escuela técnica superior de ingeniería y diseño industrial

Trabajo VHDL

Máquina Expendora

CÉSAR ZABALA MARTÍN - 55540
MARÍA SANZ PIÑA - 55463
SAMUEL SÁNCHEZ CRUZ - 55450

Índice

Introducción

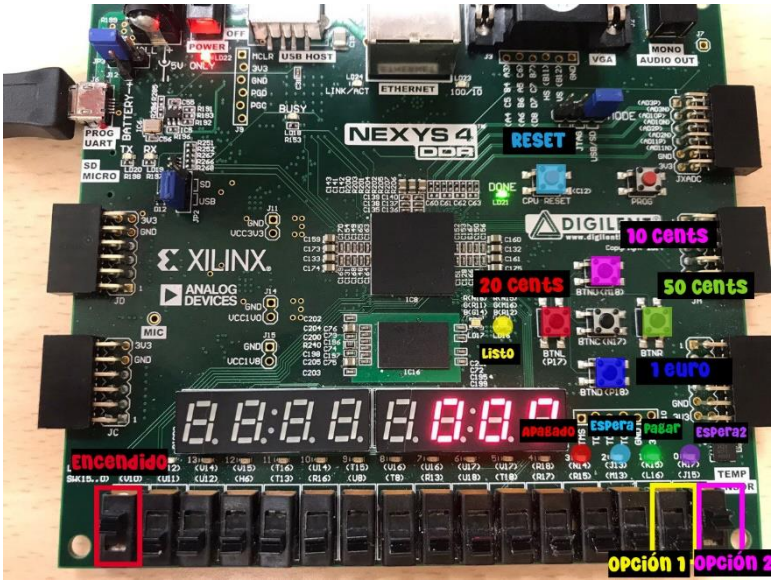
El proyecto que hemos elegido para programar en VHDL es una máquina expendedora.

Cuenta con las siguientes funciones:

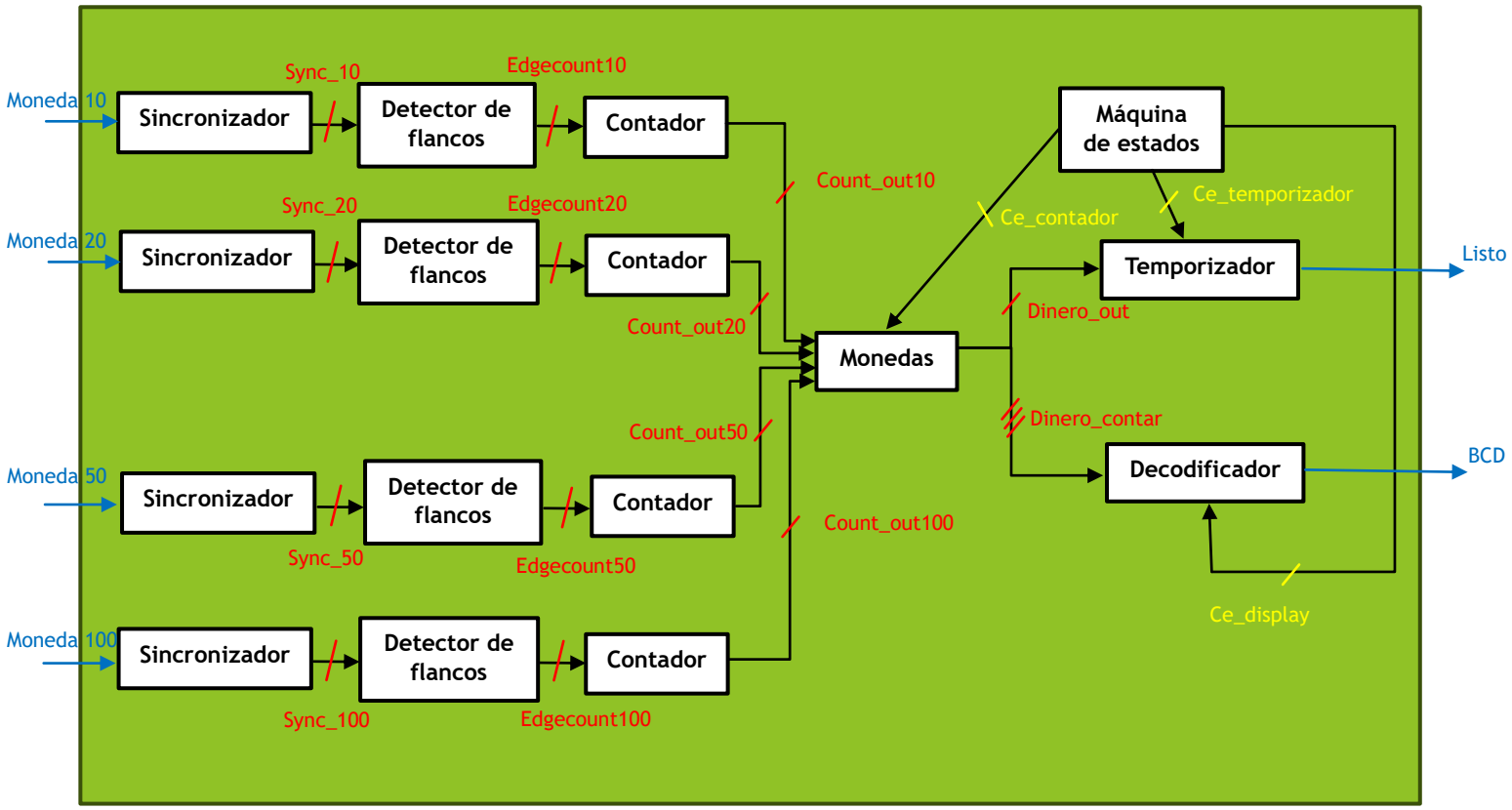
- Botón de apagado y encendido de la máquina
- Selección de opciones distintas de refrescos
- Contador de monedas y devolución de cambio
- Temporizador que actúa según la opción elegida
- Luces LED que avisa al humano que tiene que hacer en cada caso
- Detección de errores que nos avisa en cada caso

Todo esto se podría representar mediante la siguiente imagen de manera simbólica:

En nuestra FPGA:



Nos hemos basado en una estructura simple que tenga una entidad máquina de estados coordinando las demás entidades. La estructura del proyecto es la siguiente:



Entidades

Entidad TOP

Vamos a crear una entidad global que englobe las demás entidades para seguir una estructura. En la entidad top se definen las entradas y salidas de cada entidad, mientras que el funcionamiento de las entidades se programará en la propia entidad.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

entity TOP1 is
GENERIC(
MONEDAS: positive:=4; --Número de bits necesarios para representar el valor, en nuestro caso será 4
  --porque el mayor número que necesitamos representar es 9
DISPLAY: positive:=8; --Displays de 7 segmentos disponibles
SEGMENTOS: positive:=7; --segmentos del BCD
ESTADOS: positive:=4;--número de estados totales representados en los leds
OPCIONES: positive:=2 --Opciones de productos disponibles
);
PORT (
 Option : IN std_logic_vector(0 to OPCIONES-1); --Entrada de seleccion de las opciones
 Encendido : in std_logic; --Interruptor de encendido
 Monedas10: in std_logic; --Entrada de monedas de 10cents
 Monedas20: in std_logic; --Entrada de monedas de 20cents
 Monedas50: in std_logic; --Entrada de monedas de 50cents
 Monedas100: in std_logic; --Entrada de monedas de 1euro
 clk, RESET: in std_logic; --Señal de reset y de reloj
 LIGHT : out std_logic_vector(ESTADOS-1 downto 0); --Leds que marcan el estado en el que se encuentra
 LED_AZUL: out std_logic; --Led indicativo de que ha terminado el temporizador y se puede recoger el producto en el estado final
 BCD: out std_logic_vector(SEGMENTOS-1 downto 0); --Segmentos del BCD
 AN_control: out std_logic_vector(DISPLAY-1 downto 0)--Selección de los distintos displays
 );
end TOP1;
```

En esta primera parte de la entidad definimos las entradas del sistema que hemos esquematizado en la introducción y las salidas que tendremos.

Como podemos ver, tenemos unas luces de salida para controlar en qué estado nos encontramos en todo momento. Estas luces en el esquema inicial no salen, pues es funcionamiento interno para ver que no se está produciendo ningún problema.

Entidades

A continuación, se muestran todas las entidades con sus entradas y salidas así como varias señales complementarias que se utilizarán en las asignaciones más adelante.

También, se añaden los genéricos necesarios para que el programa funcione de manera correcta.

```
--DECLARACIÓN DE LAS ENTIDADES NECESARIAS

--GENERADOR DE PULSOS
Component EDGEDTCTR
port(CLK : in std_logic;
 SYNC_IN : in std_logic;
 EDGE : out std_logic
 );
 end component;
 signal edgecount10: std_logic; --Salida del generador de pulsos correspondiente a las monedas de 10cents
 signal edgecount20: std_logic; --Salida del generador de pulsos correspondiente a las monedas de 20cents
 signal edgecount50: std_logic; --Salida del generador de pulsos correspondiente a las monedas de 50cents
 signal edgecount100: std_logic; --Salida del generador de pulsos correspondiente a las monedas de 1euro

 --SINCRONIZADOR
 Component SYNCHRN2R
 port(CLK : in std_logic;
 ASYNC_IN : in std_logic;
 SYNC_OUT : out std_logic
 );
 end component;
 signal sync10: std_logic; --Salida del sincronizador correspondiente a las monedas de 10cents
 signal sync20: std_logic; --Salida del sincronizador correspondiente a las monedas de 20cents
 signal sync50: std_logic; --Salida del sincronizador correspondiente a las monedas de 50cents
 signal sync100: std_logic; --Salida del sincronizador correspondiente a las monedas de 1euro
```

```
 --CONTADOR
 Component counter
 generic (MONEDAS: positive);
 port(CLK, reset, CE: in std_logic;
    signal count: out std_logic_vector(3 downto 0)
    );
 end component;
signal count_out10: std_logic_vector(0 to MONEDAS-1); --Salida del número de monedas de 10cents
signal count_out20: std_logic_vector(0 to MONEDAS-1); --Salida del número de monedas de 10cents
signal count_out50: std_logic_vector(0 to MONEDAS-1); --Salida del número de monedas de 10cents
signal count_out100: std_logic_vector(0 to MONEDAS-1); --Salida del número de monedas de 10cents

--DECODIFICADOR
 Component decoder
 generic(
  DISPLAY: positive; --Displays de 7 segmentos disponibles
  SEGMENTOS: positive; --segmentos del BCD
  MONEDAS: positive --Número de bits necesarios para representar el valor, en nuestro caso será 4
  --porque el mayor número que necesitamos representar es 9
);
 PORT (
decod_in1 : IN std_logic_vector(MONEDAS-1 DOWNTO 0);
decod_in2 : IN std_logic_vector(MONEDAS-1 DOWNTO 0);
decod_in3 : IN std_logic_vector(MONEDAS-1 DOWNTO 0);
led : OUT std_logic_vector(SEGMENTOS-1 DOWNTO 0);
displays: out std_logic_vector(DISPLAY-1 downto 0);
CE: in std_logic;
Clk: in std_logic
);
 end component;
 signal ce_display: std_logic; --Entrada que activa la entidad


 --TEMPORIZADOR
 Component temporizador
 generic(
OPCIONES: positive --Opciones de productos disponibles
);
 port(
    Clk     : in std_logic;
    CE : in std_logic;
    Reset    : in std_logic;
    Opcion: in std_logic_vector(0 to OPCIONES-1);
    Temp_out: out std_logic
    );


 --MÁQUINA DE ESTADOS
 component fsm
 generic(
  ESTADOS: positive;--número de estados totales representados en los leds
  OPCIONES: positive --Opciones de productos disponibles
  );
 port (
  RESET : in std_logic;
  ENCENDIDO : in std_logic;
  CLK : in std_logic;
  fsm_in : in std_logic_vector(0 to OPCIONES-1);
  Led : out std_logic_vector(0 TO ESTADOS-1);
  LED_AZUL: out std_logic;
  fsm_out_contador : out  std_logic;
  fsm_out_temporizador : out  std_logic;
  temp_out: in std_logic;
  dinero_out: in std_logic;
  fsm_out_display: out std_logic
  );
 end component;

signal ce_contador: std_logic; --Entrada que activa la entidad
signal dinero_out: std_logic; --Salida que indica que se ha alcanzado el precio del producto
signal Dinero_contar1: std_logic_vector(MONEDAS-1 downto 0); --Salida del valor de los euros
signal Dinero_contar2: std_logic_vector(MONEDAS-1 downto 0); --Salida del valor de las decenas de centimos
signal Dinero_contar3: std_logic_vector(MONEDAS-1 downto 0); --Salida del valor de las unidades de centimos (siempre 0)
```

```
 --ENTIDAD ENCARGADA DE TRADUCIR EL NÚMERO DE MONEDAS A SU VALOR
 Component Dinero
 generic(
 MONEDAS: positive --Número de bits necesarios para representar el valor, en nuestro caso será 4
  --porque el mayor número que necesitamos representar es 9
);
 Port (
    Dinero_in_10: in std_logic_vector(MONEDAS-1 downto 0);
    Dinero_in_20: in std_logic_vector(MONEDAS-1 downto 0);
    Dinero_in_50: in std_logic_vector(MONEDAS-1 downto 0);
    Dinero_in_100: in std_logic_vector(MONEDAS-1 downto 0);

    CE: in std_logic;
    CLK: in std_logic;
    Dinero_out: out std_logic;
    Dinero_euros: out std_logic_vector(MONEDAS-1 downto 0);
    Dinero_centimos1: out std_logic_vector(MONEDAS-1 downto 0);
    Dinero_centimos2: out std_logic_vector(MONEDAS-1 downto 0);
    RESET: in std_logic
);
 end component;
```

Asignamos las señales a las entradas y salidas internas de cada entidad:

```
begin

sincronizador1: SYNCHRNZR port map(
clk=>clk,
 ASYNC_IN=>monedas10,
 SYNC_OUT=>sync10
);
detectorflancos1: EDGEDTCTR port map(
CLK=>clk,
 SYNC_IN=>sync10,
 EDGE=>edgecount10
);

contador1: counter
generic map(MONEDAS)
port map(
CLK=>clk,
reset=>reset,
CE=>edgecount10,
count=>count_out10
);
sincronizador2: SYNCHRNZR port map(
clk=>clk,
 ASYNC_IN=>monedas20,
 SYNC_OUT=>sync20
);
CLK=>clk,
 SYNC_IN=>sync20,
 EDGE=>edgecount20
);
contador2: counter
generic map(MONEDAS)
port map(
CLK=>clk,
reset=>reset,
CE=>edgecount20,
count=>count_out20
);
sincronizador3: SYNCHRNZR port map(
clk=>clk,
 ASYNC_IN=>monedas50,
 SYNC_OUT=>sync50
);
detectorflancos3: EDGEDTCTR port map(
CLK=>clk,
 SYNC_IN=>sync50,
 EDGE=>edgecount50
);
contador3: counter
generic map(MONEDAS)
port map(
CLK=>clk,
reset=>reset,
CE=>edgecount50,
count=>count_out50
);
```

```
detectorflancos3: EDGEDTCTR port map(
CLK=>clk,
 SYNC_IN=>sync50,
 EDGE=>edgecount50
);
contador3: counter
generic map(MONEDAS)
port map(
CLK=>clk,
reset=>reset,
CE=>edgecount50,
count=>count_out50
);
sincronizador4: SYNCHRNZR port map(
clk=>clk,
 ASYNC_IN=>monedas100,
 SYNC_OUT=>sync100
);
detectorflancos4: EDGEDTCTR port map(
CLK=>clk,
 SYNC_IN=>sync100,
 EDGE=>edgecount100
);
contador4: counter
generic map(MONEDAS)
port map(
CLK=>clk,
reset=>reset,
CE=>edgecount100,
count=>count_out100
);
temporizador1: temporizador
generic map(OPCIONES)
port map(
Clk=>clk,
Reset=>reset,
Opcion=>option,
Temp_out=>temp_out1,
CE=>ce_temporizador
);
end estructural;

maquina_estados: fsm
generic map(ESTADOS,OPCIONES)
port map(
RESET=>reset,
ENCENDIDO=>encendido,
CLK=>clk,
fsm_in=>option,
Led=>light,
fsm_out_contador=>ce_contador,
fsm_out_temporizador=>ce_temporizador,
fsm_out_display=>ce_display,
temp_out=>temp_out1,
dinero_out=>dinero_out,
LED_AZUL=>LED_AZUL
);
```

```
decoder1: decoder
generic map(DISPLAY,SEGMENTOS,MONEDAS)
port map(
decod_in1=>dinero_contar1,
decod_in2=>dinero_contar2,
decod_in3=>dinero_contar3,
led=>bcd,
displays=>AN_control,
CE=>ce_display,
Clk=>clk
);
```

```
Dinero1: dinero
generic map(MONEDAS)
port map(
Dinero_in_10=>count_out10,
Dinero_in_20=>count_out20,
Dinero_in_50=>count_out50,
Dinero_in_100=>count_out100,

CE=>ce_contador,
Clk=>clk,
Dinero_out=>dinero_out,
Dinero_euros=>dinero_contar1,
Dinero_centimos1=>dinero_contar2,
Dinero_centimos2=>dinero_contar3,
reset=>reset
);
```

Una vez tenemos todas las asignaciones hechas, el código de la entidad top está finalizado. Los Testbench correspondientes se realizarán una vez definidas todas las entidades.

Sincronizador

Todas las entradas externas que tenemos en nuestra máquina expendedora deben estar sincronizadas para evitar que entren en un estado de metaestabilidad y aparezcan errores en el sistema. En nuestro caso, tendremos un sincronizador para cada botón correspondiente a las monedas.

Todas las entidades sincronizador tendrán la misma estructura; una entrada que será el reloj y la entrada asíncrona correspondiente al botón y una salida sincronizadora out, que servirá como entrada para la siguiente entidad.

```
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;
```

Detector de flancos

Al pulsar algún botón cuando tenemos entradas asíncronas, obtenemos un pulso que tiene una duración indeterminada hasta que se deja de pulsar el botón. Esto podría causar errores en el sistema, ya que, cada botón tendría una duración diferente desestabilizándolo. Por lo tanto, necesitamos el detector de flancos, cuya función es detectar cuando se ha producido un cambio de bit, o en otras palabras detectar un flanco de bajada para obtener el tiempo de duración del pulso en un ciclo de reloj.

Las entradas serán el reloj y la salida de su correspondiente sincronizador. La salida en nuestro caso será el pulso EDGE. Necesitaremos tantos detectores de flanco como sincronizadores tengamos, 4 en nuestro caso. La estructura de estos es la 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;
```

Contador

Una vez hemos implementado las entidades sincronizador y detector de flancos, necesitaremos un contador. Este contador será el encargado de contar el número de pulsaciones que se ha realizado en cada botón. Al ser un sistema síncrono, funcionará según la señal de reloj.

Tenemos la entrada de reloj, una entrada reset y un chip enable. La entrada reset permite reiniciar el contador y el chip enable permite encender o apagar el contador a nuestro gusto. Además, tenemos un vector de salida count, que será la señal que muestra el número de monedas introducidas.

Disponemos cuatro contadores, cada uno corresponde con un detector de flancos. La estructura es la siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;


entity counter is
    generic (MONEDAS: positive); --Número máximo que puede contar (4 bits serían hasta 15)
port(
    CLK, reset, CE: in std_logic; --Señales de reloj, reset y chip enable
    signal count: out std_logic_vector(3 downto 0) --Señal de salida que muestra el número de monedas introducidas
    );
end counter;

architecture Behavioral of counter is

begin
    process (CLK, reset)
    variable count2: unsigned (count'range):=(others=>'0');
    begin
        if reset='0' then
        count2:= "0000";
        else if rising_edge(CLK) then
            if CE='1' then
             count2:= count2+1;
            end if;
         end if;
         end if;
         count<=std_logic_vector(count2);
    end process;
end Behavioral;
```

Dinero

Esta entidad será la encargada de realizar las operaciones numéricas del sistema, a ella llegarán el número de monedas de cada tipo y se encargará de multiplicar el número de monedas introducidas por su valor e irlo sumando. Una vez alcanzado el valor de coste del producto (1,20€ en nuestro caso) automáticamente te devolverá el valor del cambio.

Es decir, si introducimos tres monedas de 50cents., esta entidad multiplicará tres por el valor de 50cents., obteniendo así 1.50€, y al ser este valor mayor que el valor de coste del producto, se hará automáticamente la resta y mostrará en pantalla 0,30 que es el cambio que saldrá de la máquina.

Tenemos una entrada de reloj, un reset y un chip enable que tienen las mismas funciones que anteriormente y además, tenemos una entrada para cada moneda que vendrá de la salida correspondiente del contador. También, tenemos la salida de euros, céntimos1 y céntimos2 que irán luego al display de 7 segmentos y tenemos la salida dinero_out que se activará cuando se alcance el valor de coste del producto y esta luego se empleará en la máquina de estados.

El código de la entidad es el siguiente:

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

USE ieee.std_logic_unsigned.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DINERO is
generic(
 MONEDAS: positive --Número de bits necesarios para representar el valor, en nuestro caso será 4
  --porque el mayor número que necesitamos representar es 9
);
Port (
    Dinero_in_10: in std_logic_vector(MONEDAS-1 downto 0);
    Dinero_in_20: in std_logic_vector(MONEDAS-1 downto 0);
    Dinero_in_50: in std_logic_vector(MONEDAS-1 downto 0);
    Dinero_in_100: in std_logic_vector(MONEDAS-1 downto 0);
    CE: in std_logic;
    CLK: in std_logic;
    Dinero_out: out std_logic;
    Dinero_euros: out std_logic_vector(MONEDAS-1 downto 0);
    Dinero_centimos1: out std_logic_vector(MONEDAS-1 downto 0);
    Dinero_centimos2: out std_logic_vector(MONEDAS-1 downto 0);
    RESET: in std_logic
);
end DINERO;

architecture Behavioral of DINERO is

begin

process (Dinero_in_10,clk,Dinero_in_20,Dinero_in_50,Dinero_in_100)
subtype dinero is integer range 0 to 10;
variable Cuenta_euros: dinero:=0;
variable Cuenta_cents1: dinero:=0;
variable Cuenta_cents2: std_logic_vector(0 to MONEDAS-1);
variable Dinero_out1: std_logic;

begin

    if reset='0'  then
    Cuenta_euros:= 0;
    Cuenta_cents1:= 0;
    Cuenta_cents2:= "0000";
    Dinero_out1:='0';
        else if rising_edge(CLK) then
            if CE='1' then
                    Cuenta_cents1:=to_integer(unsigned(Dinero_in_10))+2*to_integer(unsigned(Dinero_in_20))+5*to_integer(unsigned(Dinero_in_50));
                    Cuenta_euros:=to_integer(unsigned(Dinero_in_100));
```

```
            if Cuenta_cents1>9 then
                Cuenta_euros:=Cuenta_euros+1;
                Cuenta_cents1:=Cuenta_cents1-10;
            end if;
            if (Cuenta_euros=1 and Cuenta_cents1>=2) then
                Cuenta_euros:=Cuenta_euros-1;
                Cuenta_cents1:=Cuenta_cents1-2;
                Dinero_out1:='1';
            else if Cuenta_euros>=2 then
                Cuenta_euros:=Cuenta_euros-1;
                      case Cuenta_cents1 is
                           when 0=>
                               Cuenta_euros:=Cuenta_euros-1;
                               Cuenta_cents1:=8;
                               Dinero_out1:='1';
                           when 1=>
                               Cuenta_euros:=Cuenta_euros-1;
                               Cuenta_cents1:=9;
                               Dinero_out1:='1';
                           when others=>
                               Cuenta_cents1:=Cuenta_cents1-2;
                               Dinero_out1:='1';
                       end case;
                   end if;

            end if;
       end if;
   end if;
end if;

         Dinero_centimos2<="0000";
         Dinero_centimos1<=std_logic_vector(to_unsigned(Cuenta_cents1,Dinero_centimos1'length));
         Dinero_euros<=std_logic_vector(to_unsigned(Cuenta_euros,Dinero_euros'length));
         Dinero_out<=Dinero_out1;
    end process;

end Behavioral;
```

Máquina de estados

Esta entidad es una de las más importantes del proyecto, pues es la que se encargará de manejar los estados.

En nuestro caso hemos utilizado cinco estados: APAGADO, ESPERA, PAGAR, ESPERA2 y LISTO.

Apagado: la máquina se encontrará apagada por completo. Necesitará que pulsemos el switch de encendido para que abandone este estado. Si pulsamos cualquier otro switch y este no se ha activado, la máquina no funcionará en ningún caso.

Espera: tendremos ya la máquina encendida y se encontrará esperando a que alguien seleccione alguna de las opciones de la máquina para pasar al siguiente estado. Si no pulsamos nada (o pulsamos otra cosa que no sea la opción) y la máquina esta encendida, la máquina permanecerá en este estado. Además, cuando esté listo el producto y se pulse el reset, la máquina pasará a este estado.

Pagar: la persona ya habrá seleccionado la opción que le interesa y la máquina estará esperando con el BCD encendido a que se introduzca el dinero. Una vez se vaya introduciendo el dinero, este se irá sumando hasta alcanzar el valor de coste del producto.

Espera2: ya se habrá introducido el dinero necesario y devuelto el cambio. La máquina de estados activará el temporizador correspondiente al producto seleccionado.

Listo: el temporizador habrá acabado y la máquina habrá servido el producto. Además, en este estado se encenderá un LED que nos mostrará que nuestro producto está listo para recoger.

El código empleado en la máquina de estados es el siguiente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
use IEEE.numeric_std.all;
entity fsm is
  generic(
  ESTADOS: positive;--número de estados totales representados en los leds
  OPCIONES: positive --Opciones de productos disponibles
  );
  port (
  RESET : in std_logic;
  ENCENDIDO : in std_logic;
  CLK : in std_logic;
  fsm_in : in std_logic_vector(0 to OPCIONES-1);
  Led : out std_logic_vector(0 TO ESTADOS-1);
  LED_AZUL: out std_logic;
  fsm_out_contador : out  std_logic;
  fsm_out_temporizador : out  std_logic;
  temp_out: in std_logic;
  dinero_out: in std_logic;
  fsm_out_display: out std_logic
  );
end fsm;
```

En esta primera parte hemos definido los genéricos correspondientes y las entradas y salidas. Como anteriormente, tenemos las entradas reset y clk que corresponden al reset y al reloj de la máquina de estados. También tenemos la entrada asíncrona de encendido correspondiente al switch que enciende la máquina expendedora y tenemos el fsm_in que corresponde a las dos opciones que tenemos para elegir en la expendedora. Otras entradas corresponden a la salida del temporizador y a la salida de la cuenta de dinero.

Por último, las salidas son las correspondientes a los LEDs que marcan los estados, el LED de listo (que será de color azul) y el fsm_out del contador, el temporizador y el display que correspondería con sus chip enable.

Current_state

A continuación, se adjunta el código del proceso de la máquina de estados que actualiza el current state. Si pulsamos reset el estado actual será apagado y si hay un pulso de reloj y pulsamos encendido, se podrá actualizar el estado actual.

```
architecture behavioral of fsm is
  type STATES is (APAGADO,ESPERA, PAGAR, ESPERA2, LISTO);
  signal current_state: STATES:= APAGADO; --ESTADO ACTUAL
  signal next_state: STATES:= APAGADO; --ESTADO SIGUIENTE
   begin

  state_register: process (RESET, CLK)
  begin
        if reset='0' then
            current_state<=APAGADO;

        elsif rising_edge(clk) then
            if Encendido='1' then
            current_state<=next_state;
            else
            current_state<=apagado;
            end if;
        end if;
  end process;
```

Next_State

En esta parte de la máquina de estados tendremos el proceso que actualiza el next state. Dependiendo del estado actual se encenderán unas cosas u otras y en función de las entradas se pasará a un estado u a otro. Por ejemplo, en el estado espera tenemos el temporizador, el display y el contador apagado; si pulsamos una de las opciones pasaremos al estado de Pagar y si no pulsamos nada seguiremos en el estado de espera.

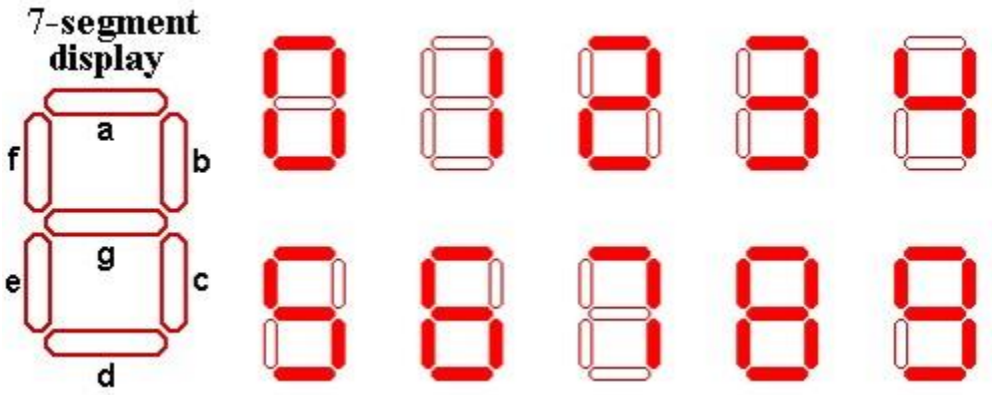
```
nextstate_decod: process (ENCENDIDO, current_state,fsm_in,dinero_out,temp_out, reset)
   begin
   next_state<=current_state;
       case current_state is
        when APAGADO =>
           fsm_out_display<='0'; --Display apagado
           fsm_out_contador<='0'; --Contador apagado
           fsm_out_temporizador<='0'; --Temporizador apagado
           if ENCENDIDO = '1' then
               next_state <= ESPERA;
           else
              next_state<=apagado;
           end if;
        when ESPERA =>
            fsm_out_display<='0'; --Display apagado
           fsm_out_contador<='0'; --Contador apagado
           fsm_out_temporizador<='0'; --Temporizador apagado

           if Encendido='0' then
              next_state<=apagado;
           elsif fsm_in = "01" then --Si pulsamos opción 1

              next_state <= PAGAR;
           elsif fsm_in = "10" then --Si pulsamos opción 2
              next_state <= PAGAR;

           else
              next_state<=ESPERA; --Si no pulsamos nada sigue en espera
           end if;
when PAGAR =>

              fsm_out_contador<='1'; --Se enciende el contador de monedas
              fsm_out_display<='1'; --Se enciende el display para mostrar la cuenta
if Encendido='0' then
              next_state<=apagado;
elsif Encendido= '1' and Dinero_out= '1' then --Si está encendido y la cuenta llega a el valor de coste
              fsm_out_contador<='0'; --Se apaga el contador
              fsm_out_display<='1'; --El display sigue encendido mostrando el cambio
              next_state<=espera2;
else
  next_state<=pagar; --Si no introducimos el dinero seguimos en el estado de pagar

end if;
when ESPERA2 =>

 fsm_out_temporizador<='1'; --Se enciende el temporizador
 fsm_out_contador<='0'; --El contador sigue apagado
 fsm_out_display<='1'; --El display sigue encendido con el cambio


if temp_out = '1'  then --Cuando el temporizador acaba
next_state <= LISTO;
end if;
if reset= '0' then --Si pulsamos el reset volvemos a espera
next_state <= ESPERA;
end if;

when LISTO =>


if reset= '0' and (fsm_in = "01" or fsm_in = "10") then --Si pulsamos el reset volvemos a espera y nos devuelve el cambio
next_state <= ESPERA;
else
next_state<= LISTO; --Si no pulsamos nada seguimos en listo
end if;
when others =>
next_state <= ESPERA;
end case;
end process;
```

En esta última parte del código, tendremos los LEDs que se encenderían en función de cada estado.

```
output_decod: process (current_state)
begin
LED <= (OTHERS => '0');
LED_AZUL <= '0';
case current_state is --Encendera los LEDs dependiendo del estado en el que estemos
   when APAGADO =>
            led(3)<='0';
            led(2)<='0';
            led(1)<='0';
            led(0)<='1';
            LED_AZUL <= '0';
   when ESPERA =>
            led(3)<='0';
            led(2)<='0';
            led(1)<='1';
            led(0)<='0';
            LED_AZUL <= '0';
   when PAGAR =>
            led(3)<='0';
            led(2)<='1';
            led(1)<='0';
            led(0)<='0';
            LED_AZUL <= '0';
   when ESPERA2 =>
            led(3)<='1';
            led(2)<='0';
            led(1)<='0';
            led(0)<='0';
            LED_AZUL <= '0';
   when LISTO =>
            led(3)<='0';
            led(2)<='0';
            led(1)<='0';
            led(0)<='0';
            LED_AZUL <= '1';
 end case;
 end process;
end behavioral;
```

Decodificador

Para que el usuario pueda ver el dinero que se ha ido introduciendo, haremos uso del display de siete segmentos.



	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0

Esta es la manera de representar cada uno de los números usando un display de 7 segmentos. Nosotros queremos representar 3 números usando 3 displays diferentes.

Por cómo es nuestra placa, todos los displays están diseñados para mostrar el mismo número, por lo tanto, para solucionar este problema, debemos añadir una señal de reloj auxiliar. En la programación, haremos que con 3 ciclos de reloj, en cada ciclo solo se encienda uno de los displays.

Para representar 080 por ejemplo, se representará el primer 0 en el primer ciclo de reloj, el 8 en el segundo y el último 0 en el tercer ciclo de reloj. Esto es imperceptible al ojo humano, pues los ciclos de reloj son extremadamente rápidos.

En esta entidad tendremos las entradas correspondientes al número introducido de monedas, el chip enable y el reloj. Además, tendremos de salidas los segmentos del BCD y los displays de 7 segmentos que se activarán.

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

ENTITY decoder IS
generic(
  DISPLAY: positive; --Displays de 7 segmentos disponibles
  SEGMENTOS: positive; --segmentos del BCD
  MONEDAS: positive --Número de bits necesarios para representar el valor, en nuestro caso será 4
  --porque el mayor número que necesitamos representar es 9
);
PORT (
decod_in1 : IN std_logic_vector(MONEDAS-1 DOWNTO 0);
decod_in2 : IN std_logic_vector(MONEDAS-1 DOWNTO 0);
decod_in3 : IN std_logic_vector(MONEDAS-1 DOWNTO 0);
led : OUT std_logic_vector(SEGMENTOS-1 DOWNTO 0);
displays: out std_logic_vector(DISPLAY-1 downto 0);
CE: in std_logic;
Clk: in std_logic
);
END ENTITY decoder;
ARCHITECTURE dataflow OF decoder IS

    signal reloj2: std_logic:='0';
BEGIN
    process(clk,CE)
        subtype ciclos is integer range 0 to 10**5;

        variable reloj: ciclos;

    begin

        if rising_edge(clk) and CE='1' then
            reloj:=reloj+1;
            if reloj=10**5-1 then
                reloj:=0;
                reloj2<=not reloj2;
            end if;
        end if;
    end process;
```

```
process(reloj2,CE,clk)
subtype display_encender is integer range 0 to 4;
 variable display_encendido: display_encender:=3;
begin
if ce='0' then
        display_encendido:=3;
        displays<="11111111";
else if rising_edge(reloj2) then
    CASE display_encendido is
        when 0=>
            displays<="11111011";-- como la seleccion esta negada ponemos todas las cifras a 1 menos la 3 que son los euros
            case decod_in1 is
                WHEN "0000"=>
                    led <= "0000001";
                WHEN "0001"=>
                    led <="1001111";
                WHEN "0010"=>
                    led <="0010010";
                WHEN "0011"=>
                    led <="0000110";
                WHEN others=>
                    led <="0000001";
            end case;
        when 1=>
            displays<="11111101";
            case decod_in2 is
                WHEN "0000"=>
                    led <="0000001";
                WHEN "0001"=>
                    led <="1001111";
                WHEN "0010"=>
                    led <="0010010";
                WHEN "0011"=>
                    led <="0000110";
                WHEN "0100"=>
                    led <="1001100";
                 WHEN "0101"=>
                    led <="0100100";
                WHEN "0110"=>
                    led <="0100000";
                WHEN "0111"=>
                    led <="0001111";
                WHEN "1000"=>
                    led <="0000000";
                WHEN "1001"=>
                    led <="0000100";
                WHEN others=>
                    led <="0000001";
            end case;
        when 2=>
            displays<="11111110";
            led<="0000001";
        when others=>
            displays<="11111111";
            led<="1111111";
     end case;
   Display_encendido:=Display_encendido+1;
     if  Display_encendido=4 then
         Display_encendido:=0;
     end if;
 end if;
end if;
end process;
end architecture dataflow;
```

Temporizador

Como hemos dicho en entidades anteriores, cada una de las opciones lleva asociada un tiempo de servicio, por lo tanto, necesitamos un temporizador que administre estos tiempos.

Para poder contar el tiempo, nos basamos en una señal de reloj. Como nuestra placa tiene una frecuencia de 100MHz, tiene un periodo asociado de 10^{-8} segundos. Esto quiere decir, que cuando contamos 10^8 flancos positivos de reloj, transcurrirá 1 segundo. Para nuestras opciones usaremos temporizadores de 3 segundos y 5 segundos.

El código tiene entradas reset, clk y chip enable como algunas de las anteriores entidades, pero, además, tiene una entrada asíncrona que depende de la opción que hayamos elegido. Como salida, tenemos temp_out, que es la que nos marcará cuando el temporizador ha acabado.

El código empleado es el siguiente:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity temporizador is
generic(
OPCIONES: positive --Opciones de productos disponibles
);
port(
    Clk     : in std_logic;
    Reset    : in std_logic;
    Opcion: in std_logic_vector(0 to OPCIONES-1);
    Temp_out: out std_logic;
    CE: in std_logic
    );

end entity temporizador;

architecture behavioral of temporizador is

    signal temp_out1: std_logic;
begin

    process(Clk,CE,reset) is
    subtype ciclos is integer range 0 to 10**8;
    variable Ticks : ciclos; --Flancos positivos de reloj
    variable Segundos: integer;
    constant t1: integer:= 3; --Segundos de la opción 1
    constant t2: integer:= 5; --Segundos de la opción 2
    begin
      if CE='0' or reset='0' then
        Ticks   := 0;
        Segundos := 0;
        temp_out1<='0';
      else if rising_edge(Clk) and CE='1' and reset='1' then
                Ticks := Ticks + 1; --Vamos sumando flancos
                if Ticks = 10**8 - 1 then --Cuando flancos llega a 10^8
                    Ticks := 0; --Ponemos ticks a 0
                    Segundos := Segundos + 1; --Sumamos 1 segundo
                case Opcion is
                when "01" => --Si hemos elegido anteriormente opción 1
                    if Segundos=t1 then --Cuando segundos es igual a los segundos de la opción 1
                        temp_out1<='1'; --Temporizador ha acabado
                    end if;
                when "10" => --Si hemos elegido anteriormente opción 2
                    if Segundos=t2 then --Cuando segundos es igual a los segundos de la opción 2
                        temp_out1<='1'; --Temporizador ha acabado
                    end if;
                when others =>
                    temp_out1<='0';
            end case;
            end if;
            end if;
        end if;

    end process;
  temp_out<=temp_out1; --Señal final del temporizador
end architecture;
```

Testbench

Antes de introducir el código a la placa, haremos los códigos de prueba correspondiente a cada entidad, para así no dañar la placa y encontrar los posibles errores. Esta prueba, nos ayudará a corregir errores en el código por separado.

Vamos a ir haciendo los testbench cada vez que acabemos de programar una entidad. A continuación se muestran dichos códigos por entidades:

Sincronizador_tb

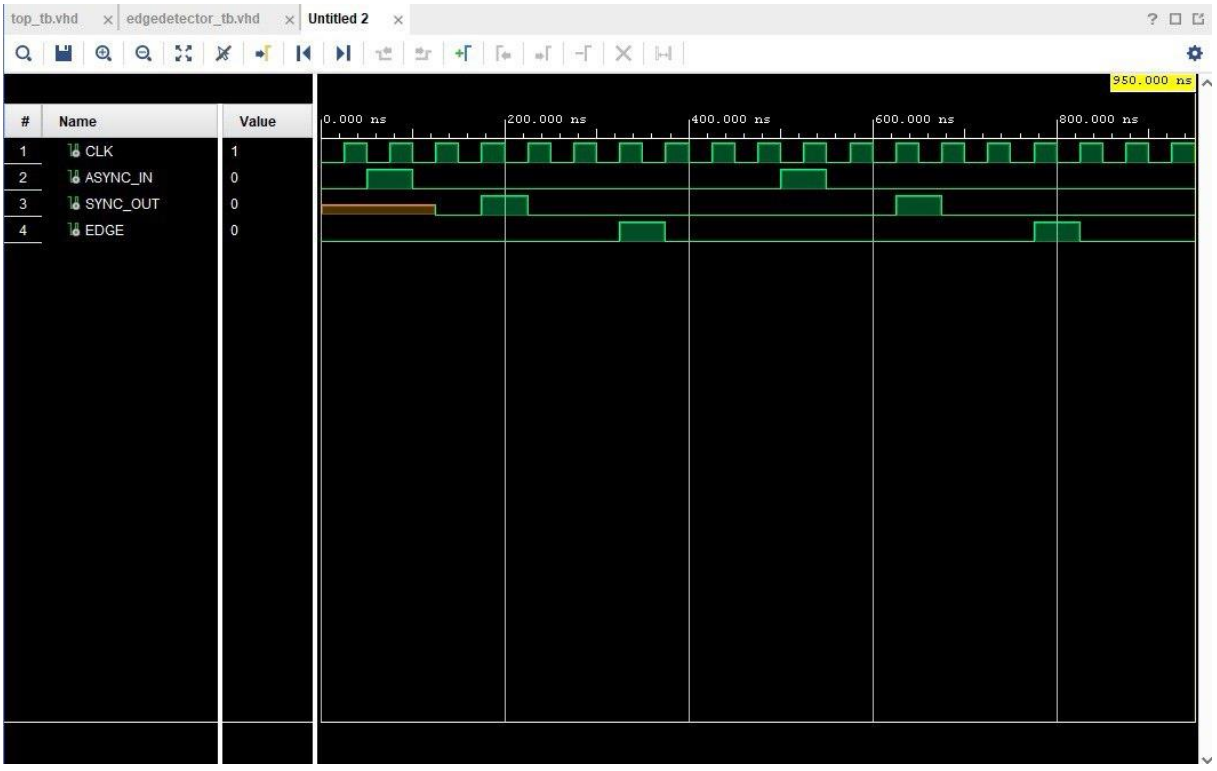
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


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

architecture tb of sync_tb is

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

    signal CLK      : std_logic;
    signal ASYNC_IN : std_logic;
    signal SYNC_OUT : std_logic;

 begin
    process
    begin
        clk<='0';
        wait for 25ns;
        clk<='1';
        wait for 25ns;
    end process;

    dut : SYNCHRNZR
    port map (CLK      => CLK,
              ASYNC_IN => ASYNC_IN,
              SYNC_OUT => SYNC_OUT);
    stimuli : process
    begin
       ASYNC_IN <= '0';
       wait for 50ns;
        async_in<='1';
        wait for 50ns;
         ASYNC_IN <= '0';
        wait for 50ns;
    begin
       ASYNC_IN <= '0';
       wait for 50ns;
        async_in<='1';
        wait for 50ns;
         ASYNC_IN <= '0';
        wait for 50ns;
        async_in<='1';
         wait for 50ns;
         ASYNC_IN <= '0';
         wait for 50ns;
        async_in<='1';
        wait for 50ns;
         ASYNC_IN <= '0';
        wait for 50ns;
        async_in<='1';
         wait for 50ns;
         ASYNC_IN <= '0';
         wait for 200ns;
         assert false
            report "Simulacion finalizada."
            severity failure;
          end process;

end tb;
```

Edgedetector_tb

```
library ieee;
use ieee.std_logic_1164.all;

entity edgedetector_tb is
end edgedetector_tb;

architecture tb of edgedetector_tb is
    component SYNCHRNZR
        port (CLK      : in std_logic;
              ASYNC_IN : in std_logic;
              SYNC_OUT : out std_logic);
    end component;
    component EDGEDTCTR
        port (CLK      : in std_logic;
              SYNC_IN  : in std_logic;
              EDGE     : out std_logic);
    end component;


    signal CLK      : std_logic;
    signal ASYNC_IN : std_logic;
    signal SYNC_OUT : std_logic;
    signal EDGE     : std_logic;
```

```
begin
  process
  begin
      clk<='0';
      wait for 25ns;
      clk<='1';
      wait for 25ns;
  end process;
  dut1 : SYNCHRNZR
  port map (CLK      => CLK,
            ASYNC_IN => ASYNC_IN,
            SYNC_OUT => SYNC_OUT);
  dut2 : EDGEDTCTR
  port map (CLK      => CLK,
            SYNC_IN  => SYNC_OUT,
            EDGE     => EDGE);


  process
  begin
     ASYNC_IN <= '0';
     wait for 50ns;
      async_in<='1';
      wait for 50ns;
       ASYNC_IN <= '0';
       wait for 400ns;
      async_in<='1';
      wait for 50ns;
       ASYNC_IN <= '0';
       wait for 400ns;
assert false
          report "Simulacion finalizada."
          severity failure;
  end process;

end tb;
```

El resultado del Testbench de la entidad sincronizador y detector de flancos juntos es el siguiente:

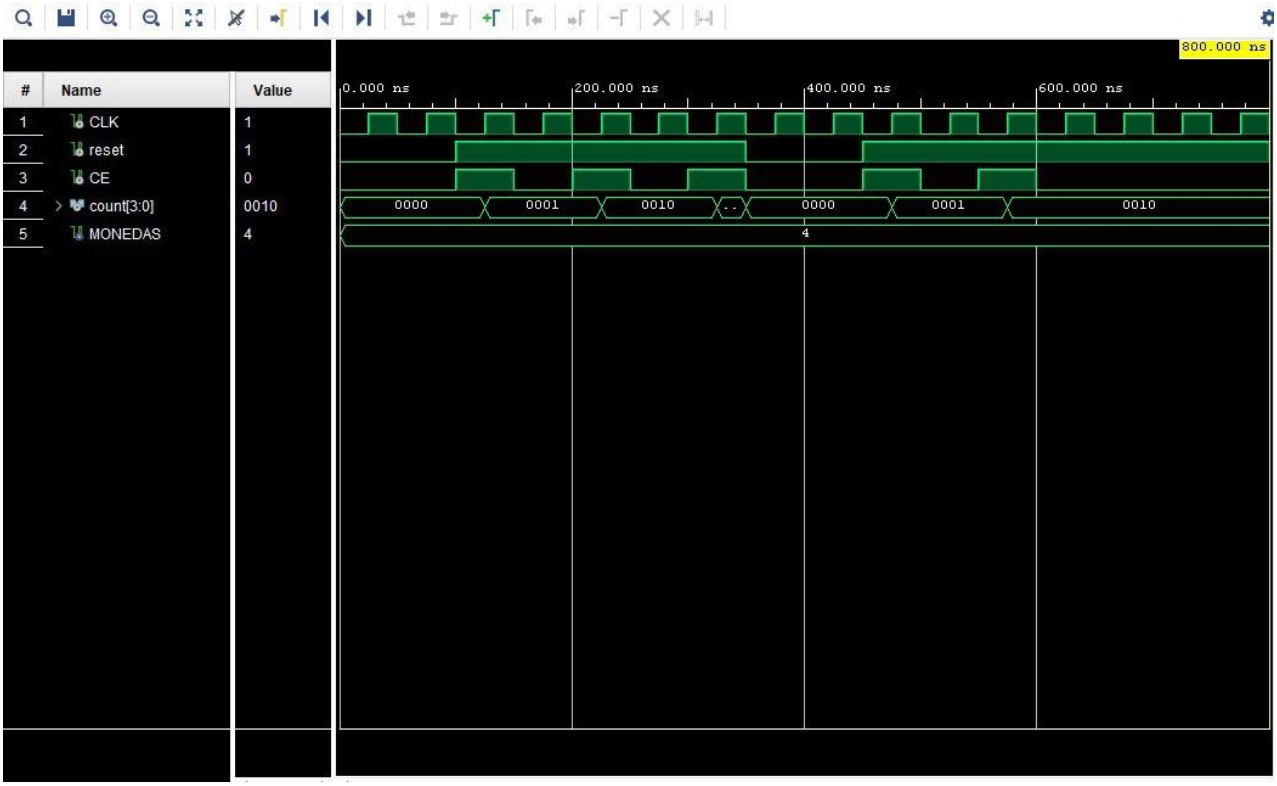


Contador_tb

```
library ieee;
use ieee.std_logic_1164.all;

entity counter_tb is
end counter_tb;

architecture tb of counter_tb is

    component counter
    generic (MONEDAS: positive); --Número máximo que puede contar (4 bits serían hasta 15)
        port (CLK   : in std_logic;
              reset : in std_logic;
              CE    : in std_logic;
              count : out std_logic_vector (MONEDAS-1 downto 0));
    end component;
    constant MONEDAS: positive:=4;
    signal CLK   : std_logic;
    signal reset : std_logic;
    signal CE    : std_logic;
    signal count : std_logic_vector (MONEDAS-1 downto 0);
  begin
    process
    begin
        clk<='0';
        wait for 25ns;
        clk<='1';
        wait for 25ns;
    end process;
    dut : counter
    generic map(MONEDAS)
    port map (CLK   => CLK,
              reset => reset,
              CE    => CE,
              count => count);
```

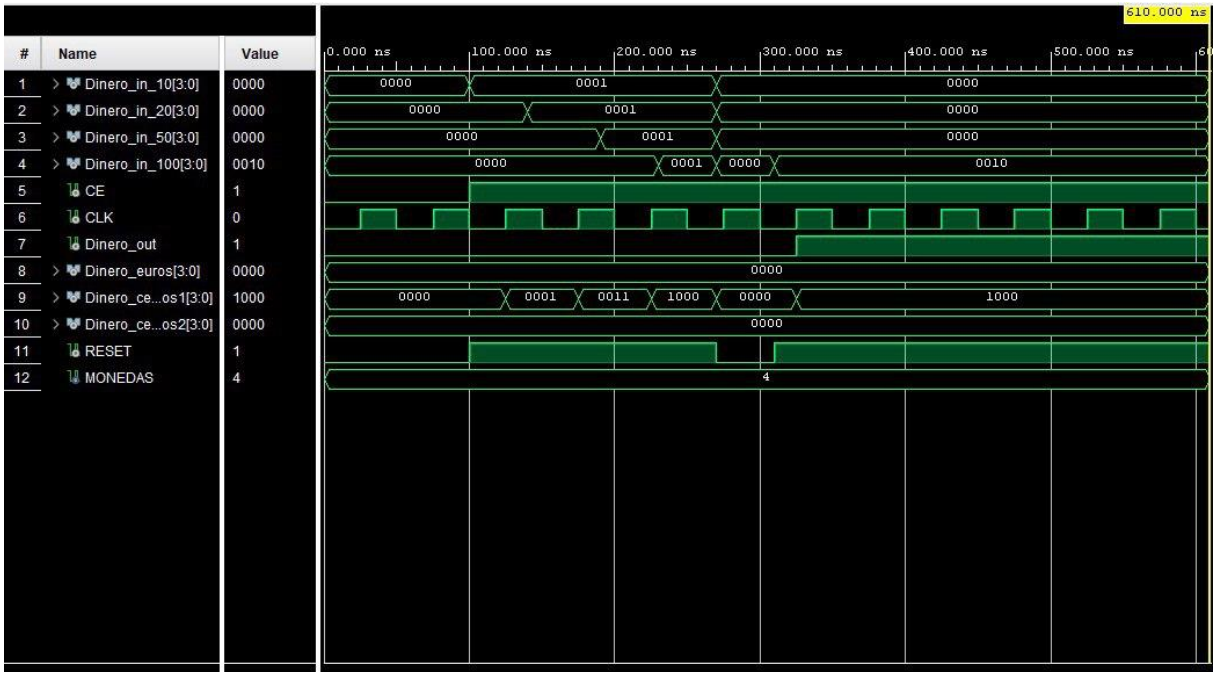
```
    stimuli : process
    begin

        CE <= '0';
        reset <= '0';
        wait for 100 ns;
        reset <= '1';
        CE <= '1';
        wait for 50 ns;
        CE <= '0';
        wait for 50 ns;
        CE <= '1';
        wait for 50 ns;
        CE <= '0';
        wait for 50 ns;
        CE <= '1';
        wait for 50 ns;
        CE<='0';
        reset<='0';
        wait for 100 ns;
        reset <= '1';
        CE <= '1';
        wait for 50 ns;
        CE <= '0';
        wait for 50 ns;
        CE <= '1';
        wait for 50 ns;
        CE <= '0';
              wait for 200 ns;
         assert false
            report "Simulacion finalizada."
            severity failure;
    end process;

end tb;
```

El resultado del testbench es el siguiente:



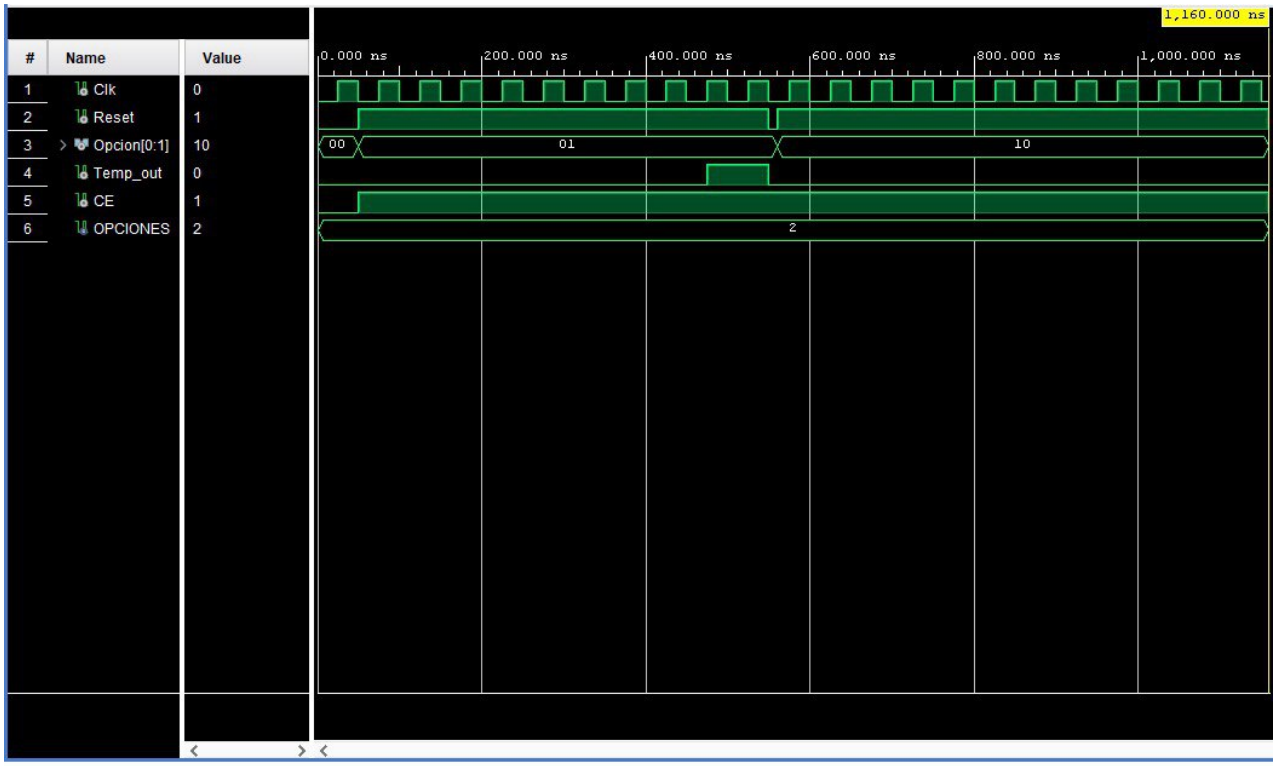
Dinero_tb

```
library ieee;
use ieee.std_logic_1164.all;

entity Dinero_tb is
end Dinero_tb;

architecture tb of Dinero_tb is

    component DINERO
    generic(
 MONEDAS: positive --Número de bits necesarios para representar el valor, en nuestro caso será 4
  --porque el mayor número que necesitamos representar es 9
);
        port (Dinero_in_10     : in std_logic_vector (MONEDAS-1 downto 0);
              Dinero_in_20     : in std_logic_vector (MONEDAS-1 downto 0);
              Dinero_in_50     : in std_logic_vector (MONEDAS-1 downto 0);
              Dinero_in_100    : in std_logic_vector (MONEDAS-1 downto 0);
              CE               : in std_logic;
              CLK              : in std_logic;
              Dinero_out       : out std_logic;
              Dinero_euros     : out std_logic_vector (MONEDAS-1 downto 0);
              Dinero_centimos1 : out std_logic_vector (MONEDAS-1 downto 0);
              Dinero_centimos2 : out std_logic_vector (MONEDAS-1 downto 0);
              RESET            : in std_logic);
    end component;
    constant MONEDAS: positive:=4;
    signal Dinero_in_10     : std_logic_vector (MONEDAS-1 downto 0);
    signal Dinero_in_20     : std_logic_vector (MONEDAS-1 downto 0);
    signal Dinero_in_50     : std_logic_vector (MONEDAS-1 downto 0);
    signal Dinero_in_100    : std_logic_vector (MONEDAS-1 downto 0);
    signal CE               : std_logic;
    signal CLK              : std_logic;
    signal Dinero_out       : std_logic;
    signal Dinero_euros     : std_logic_vector (MONEDAS-1 downto 0);
    signal Dinero_centimos1 : std_logic_vector (MONEDAS-1 downto 0);
    signal Dinero_centimos2 : std_logic_vector (MONEDAS-1 downto 0);
    signal RESET            : std_logic;

begin
    process
    begin
        clk<='0';
        wait for 25ns;
        clk<='1';
        wait for 25ns;
    end process;
    dut : DINERO
    generic map(MONEDAS)
    port map (Dinero_in_10     => Dinero_in_10,
              Dinero_in_20     => Dinero_in_20,
              Dinero_in_50     => Dinero_in_50,
              Dinero_in_100    => Dinero_in_100,
              CE               => CE,
              CLK              => CLK,
              Dinero_out       => Dinero_out,
              Dinero_euros     => Dinero_euros,
              Dinero_centimos1 => Dinero_centimos1,
              Dinero_centimos2 => Dinero_centimos2,
              RESET            => RESET);

    stimuli : process
    begin

        Dinero_in_10 <= (others => '0');
        Dinero_in_20 <= (others => '0');
        Dinero_in_50 <= (others => '0');
        Dinero_in_100 <= (others => '0');
        CE <= '0';
        RESET <= '0';
        wait for 100 ns;
        reset<='1';
        Ce<='1';
        Dinero_in_10<="0001";
        wait for 40ns;
        Dinero_in_20<="0001";
        wait for 50ns;
        Dinero_in_50<="0001";
        wait for 40ns;
        Dinero_in_100<="0001";
        wait for 40ns;
        reset<='0';
       Dinero_in_20<="0000";
       Dinero_in_50<="0000";
       Dinero_in_100<="0000";
       wait for 40ns;
       reset<='1';
       Dinero_in_100<="0010";
       wait for 300ns;
       assert false
           report "Simulacion finalizada."
           severity failure;
       end process;

end tb;
```



Temporizador_tb

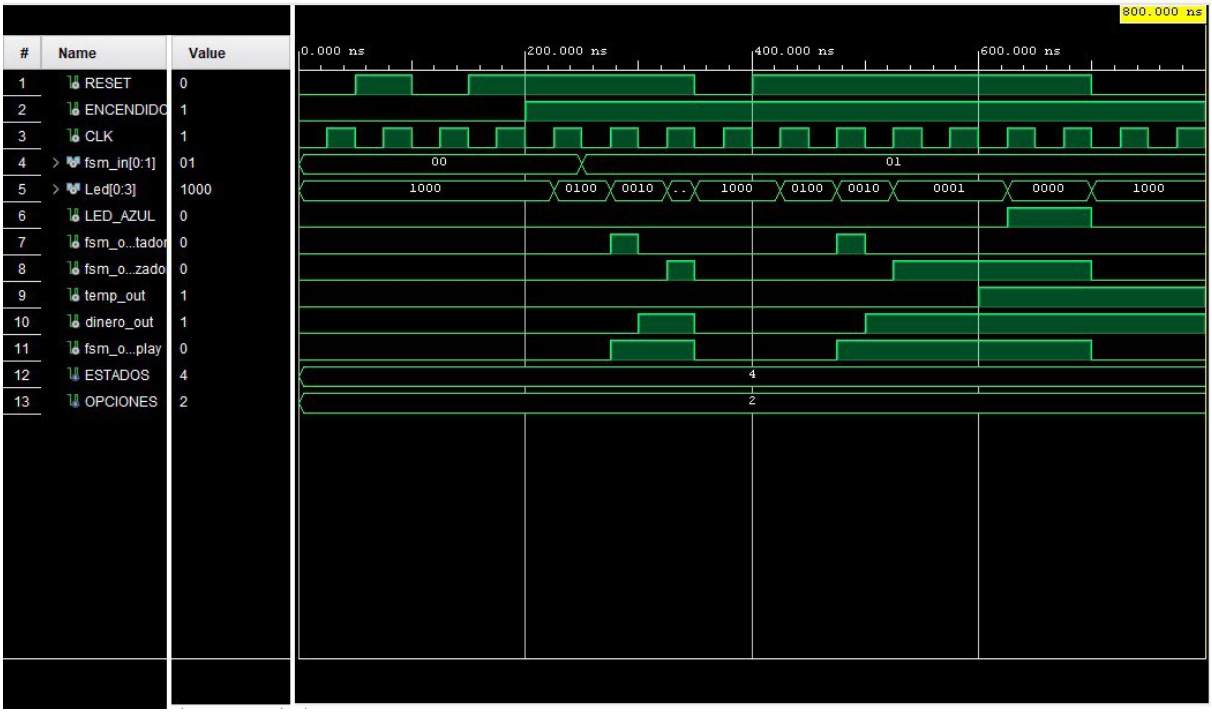
```
library ieee;
use ieee.std_logic_1164.all;

entity temp_tb is
end temp_tb;

architecture tb of temp_tb is

    component temporizador
    generic(
OPCIONES: positive --Opciones de productos disponibles
);
        port (Clk      : in std_logic;
              Reset    : in std_logic;
              Opcion   : in std_logic_vector (0 to OPCIONES-1);
              Temp_out : out std_logic;
              CE       : in std_logic);
    end component;
    constant OPCIONES: positive:=2;
    signal Clk      : std_logic;
    signal Reset    : std_logic;
    signal Opcion   : std_logic_vector (0 to OPCIONES-1);
    signal Temp_out : std_logic;
    signal CE       : std_logic;

begin
    process
    begin
        clk<='0';
        wait for 25ns;
        clk<='1';
        wait for 25ns;
    end process;
    dut : temporizador
    generic map(OPCIONES)
    port map (Clk      => Clk,
              Reset    => Reset,
              Opcion   => Opcion,
              Temp_out => Temp_out,
              CE       => CE);

    stimuli : process
    begin

        Reset <= '0';
        Opcion <= (others => '0');
        CE <= '0';
        wait for 50ns;
        CE<='1';
        reset<='1';
        Opcion<="01";
        wait for 500ns;
        reset<='0';
        wait for 10ns;
        reset<='1';
        Opcion<="10";
        wait for 600ns;

    assert false
            report "Simulacion finalizada."
            severity failure;
    end process;

end tb;
```



Fsm_tb

```
library ieee;
use ieee.std_logic_1164.all;

entity fsm_tb is
end fsm_tb;

architecture tb of fsm_tb is

    component fsm
    generic(
  ESTADOS: positive;--número de estados totales representados en los leds
  OPCIONES: positive --Opciones de productos disponibles
  );
        port (RESET                : in std_logic;
              ENCENDIDO            : in std_logic;
              CLK                  : in std_logic;
              fsm_in               : in std_logic_vector (0 to OPCIONES-1);
              Led                  : out std_logic_vector (0 to ESTADOS-1);
              LED_AZUL             : out std_logic;
              fsm_out_contador     : out std_logic;
              fsm_out_temporizador : out std_logic;
              temp_out             : in std_logic;
              dinero_out           : in std_logic;
              fsm_out_display      : out std_logic);
    end component;
    constant ESTADOS:positive:=4;
    constant OPCIONES:positive:=2;
    signal RESET                : std_logic;
    signal ENCENDIDO            : std_logic;
    signal CLK                  : std_logic;
    signal fsm_in               : std_logic_vector (0 to OPCIONES-1);
    signal Led                  : std_logic_vector (0 to ESTADOS-1);
    signal LED_AZUL             : std_logic;
    signal fsm_out_contador     : std_logic;
    signal fsm_out_temporizador : std_logic;
    signal temp_out             : std_logic;
    signal dinero_out           : std_logic;
    signal fsm_out_display      : std_logic;
```

```
begin
    process
    begin
        clk<='0';
        wait for 25ns;
        clk<='1';
        wait for 25ns;
    end process;
    dut : fsm
    generic map(ESTADOS, OPCIONES)
    port map (RESET                => RESET,
              ENCENDIDO            => ENCENDIDO,
              CLK                  => CLK,
              fsm_in               => fsm_in,
              Led                  => Led,
              LED_AZUL             => LED_AZUL,
              fsm_out_contador     => fsm_out_contador,
              fsm_out_temporizador => fsm_out_temporizador,
              temp_out             => temp_out,
              dinero_out           => dinero_out,
              fsm_out_display      => fsm_out_display);

    stimuli : process
    begin
        -- EDIT Adapt initialization as needed
        RESET <= '0';
        ENCENDIDO <= '0';
        fsm_in <= (others => '0');
        temp_out <= '0';
        dinero_out <= '0';
        wait for 50ns;
         RESET <= '1';
         wait for 50ns;
         RESET <= '0';
         wait for 50ns;
         RESET <= '1';
         wait for 50ns;
        ENCENDIDO <= '1';
        wait for 50ns;
        fsm_in <="01";
        wait for 50ns;
        dinero_out<='1';
        wait for 50ns;
        RESET <= '0';
        dinero_out<='0';
         wait for 50ns;
         RESET <= '1';
         wait for 100ns;
         dinero_out<='1';
         wait for 100ns;
         temp_out<='1';
         wait for 100ns;
         reset<='0';
         wait for 100ns;
        assert false
            report "Simulacion finalizada."
            severity failure;
    end process;

end tb;
```

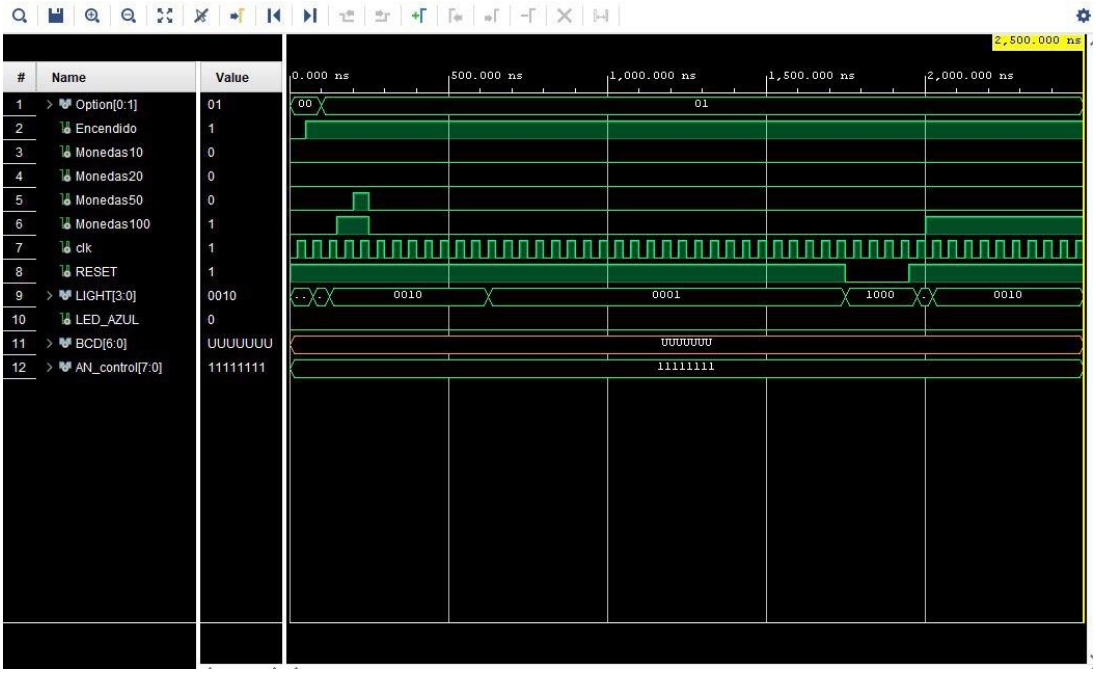


Decoder_tb

```
library ieee;
use ieee.std_logic_1164.all;

entity decoder_tb is
end decoder_tb;

architecture tb of decoder_tb is

    component decoder
    generic(
  DISPLAY: positive; --Displays de 7 segmentos disponibles
  SEGMENTOS: positive; --segmentos del BCD
  MONEDAS: positive --Número de bits necesarios para representar el valor, en nuestro caso será 4
  --porque el mayor número que necesitamos representar es 9
);
        port (decod_in1 : in std_logic_vector (MONEDAS-1 downto 0);
              decod_in2 : in std_logic_vector (MONEDAS-1 downto 0);
              decod_in3 : in std_logic_vector (MONEDAS-1 downto 0);
              led       : out std_logic_vector (SEGMENTOS-1 downto 0);
              displays  : out std_logic_vector (DISPLAY-1 downto 0);
              CE        : in std_logic;
              Clk       : in std_logic);
    end component;
    constant DISPLAY: positive:=8;
    constant SEGMENTOS: positive:=7;
    constant MONEDAS: positive:=4;
    signal decod_in1 : std_logic_vector (3 downto 0);
    signal decod_in2 : std_logic_vector (MONEDAS-1 downto 0);
    signal decod_in3 : std_logic_vector (MONEDAS-1 downto 0);
    signal led       : std_logic_vector (SEGMENTOS-1 downto 0);
    signal displays  : std_logic_vector (DISPLAY-1 downto 0);
    signal CE        : std_logic;
    signal Clk       : std_logic;

begin
    process
    begin
        clk<='0';
        wait for 25ns;
        clk<='1';
        wait for 25ns;
    end process;
    dut : decoder
    generic map(DISPLAY,SEGMENTOS,MONEDAS)
    port map (decod_in1 => decod_in1,
              decod_in2 => decod_in2,
              decod_in3 => decod_in3,
              led       => led,
              displays  => displays,
              CE        => CE,
              Clk       => Clk);

    stimuli : process
    begin

        decod_in1 <= (others => '0');
        decod_in2 <= (others => '0');
        decod_in3 <= (others => '0');
        CE <= '0';
        wait for 50ns;
        decod_in1<="0001";
        decod_in2<="0010";
        decod_in3<="0011";
        wait for 50ns;
         CE <= '1';
         decod_in1<="0001";
        decod_in2<="0010";
        decod_in3<="0011";
        wait for 700ns;
        ce<='0';
        wait for 100ns;
         ce<='1';
         wait for 500ns;
        -- EDIT Add stimuli here
        assert false
            report "Simulacion finalizada."
            severity failure;
    end process;

end tb;
```

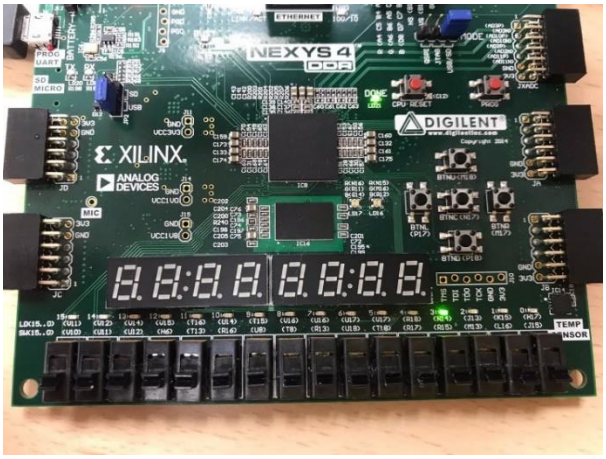
Top_tb

```
library ieee;
use ieee.std_logic_1164.all;

entity top_tb is
end top_tb;

architecture tb of top_tb is

    component TOP1
        port (Option     : in std_logic_vector (0 to 1);
              Encendido  : in std_logic;
              Monedas10  : in std_logic;
              Monedas20  : in std_logic;
              Monedas50  : in std_logic;
              Monedas100 : in std_logic;
              clk        : in std_logic;
              RESET      : in std_logic;
              LIGHT      : out std_logic_vector (3 downto 0);
              LED_AZUL   : out std_logic;
              BCD        : out std_logic_vector (6 downto 0);
              AN_control : out std_logic_vector (7 downto 0));
    end component;

    signal Option     : std_logic_vector (0 to 1);
    signal Encendido  : std_logic;
    signal Monedas10  : std_logic;
    signal Monedas20  : std_logic;
    signal Monedas50  : std_logic;
    signal Monedas100 : std_logic;
    signal clk        : std_logic;
    signal RESET      : std_logic;
    signal LIGHT      : std_logic_vector (3 downto 0);
    signal LED_AZUL   : std_logic;
    signal BCD        : std_logic_vector (6 downto 0);
    signal AN_control : std_logic_vector (7 downto 0);


begin
    process
    begin
        clk<='0';
        wait for 25ns;
        clk<='1';
        wait for 25ns;
    end process;
    dut : TOP1
    port map (Option     => Option,
              Encendido  => Encendido,
              Monedas10  => Monedas10,
              Monedas20  => Monedas20,
              Monedas50  => Monedas50,
              Monedas100 => Monedas100,
              clk        => clk,
              RESET      => RESET,
              LIGHT      => LIGHT,
              LED_AZUL   => LED_AZUL,
              BCD        => BCD,
              AN_control => AN_control);

    stimuli : process
    begin
        -- EDIT Adapt initialization as needed
        Option <= (others => '0');
        Encendido <= '0';
        Monedas10 <= '0';
        Monedas20 <= '0';
        Monedas50 <= '0';
        Monedas100 <= '0';
        RESET <= '1';
        wait for 50ns;
        Encendido<='1';
        wait for 50ns;
        Option<="01";
        wait for 50ns;
        Monedas100<='1';
       wait for 50ns;
       Monedas50<='1';
       wait for 50ns;
       Monedas100<='0';
       Monedas50<='0';
       wait for 1500ns;
       reset<='0';
       wait for 200ns;
       reset<='1';
       wait for 50ns;
       Monedas100<='1';
       wait for 500ns;
      assert false
           report "Simulacion finalizada."
           severity failure;
    end process;

end tb;
```



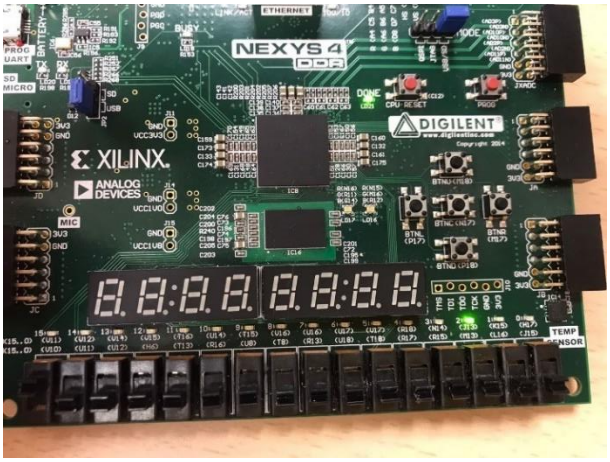
Funcionamiento

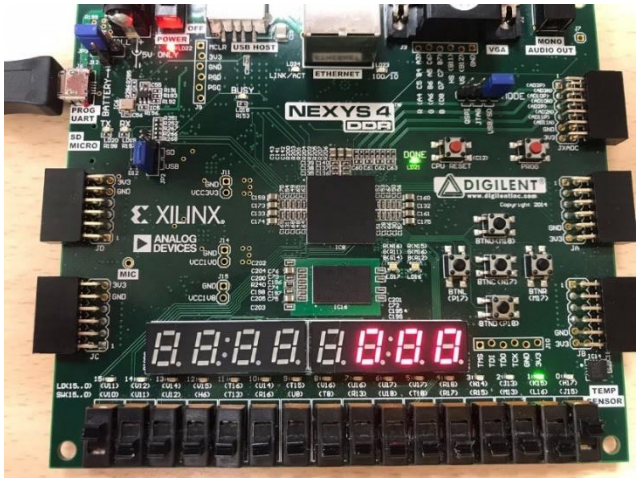
Para mostrar el funcionamiento de manera sencilla, vamos a añadir varias capturas y un enlace a un vídeo.



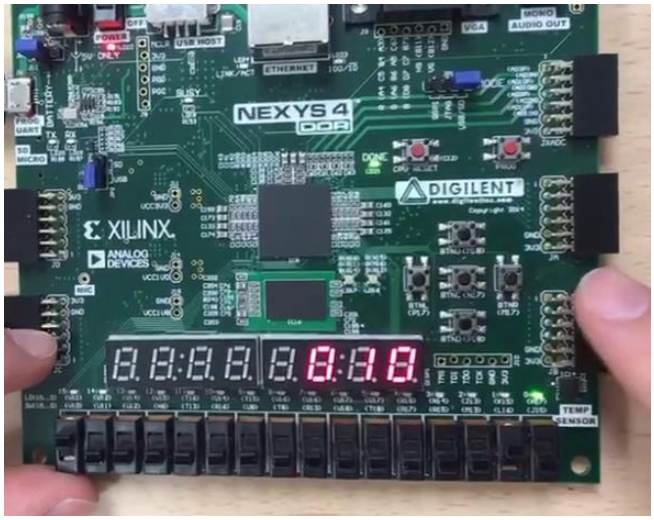
En la imagen de la izquierda podemos ver la placa en el estado ***Apagado*** pues el switch de encendido situado a la izquierda del todo no está accionado. También, se puede observar que el LED que está encendido es el que corresponde al dicho estado y que el display se encuentra apagado también.

En esta segunda imagen, nos encontramos en el estado de ***Espera*** porque hemos encendido el switch de encendido pero no hemos seleccionado la opción todavía (dos switches de a derecha apagados). Como se puede ver, se ha encendido el LED de dicho estado y no se ha encendido el display.



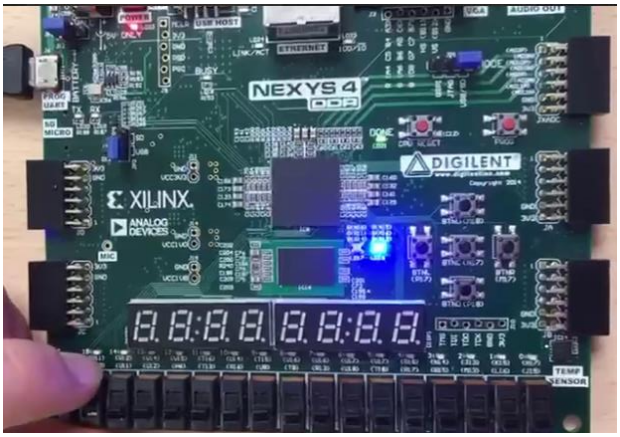


En la tercera captura, nos encontramos en el estado de ***Pagar*** pues hemos seleccionado uno de los dos switches de opciones. Se observa que se ha encendido el LED correspondiente a dicho estado así como el display para mostrar el dinero que vamos introduciendo.



Ahora estamos en el estado de ***Espera2***, en el que ya hemos introducido nuestro dinero y estamos esperando a que nos dé el producto. El display de la imagen muestra la cantidad de cambio que nos va a dar la máquina. En este momento el temporizador del tiempo a esperar está activo.

Por último, podemos observar cómo hemos llegado al estado de ***Listo,*** pues se ha encendido el LED azul que nos indica que ha acabado de servir. Además, podemos observar que se ha apagado el display, al haberse realizado el cambio.



Enlace al vídeo: https://mega.nz/file/3AJ32IqA#dg-nie6iSCAteb1Fuzh903STisj9tVw1RgoSNmTk-N4