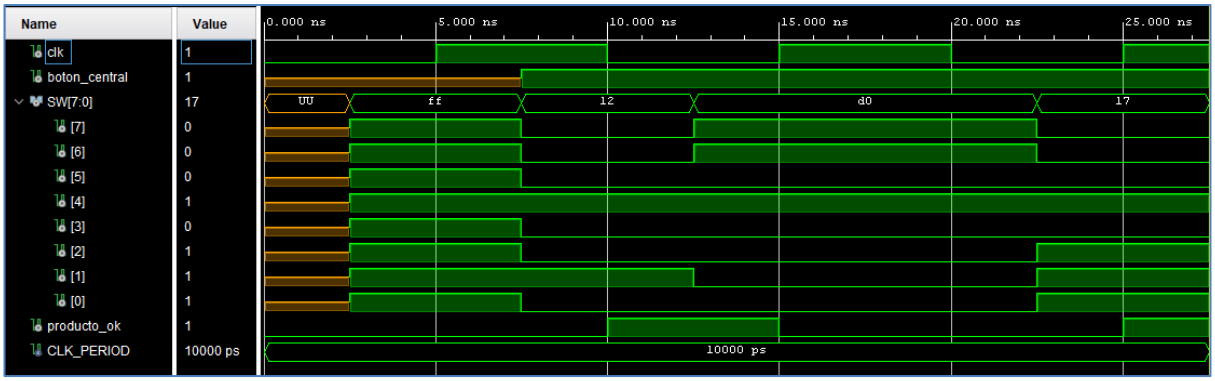

Ilustración 8. Testbench introducirProducto

2. Bloque Máquina

Para probar el bloque máquina, básicamente probamos su diagrama de estados, evolucionando de un estado al siguiente. En el bloque máquina, solo excitamos los relojes y las entradas de *producto_ok* y *dinero_ok*, ya que internamente se gestionan los temporizadores. Las formas de onda son las siguientes:

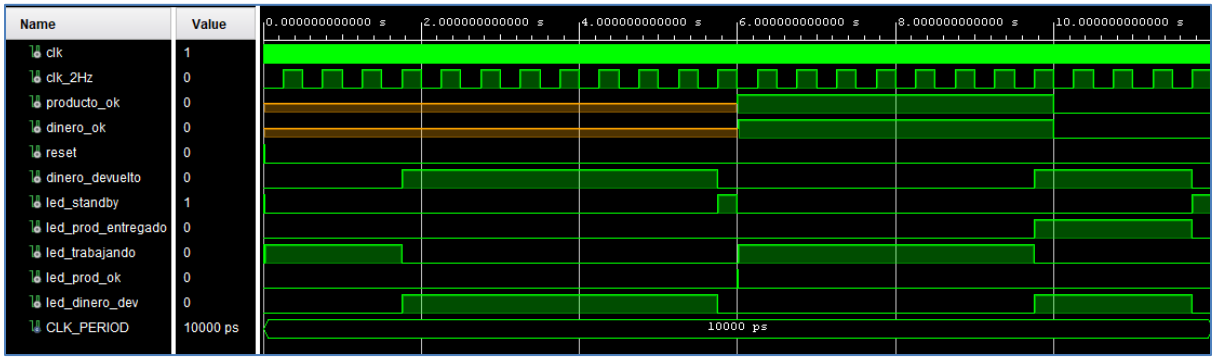

Ilustración 9. Testbench Máquina

Como podemos ver, lo primero que hacemos es darle al botón de *reset*, lo que hace que la máquina trabaje 2 segundos para devolver el dinero, y después devuelve el dinero y se queda 4 segundos hasta que vuelve al *standby*. Una vez en *standby*, le mandamos la señal de *producto_ok* y se pasa al estado de *producto_ok*, mandamos la señal de *dinero_ok* y la maquina se pone a trabajar durante 4 segundos, y después entrega el producto y el dinero sobrante, y tras 2 segundos vuelve al standby.

3. Bloque FSM

En este *testbench* probamos exactamente lo mismo que en el del bloque máquina, con la diferencia de que tenemos nosotros que variar los temporizadores manualmente. Las formas de onda son las siguientes:

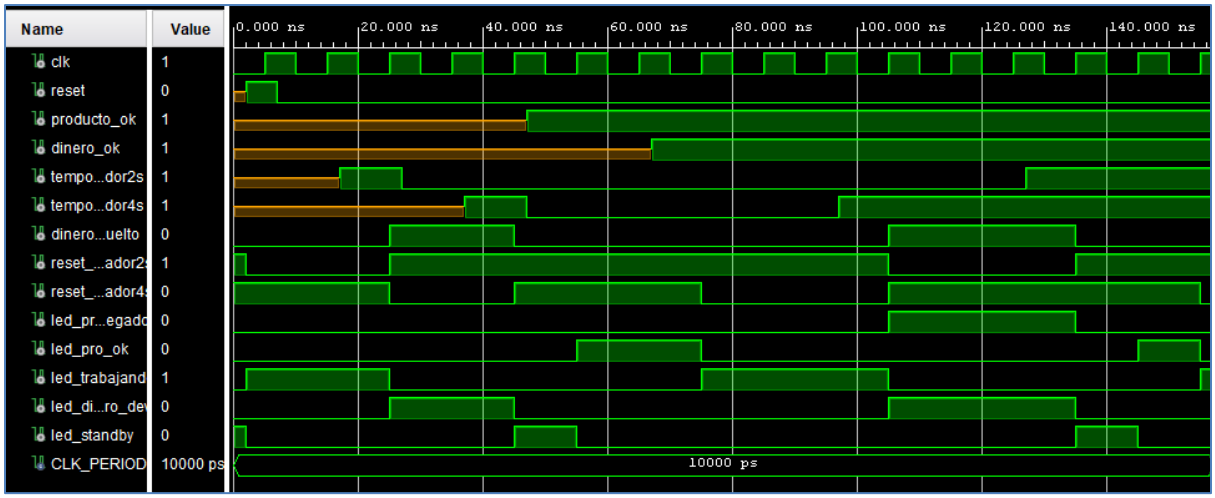

Ilustración 10. Testbench FSM

Cabe destacar que, en este caso, la escala de tiempos no es relevante, pues solo probamos la transición entre estados y sus salidas.

4. Bloque temporizador

En este testbench probamos el funcionamiento general del bloque. Cómo podemos adaptar el número de flancos de subida de reloj que queremos que cuente gracias al uso de genéricos, establecemos MODULE como 10, por lo que tras nueve flancos de reloj deberá la salida ponerse a 1. La forma de onda es la siguiente:

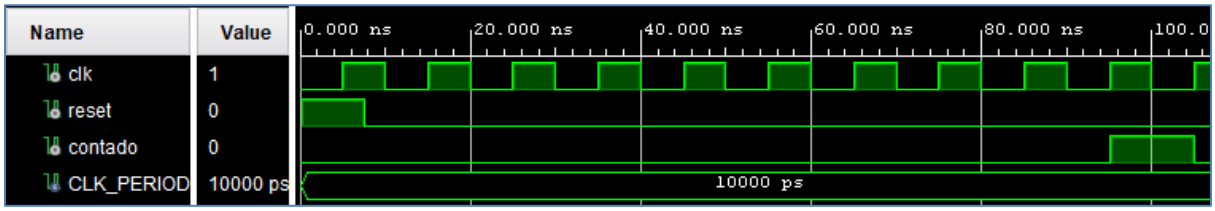

Ilustración 11. Testbench Temporizador

Como podemos observar, si la entrada no es nula, la salida se pone a 1 tras nueve flancos ascendentes de reloj.

5. Bloque Contar Dinero

Para probar el correcto funcionamiento de nuestro bloque *contarDinero* introduciremos varias monedas e iremos viendo como el contador almacena correctamente los valores introducidos y los distribuye en las señales referentes a las decenas de céntimos y centenas de céntimos (euros), a su vez también cambiará el valor de la señal *dinero_ok* a 1 cuando el contador supere el valor 10, correspondiente a un euro.

Inicialmente se introducirá una moneda de 10 céntimos, luego una de 20, una de 50 y finalmente un de 1 euro que al no ser desactivada contabilizará en cada flanco de reloj, de forma equivalente a haber introducido 3. Se puede observar como cuando el contador está en 8 (80 céntimos) y se introduce una moneda de 1 euro, se activa el valor *dinero_ok*.

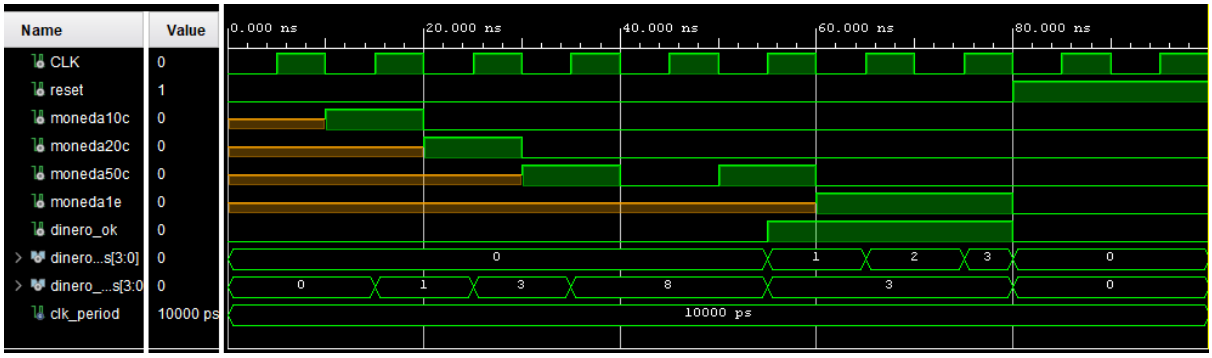

Ilustración 12. Testbench contarDinero

6. Bloque Top

Finalmente, en este testbench, comprobamos que la unión de todas las entidades funciona correctamente, siguiendo la secuencia de operación de la máquina expendedora.

ANEXO CÓDIGO TESTBENCHs

Testbench introducirProducto

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

entity IntroducirProducto_tb is
end IntroducirProducto_tb;

architecture Behavioral of IntroducirProducto_tb is
    COMPONENT IntroducirProducto is
    Port (
    clk : in std_logic;
    boton_central : in std_logic;
    SW : in std_logic_vector(7 downto 0);
    producto_ok : out std_logic
    );
    end COMPONENT;
    signal clk : std_logic := '0';
    signal boton_central : std_logic;
    signal SW : std_logic_vector(7 downto 0);
    signal producto_ok : std_logic;

    constant CLK_PERIOD : time := 10 ns;
begin

genclk : clk <= not clk after 0.5 * CLK_PERIOD;

uut : IntroducirProducto PORT MAP(
    clk => clk,
    boton_central => boton_central,
    SW => SW,
    producto_ok => producto_ok
```

```
);

estimulos : process
begin
    wait for 0.25*CLK_PERIOD;
    SW <= "11111111";
    wait for 0.5*CLK_PERIOD;
    assert producto_ok = '0'
        report "[ERROR] Se ha activado la salida sin activar el boton"
        severity failure;

    SW <= "00010010";
    boton_central <= '1';
    wait for 0.5*CLK_PERIOD;
    boton_central <= '0';
    assert producto_ok = '1'
        report "[ERROR] Se ha introducido un codigo correcto y la salida no se ha activado"
        severity failure;

    SW <= "11010000";
    boton_central <= '1';
    wait for 1*CLK_PERIOD;
    boton_central <= '0';
    assert producto_ok = '0'
        report "[ERROR] Se ha introducido un codigo incorrecto y la salida se ha activado"
        severity failure;

    SW <= "00010111";
    boton_central <= '1';
    wait for 0.5*CLK_PERIOD;
    boton_central <= '0';
    assert producto_ok = '1'
        report "[ERROR] Se ha introducido un codigo correcto y la salida no se ha activado"
        severity failure;

    assert false
        report "[EXITO] Se ha completado la simulacion correctamente"
        severity failure;


end process;

end Behavioral;
```

Testbench Maquina

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

entity maquina_tb is
end maquina_tb;


architecture Behavioral of maquina_tb is

COMPONENT maquina is
    Port (
    producto_ok : in std_logic;
    dinero_ok : in std_logic;
    clk : in std_logic;
    clk_2Hz : in std_logic;
    reset : in std_logic;
    dinero_devuelto : out std_logic;
    led_standby : out std_logic;
    led_prod_entregado : out std_logic;
    led_trabajando : out std_logic;
    led_prod_ok : out std_logic;
    led_dinero_dev : out std_logic
    );
end COMPONENT;

signal clk, clk_2Hz : std_logic := '0';
signal producto_ok, dinero_ok, reset, dinero_devuelto, led_standby, led_prod_entregado,
led_trabajando, led_prod_ok, led_dinero_dev : std_logic;
```

```
constant CLK_PERIOD : time := 10 ns;
begin

uut : maquina port map (
    producto_ok => producto_ok, dinero_ok => dinero_ok, clk => clk, clk_2Hz => clk_2Hz,
reset => reset, dinero_devuelto => dinero_devuelto,
    led_standby => led_standby, led_prod_entregado => led_prod_entregado, led_trabajando =>
led_trabajando,
    led_prod_ok => led_prod_ok, led_dinero_dev => led_dinero_dev
);

clk_gen : clk <= not clk after 0.5 * CLK_PERIOD;
clk2HZ_gen : clk_2Hz <= not clk_2Hz after 250 ms;

stimulus : process
begin

    ------------------------------------------------ RESETEO -------------------------------
--------------------
    wait for 0.2 * CLK_PERIOD;
    assert led_standby = '1'
        report "[FALLO] Error en el inicio de la FSM"
        severity failure;

   reset <= '1';
   wait for 0.5*CLK_PERIOD;
   reset <= '0';
   assert led_trabajando = '1'
        report "[FALLO] Error en el reset"
        severity failure;
   wait for CLK_PERIOD;


   wait for 2000 ms;
   wait for 2* CLK_PERIOD;
   assert led_dinero_dev = '1'
        report "[FALLO] Error devolviendo el dinero desde el reset"
        severity warning;

   wait for 4000 ms;
   wait for CLK_PERIOD;
   assert led_standby = '1'
     report "[FALLO] Error volviendo al standby"
     severity warning;
   ------------------------------------------------------------------------------------------
----------------------------------

   ----------------------------------------- SECUENCIA NORMAL ---------------------------
----------------------------------
   producto_ok <= '1';
   wait for 5 * CLK_PERIOD;
   assert led_prod_ok = '1'
        report "[FALLO] Error seleccionando el producto"
        severity warning;
    wait for CLK_PERIOD;

    dinero_ok <= '1';
    wait for CLK_PERIOD;
    assert led_trabajando = '1'
        report "[FALLO] Error pagando el producto"
        severity warning;
    wait for 2*CLK_PERIOD;

    wait for 4000 ms;
    producto_ok <= '0';
    dinero_ok <= '0';
    assert led_prod_entregado = '1'
        report "[FALLO] Error entregando el producto"
        severity warning;
     assert led_dinero_dev = '1'
        report "[FALLO] Error devolviendo el dinero"
        severity warning;
     wait for 2 * CLK_PERIOD;

     wait for 2000 ms;
     wait for CLK_PERIOD;
```

```
     assert led_standby = '1'
        report "[FALLO] Error volviendo al standby"
        severity warning;


   wait for 2 * CLK_PERIOD;
   assert false
        report "[EXITO] Simulacion completada correctamente"
        severity failure;

end process;

end Behavioral;
```

Testbench FSM

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


entity fsm_tb is
end fsm_tb;

architecture Behavioral of fsm_tb is
COMPONENT fsm is
    port (
        CLK                     : in std_logic;                   -- Señal de reloj
        reset                 : in std_logic;                  -- Entrada reset activa a
nivel alto
        producto_ok             : in std_logic;                   -- Entrada de que el
producto se ha seleccionado correctamente
        dinero_ok               : in std_logic;                   -- Entrada de que se ha
superado 1 euro
        temporizador2s          : in std_logic;                   -- Entrada del
temporizador
        temporizador4s          : in std_logic;                   -- Entrada del
temporizador
        dinero_devuelto         : out std_logic;                  -- Salida que reprenta
que se ha devuelto el producto
        enable_temporizador2s   : out std_logic;                  -- Salida para activar el
temporizador de 2 segundos
        enable_temporizador4s   : out std_logic;                  -- Salida para activar el
temporizador de 4 segundos
        led_pro_entregado       : out std_logic;                  -- true producto
entregado, 0 producto no entregado
        led_pro_ok              : out std_logic;                  -- true producto elegido
correctamente
        led_trabajando          : out std_logic;                  -- true si la maquina
está procesando, o bien esperando al pago, devolviendo o entregando producto
        led_dinero_dev          : out std_logic;                  -- true si se ha devuelto
correctamente el pago
        led_standby             : out std_logic                   -- true si se esta en el
estado de standby
 );
end COMPONENT;

    signal clk : std_logic := '0';
    signal reset, producto_ok, dinero_ok, temporizador2s, temporizador4s, dinero_devuelto,
enable_temporizador2s : std_logic;
    signal enable_temporizador4s, led_pro_entregado, led_pro_ok, led_trabajando,
led_dinero_dev, led_standby : std_logic;
    --signal dinero : unsigned;
    constant CLK_PERIOD : time := 10 ns;
begin

uut: fsm port map (
    CLK => clk, reset => reset, producto_ok => producto_ok, dinero_ok => dinero_ok,
temporizador2s => temporizador2s,
    temporizador4s => temporizador4s, dinero_devuelto => dinero_devuelto,
enable_temporizador2s => enable_temporizador2s,
    enable_temporizador4s => enable_temporizador4s, led_pro_entregado => led_pro_entregado,
led_pro_ok => led_pro_ok, led_trabajando => led_trabajando,
```

```
    led_dinero_dev => led_dinero_dev, led_standby => led_standby
);

genclk : clk <= not clk after 0.5 * CLK_PERIOD;

stimulus: process
begin

    ------------------------------------------------ RESETEO -------------------------------
--------------------
    wait for 0.2 * CLK_PERIOD;
    assert led_standby = '1'
        report "[FALLO] Error en el inicio de la FSM"
        severity failure;

   reset <= '1';
   wait for 0.5*CLK_PERIOD;
   reset <= '0';
   assert led_trabajando = '1'
        report "[FALLO] Error en el reset"
        severity failure;
   wait for CLK_PERIOD;

   temporizador2s <= '1';
   wait for CLK_PERIOD;
   temporizador2s <= '0';
   assert led_dinero_dev = '1'
        report "[FALLO] Error devolviendo el dinero desde el reset"
        severity failure;

   wait for CLK_PERIOD;
   temporizador4s <= '1';
   wait for CLK_PERIOD;
   assert led_standby = '1'
     report "[FALLO] Error volviendo al standby"
     severity failure;
   temporizador4s <= '0';
   ------------------------------------------------------------------------------------------
-----------------------------------

   ------------------------------------------- SECUENCIA NORMAL ---------------------------
-----------------------------------
   producto_ok <= '1';
   wait for CLK_PERIOD;
   assert led_pro_ok = '1'
        report "[FALLO] Error seleccionando el producto"
        severity failure;
    wait for CLK_PERIOD;

    dinero_ok <= '1';
    wait for CLK_PERIOD;
    assert led_trabajando = '1'
        report "[FALLO] Error pagando el producto"
        severity failure;
    wait for 2*CLK_PERIOD;

    temporizador4s <= '1';
    wait for CLK_PERIOD;
    assert led_pro_entregado = '1'
        report "[FALLO] Error entregando el producto"
        severity failure;
     assert led_dinero_dev = '1'
        report "[FALLO] Error devolviendo el dinero"
        severity failure;
     wait for 2 * CLK_PERIOD;

     temporizador2s <= '1';
     wait for CLK_PERIOD;
     assert led_standby = '1'
        report "[FALLO] Error volviendo al standby"
        severity failure;


   wait for 2 * CLK_PERIOD;
   assert false
        report "[EXITO] Simulacion completada correctamente"
```

```
        severity failure;

end process;

end Behavioral;
```

Testbench Temporizador

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

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

entity temporizador_tb is
end temporizador_tb;

architecture Behavioral of temporizador_tb is
    COMPONENT temporizador is
      generic(
      MODULO : positive
      );
      Port (
      clk       : in std_logic;  --Reloj
      reset     : in std_logic;  --Entrada reset asíncrona
      contado   : out std_logic  --Salida del temporizador, 1 cuando se acaba la cuenta
      );
    end COMPONENT;
    signal clk : std_logic := '0';
    signal reset,contado : std_logic;
    constant CLK_PERIOD : time := 10 ns;
begin

uut : temporizador
generic map(
    MODULO => 10
)
port map (
    clk => clk,
    reset => reset,
    contado => contado
);

genclk : clk <= not clk after 0.5 * CLK_PERIOD;

stimulus : process
begin

    reset <= '1';
    wait for 0.75 * CLK_PERIOD;
    assert contado = '0'
        report "[FALLO] Reset no funciona correctamente"
        severity failure;

    reset <= '0';
    wait for 9 * CLK_PERIOD;
    assert contado = '1'
        report "[FALLO] El temporizador alcanza la cuenta pero la salida no se activa"
        severity failure;

    wait for 1 * CLK_PERIOD;
    assert contado = '0'
        report "[FALLO] El temporizador no se pone a 0 cuando se acaba la cuenta"
        severity failure;

    assert false
        report "[EXITO] Se ha completado la simulacion correctamente"
```

```
        severity failure;

end process;
```

Testbench contarDinero

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

entity contarDinero_tb is
--  Port ( );
end contarDinero_tb;

architecture Behavioral of contarDinero_tb is
 component  contarDinero is
        port (
      CLK:                in std_logic;
      reset:              in std_logic;
      moneda10c:          in std_logic;
      moneda20c:          in std_logic;
      moneda50c:          in std_logic;
      moneda1e:           in std_logic;
      dinero_ok:          out std_logic;
      dinero_decenas:     out std_logic_vector (3 downto 0);
      dinero_centenas:    out std_logic_vector (3 downto 0)
    );

end component ;
signal CLK : std_logic :='0';
signal reset : std_logic :='0';
signal moneda10c,moneda20c,moneda50c,moneda1e,dinero_ok: std_logic;
signal dinero_centenas,dinero_decenas:  std_logic_vector (3 downto 0);

constant clk_period:time := 10ns;
begin
CLK <= not CLK after 0.5*clk_period ;
uut: contarDinero port map (
CLK => CLK ,
reset => reset ,
moneda10c => moneda10c,
moneda20c => moneda20c ,
moneda50c => moneda50c,
moneda1e => moneda1e,
dinero_ok => dinero_ok,
dinero_centenas => dinero_centenas,
dinero_decenas => dinero_decenas
);

stim_proc: process
begin


wait for 10 ns;
moneda10c <= '1';
wait for 10 ns;
moneda10c <= '0';
moneda20c <= '1';
wait for 10 ns;
moneda20c <= '0';
moneda50c <= '1';
wait for 10 ns;
moneda50c <= '0';
wait for 10 ns;
moneda50c <= '1';
wait for 10 ns;
moneda50c <= '0';
moneda1e <= '1';
wait for 20 ns;
moneda1e <= '0';
reset <= '1';
wait for 20 ns;
```

```
assert FALSE
    report "La simulación se finalizó correctamente"
    severity FAILURE ;
end process;
end Behavioral;
```

Testbench Top

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


entity top_tb is
end top_tb;

architecture Behavioral of top_tb is
COMPONENT top is
    Port (
        CLK                 : in std_logic;                     -- Señal de reloj
        reset             : in std_logic;                   -- Entrada reset activa a nivel
alto
        boton_central       : in std_logic;                     -- Utilaza para seleccion
del producto y modo de pago con tarjeta
        moneda_10c          : in std_logic;                     -- Moneda 10 centimos
        moneda_20c          : in std_logic;                     -- Moneda 20 centimos
        moneda_50c          : in std_logic;                     -- Moneda 50 centimos
        moneda_1e           : in std_logic;                     -- Moneda 1 euro
        producto_SW         : in std_logic_vector(7 downto 0);  -- Switches para seleccion
numero de producto en BCD
        seven_segment       : out std_logic_vector(6 downto 0); -- Salida a los displays
        punto               : out std_logic;                    -- Salida para representar
el punto decimal en el dinero
        select_SW1          : out std_logic;                    -- Salida para representar
las unidades del producto
        select_SW2          : out std_logic;                    -- Salida para representar
las decenas del producto
        select_decenas      : out std_logic;                    -- Salida para representar
las decenas (centimos) del dinero
        select_centenas     : out std_logic;                    -- Salida para representar
las centenas (euros) del dinero
        select_unidades     : out std_logic;                    -- Salida para representar
las unidades (centimos) del dinero
        led_pro_entregado   : out std_logic;                    -- true producto entregado,
0 producto no entregado
        led_pro_ok          : out std_logic;                    -- true producto elegido
correctamente
        led_trabajando      : out std_logic;                    -- true si la maquina está
procesando, o bien esperando al pago, devolviendo o entregando producto
        led_dinero_dev      : out std_logic;                     -- true si se ha devuelto
correctamente el pago
        led_standby         : out std_logic                   -- true si se esta en el
estado de standby
     );
end COMPONENT;
    signal clk : std_logic := '0';
    signal reset, boton_central, moneda_10c, moneda_20c, moneda_50c, moneda_1e: std_logic;
    signal led_pro_entregado, led_pro_ok, led_trabajando, led_dinero_dev, led_standby :
std_logic;
    signal punto, select_SW1, select_SW2, select_unidades, select_decenas, select_centenas
: std_logic;
    signal producto_SW : std_logic_vector(7 downto 0);
    signal seven_segment: std_logic_vector(6 downto 0);

    constant CLK_PERIOD : time := 10 ns;
begin

genclk : clk <= not clk after 0.5 * CLK_PERIOD;

uut: top port map (
        CLK                 => clk,
```

```
        reset             => reset,
        boton_central       => boton_central,
        moneda_10c          => moneda_10c,
        moneda_20c          => moneda_20c,
        moneda_50c          => moneda_50c,
        moneda_1e           => moneda_1e,
        producto_SW         => producto_SW,
        seven_segment       => seven_segment,
        punto               => punto,
        select_SW1          => select_SW1,
        select_SW2          => select_SW2,
        select_decenas      => select_decenas,
        select_centenas     => select_centenas,
        select_unidades     => select_unidades,
        led_pro_entregado   => led_pro_entregado,
        led_pro_ok          => led_pro_ok,
        led_trabajando      => led_trabajando,
        led_dinero_dev      => led_dinero_dev,
        led_standby         => led_standby
);

estimulos : process
begin


    reset <= '1' after 0.25 * CLK_PERIOD;
    wait for 1 * CLK_PERIOD;
    assert led_trabajando = '1'
        report "[ERROR] Reset no funciona correctamente"
        severity warning;
    reset <= '0';

    wait for 2000 ms; --Espera al temporizador que representa a la maquina devolviendo el
dinero
    wait for 1 * CLK_PERIOD;
    assert led_dinero_dev = '1'
        report "[ERROR]: Error al devolver el dinero"
        severity warning;

    wait for 4000 ms; --Espera al temporizador que representa a la maquina devolviendo el
dinero
    wait for 1 * CLK_PERIOD;
    assert led_standby = '1'
        report "[ERROR]: Error al volver al standby desde devolver dinero"
        severity warning;

       wait for 1 * CLK_PERIOD;
       producto_SW <= ("11111111");
       boton_central  <= '1';
       wait for 0.01*CLK_PERIOD;
       boton_central<= '0';
       wait for 100 ms;
       assert led_standby = '1'
        report "[ERROR] No funciona correctamente la seleccion del producto, codigo
incorrecto ha sido detectado como correcto"
        severity warning;

        wait for 1*CLK_PERIOD;
        producto_SW <= ("00010000");
        wait for 0.01*CLK_PERIOD;
        boton_central<= '1';
        wait for 100 ms;
        boton_central<= '0';
        assert led_pro_ok = '1'
        report "[ERROR] No funciona correctamente la seleccion del producto, codigo
incorrecto ha sido detectado como correcto"
        severity warning;

        moneda_50c <= '1';
        wait for 50 ms;
        moneda_50c <='0';
        assert led_pro_ok = '1'
        report "[ERROR] Se introdujeron 50c y se aceptó el pago incorrectmanete"
        severity warning;
        moneda_1e <= '1';
```

```
        wait for 50 ms;
        assert led_trabajando = '1'
            report "[ERROR] No trabaja la maquina al introducir el dinero"
            severity warning;
        wait for 4000 ms;
        assert led_pro_entregado = '1'
            report "[ERROR] No se ha encendido el led de producto entregado"
            severity warning;
        assert led_dinero_dev = '1'
            report "[ERROR] No se ha encendido el led de dinero devuelto"
            severity warning;

        wait for 2000 ms;
        assert led_standby = '1'
            report "[ERROR] No se ha vuelto al standby"
            severity warning;

      assert false
        report "[EXITO]: Simulacion Completada"
        severity failure;

end process;

end Behavioral;
```

BIBLIOGRAFÍA

- Material de clase
- Material del laboratorio de la asignatura
- Youtube.com
- Ics.uci.edu
- Fpgatutorial.com
- Controllerstech.com
- Nandland.com
- Electronics.stackexchange.com
- Datasheet Nexys DDR4
- Allaboutfpga.com