

PROYECTO DE VHDL

MÁQUINA EXPENDEDORA DE REFRESCOS



Sergio de Paula Moratilla 55209

Carlos Perchín García 55401

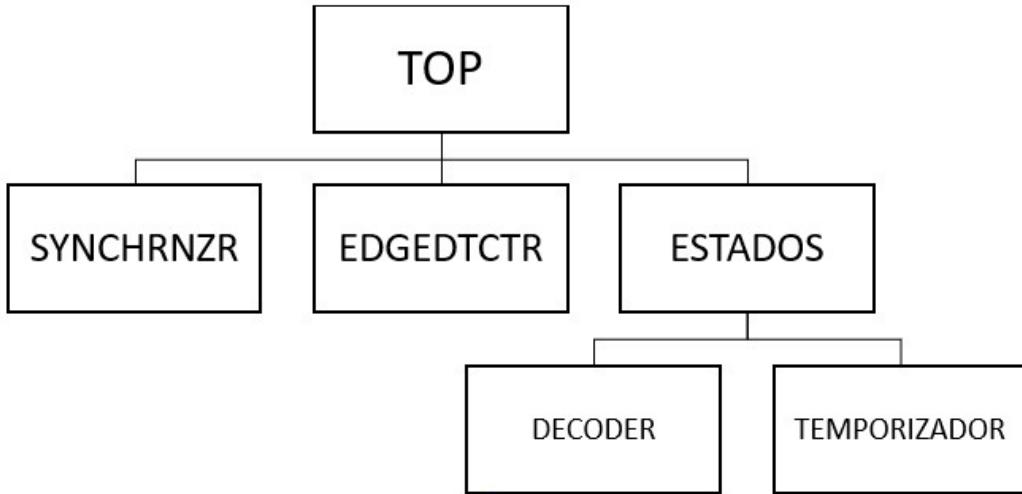
Rubén Ruiz Martínez 54935

ÍNDICE

1 ESTRUCTURA DEL CODIGO	3
2 SINCRONIZADOR.....	3
2.1 CÓDIGO VHDL	3
2.2 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)	4
3 GENERADOR DE PULSOS.....	6
3.1 CÓDIGO VHDL	6
3.2 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)	7
4 MÁQUINA DE ESTADOS	9
4.1 CÓDIGO VHDL	11
4.2 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)	14
5 DECODIFICADOR	15
5.1 REPRESENTACIÓN DE NÚMEROS EN EL DISPLAY DE 7 SEGMENTOS.....	15
5.2 REPRESENTACIÓN DEL DINERO USANDO LOS DISPLAYS DE 7 SEGMENTOS	15
5.3 CÓDIGO VHDL	16
5.4 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)	18
6 TEMPORIZADOR.....	20
6.1 CÓDIGO VHDL	20
6.2 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)	21
7 ENTIDAD TOP	23
7.1 CÓDIGO VHDL	23
7.2 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)	25
8 MAPA DE RESTRICCIONES DE LA PLACA	29
9 COMPROBACIÓN DEL FUNCIONAMIENTO EN LA PLACA.....	30

1 ESTRUCTURA DEL CODIGO

Para la elaboración de nuestra máquina de refrescos, hemos empleado la siguiente relación entre entidades:



El estilo de descripción empleado en el código ha sido una mezcla entre *Dataflow* y *Behavioral*, pues hemos utilizado tanto procesos (por ejemplo, para la implementación de la máquina de estados) como componentes (para llamar a unas entidades dentro de otras). A continuación, vamos a analizar las entidades empleadas.

2 SINCRONIZADOR

En la actualidad, prácticamente todos los circuitos digitales son síncronos. Es decir, todos los cambios de valor de las señales tienen lugar en uno de los flancos (en nuestro caso de subida) de la señal de reloj CLK.

2.1 CÓDIGO VHDL

En nuestro proyecto, las entradas (monedas y producto seleccionado) son asíncronas. Para sincronizar estas señales con el reloj, hacemos uso del “*Sincronizador*”. Su código es:

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

entity SYNCHRNR is
    GENERIC (WITDH : positive);
    PORT (
        CLK : in std_logic;
        ASYNC_IN : in std_logic_vector(WITDH - 1 downto 0);
        SYNC_OUT : out std_logic_vector(WITDH - 1 downto 0)
    );
end SYNCHRNR;

architecture BEHAVIORAL of SYNCHRNR is
    signal sreg : std_logic_vector(2*WITDH - 1 downto 0);
begin
    process (CLK)
    begin
        if rising_edge(CLK) then
            for i in 0 to WITDH - 1 loop
                SYNC_OUT(i) <= sreg(2*i + 1);
                sreg(2*i + 1 downto 2*i) <= sreg(2*i) & ASYNC_IN(i);
            end loop;
        end if;
    end process;
end BEHAVIORAL;
```

2.2 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)

Para comprobar el funcionamiento del sincronizador, hemos empleado el siguiente testbench:

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

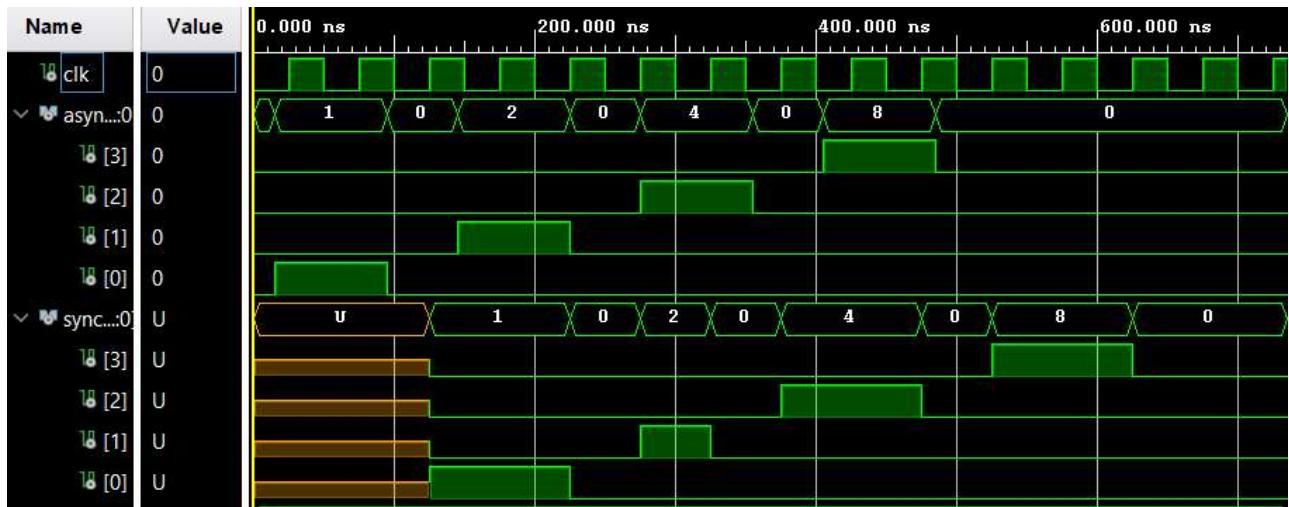
entity tb_synchrnzs is
end tb_synchrnzs;

architecture Behavioral of tb_synchrnzs is
COMPONENT SYNCHRNZR is
generic (WITDH : positive);
port (
    CLK : in std_logic;
    ASYNC_IN : in std_logic_vector(WITDH -1 downto 0);
    SYNC_OUT : out std_logic_vector(WITDH -1 downto 0)
);
END COMPONENT;
constant witzdh : positive := 4;
signal clk : std_logic;
signal async_in, sync_out : std_logic_vector(WITDH -1 downto 0);
begin
process
begin
    clk <= '0';
    wait for 25ns;
    clk <= '1';
    wait for 25ns;
end process;

Inst_sincronizador : SYNCHRNZR
generic map (witzdh)
port map(
    CLK => clk,
    ASYNC_IN => async_in,
    SYNC_OUT => sync_out
);

process
begin
    async_in <= "0000";
    wait for 15 ns;
    for i in 0 to WITDH -1 loop
        async_in(i) <= '1';
        wait for 80 ns;
        async_in(i) <= '0';
        wait for 50 ns;
    end loop;
    wait for 200ns;
    assert false
        report "Simulation finished."
        severity failure;
end process;
end Behavioral;
```

Como resultado de la simulación obtuvimos:



Se puede observar como la señal sincronizada solo cambia en los flancos positivos de reloj.

3 GENERADOR DE PULSOS

Una vez sincronizada la señal, tenemos que pasarla por un generador de pulsos. Esto sirve para que, cuando pulsemos el botón, no se produzca varias veces la suma de la cantidad introducida.

3.1 CÓDIGO VHDL

En cada flanco positivo de la señal de reloj se comprobará el valor de la señal sincronizada. En nuestro caso, hemos optado por generar el pulso en el flanco de bajada de la señal, es decir, cuando dejemos de pulsar el botón. Para evitar rebotes, emplearemos 2 ciclos con valor 0.

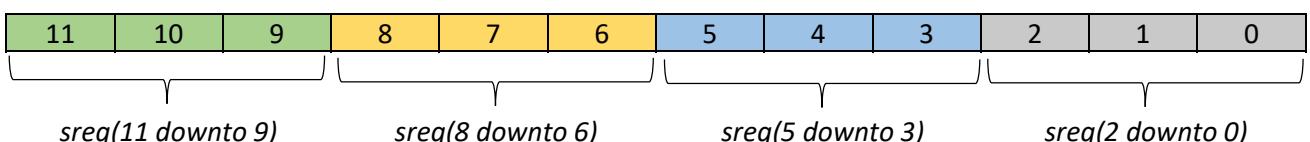
Es decir, el pulso se generará, durante 1 solo ciclo de reloj, cuando la señal valga en 3 flancos de subida de reloj consecutivos “100”. Su código es:

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

entity EDGEDTCTR is
    GENERIC (WITDH : positive);
    PORT (
        CLK : in std_logic;
        SYNC_IN : in std_logic_vector(WITDH - 1 downto 0);
        EDGE : out std_logic_vector(WITDH - 1 downto 0)
    );
end EDGEDTCTR;

architecture BEHAVIORAL of EDGEDTCTR is
    signal sreg : std_logic_vector(3*WITDH - 1 downto 0);
begin
    process (CLK)
    begin
        if rising_edge(CLK) then
            for i in 0 to WITDH - 1 loop
                sreg(3*i + 2 downto 3*i) <= sreg(3*i + 1 downto 3*i) & SYNC_IN(i);
                case sreg(3*i + 2 downto 3*i) is
                    when "100" =>
                        EDGE(i) <= '1';
                    when others =>
                        EDGE(i) <= '0';
                end case;
            end loop;
        end if;
    end process;
end BEHAVIORAL;
```

Para cada componente, necesitamos llevar constancia de los últimos 3 valores de la señal. Por eso, la dimensión de la variable auxiliar *sreg* debe ser $3 * width - 1$. Cada 3 valores, corresponde con una señal diferente:



Finalmente, comparamos el valor de *sreg* con el buscado (“100”). Como ese valor obligatoriamente solo se va a cumplir durante 1 ciclo, el pulso generado solo va a durar 1 ciclo.

3.2 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)

Para comprobar el funcionamiento del generador de pulsos lo hemos combinado con el sincronizador, tal y como vamos a hacer en la entidad top. Hemos empleado el siguiente testbench:

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

entity tb_edgedtctr is
end tb_edgedtctr;

architecture Behavioral of tb_edgedtctr is
COMPONENT SYNCHRNZR is
generic (WITDH : positive);
port (
    CLK : in std_logic;
    ASYNC_IN : in std_logic_vector(WITDH -1 downto 0);
    SYNC_OUT : out std_logic_vector(WITDH -1 downto 0)
);
END COMPONENT;
COMPONENT EDGEDTCTR is
generic (WITDH : positive);
port (
    CLK : in std_logic;
    SYNC_IN : in std_logic_vector(WITDH -1 downto 0);
    EDGE : out std_logic_vector(WITDH -1 downto 0)
);
END COMPONENT;
constant withub : positive := 4;
signal clk : std_logic;
signal async_in, sync_out, pulso : std_logic_vector(WITDH -1 downto 0);
begin
process
begin
    clk <= '0';
    wait for 25ns;
    clk <= '1';
    wait for 25ns;
end process;

Inst_sincronizador : SYNCHRNZR
generic map (WITDH)
port map(
    CLK => clk,
    ASYNC_IN => async_in,
    SYNC_OUT => sync_out
);
Inst_generador_pulsos : EDGEDTCTR
generic map (WITDH)
port map(
    CLK => clk,
    SYNC_IN => sync_out,
    EDGE => pulso
);
```

```

process
begin
    async_in <= "0000";
    wait for 15 ns;
    for i in 0 to WITDH -1 loop
        async_in(i) <= '1';
        wait for 80 ns;
        async_in(i) <= '0';
        wait for 50 ns;
    end loop;
    wait for 400ns;
    assert false
        report "Simulation finished."
        severity failure;
    end process;
end Behavioral;

```

Como resultado de la simulación obtuvimos:



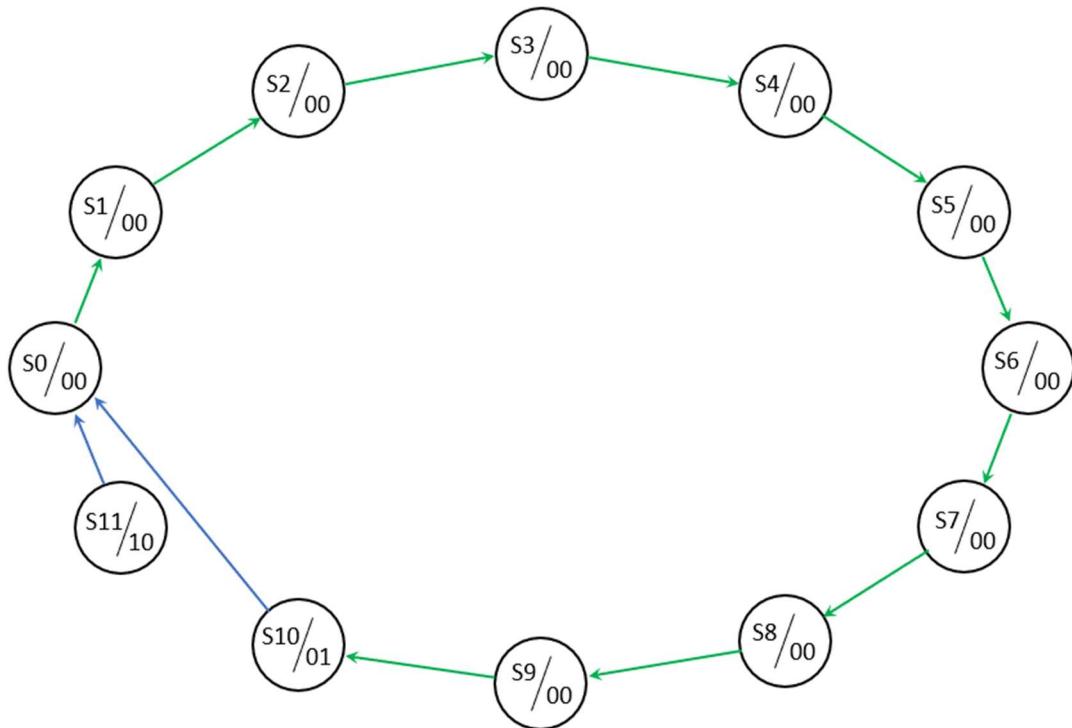
Como se puede observar, a partir de una señal asíncrona, gracias al sincronizador y al generador de pulsos, obtenemos a la salida un pulso, de duración 1 ciclo, sincronizado con el flanco de subida del reloj.

Si bien el pulso comienza varios ciclos de reloj después de dejar de pulsar el botón, hay que recordar que el reloj presenta una frecuencia muy elevada, por lo que dicho “retraso” será tan corto que, a “escala humana”, será imperceptible.

4 MÁQUINA DE ESTADOS

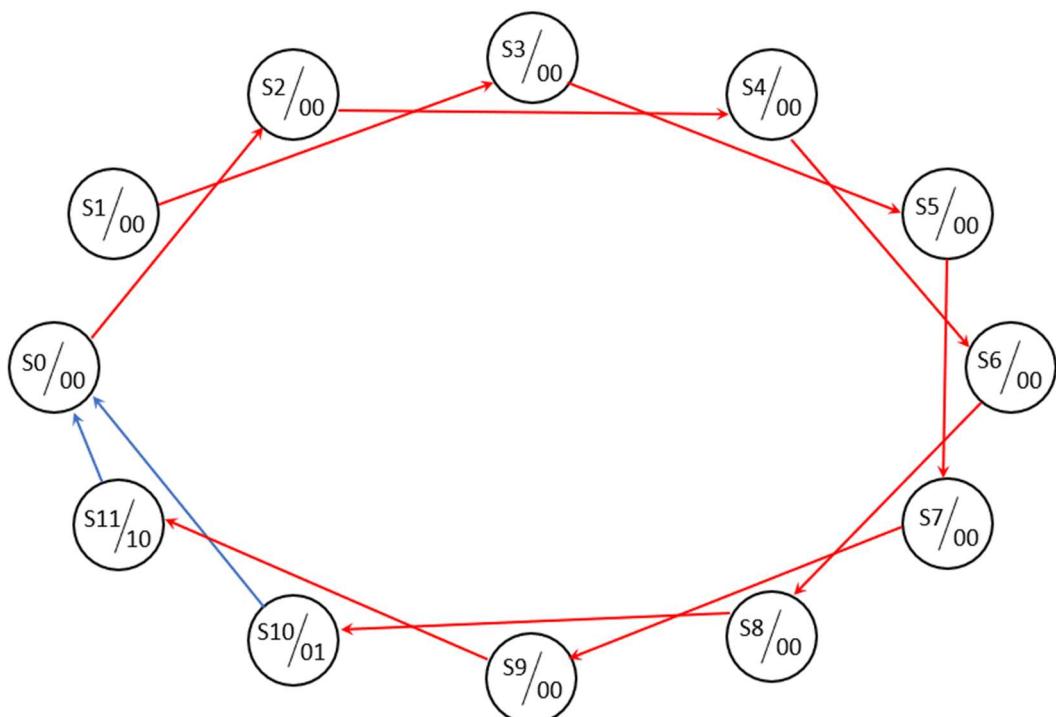
Dentro de la entidad top, tenemos una entidad encargada de controlar la evolución de los estados de nuestro sistema. Recibiendo como entradas el reloj, el reset, las monedas y los productos, nos indica el dinero introducido en cada momento y avisa cuando hemos terminado de introducir dinero.

Cambios de estado si introducimos una moneda de 10 céntimos

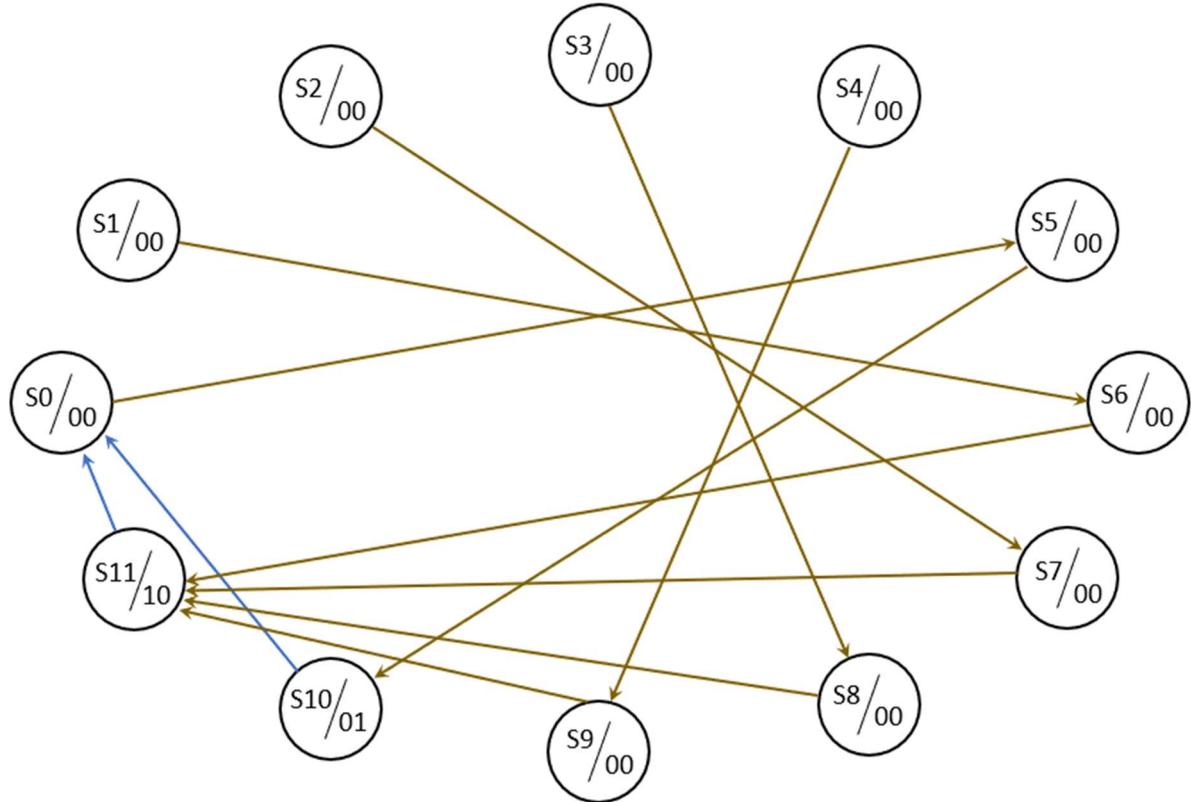


La notación empleada para las salidas es bit_dinero_incorrecto – bit_dinero_correcto

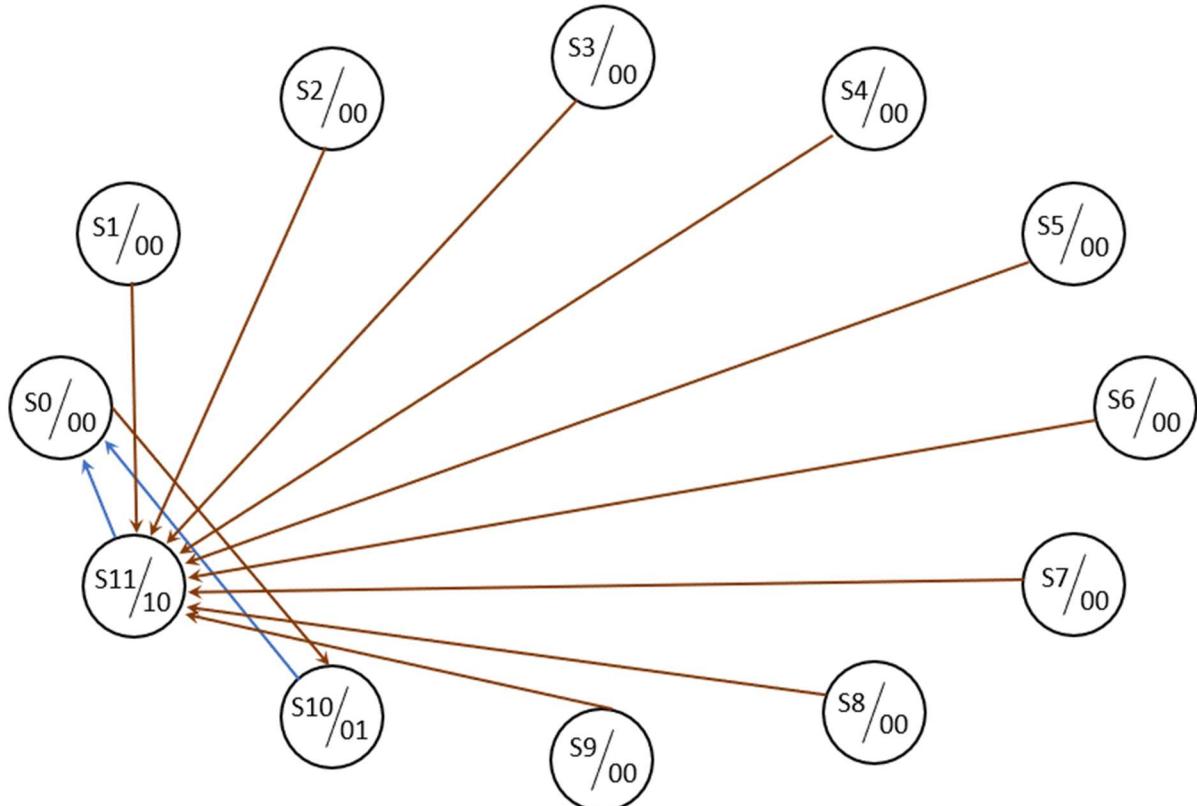
Cambios de estado si introducimos una moneda de 20 céntimos



Cambios de estado si introducimos una moneda de 50 céntimos



Cambios de estado si introducimos una moneda de 1 euro



Prácticamente todos los cambios de estado se producen cuando introducimos una moneda en la máquina. Los únicos diferentes son los cambios de S10 y S11 a S0. En ellos, el cambio es por tiempo, permaneciendo 3 segundos en dichos estados antes de producirse el cambio. Esta labor la realiza el TEMPORIZADOR.

4.1 CÓDIGO VHDL

La entidad ESTADOS, nos devolverá 3 cosas: los LEDS que debemos encender (controlado por esta entidad), los displays que debemos encender en cada momento y el dinero a representar. Estas 2 últimas labores son realizadas por la entidad DECODER, como explicaremos más adelante. El código de la entidad ESTADOS es:

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

entity ESTADOS is
    GENERIC (WITDH : positive; -- Numero total de tipos de monedas/productos
              DISPLAYS : positive; -- Numero total de displays que hay
              SEGMENTOS : positive -- Segmentos que tiene cada display
            );
    PORT ( CLK : in std_logic; -- Señal de reloj
            RESET : in std_logic; -- Señal de reset
            MONEDA : in std_logic_vector(WITDH - 1 downto 0); -- Entradas de las monedas
            PRODUCTO : in std_logic_vector(WITDH - 1 downto 0); -- Entradas de productos
            LEDS : out std_logic_vector(WITDH downto 0); -- Leds que indican que se ha terminado de introducir la cantidad
            DISPLAYS_ENCENDIDOS : out std_logic_vector(DISPLAYS - 1 downto 0); -- Displays a encender en cada momento
            SEGMENTOS_ENCENDIDOS : out std_logic_vector(SEGMENTOS - 1 downto 0) -- Segmentos a encender en cada momento
          );
end ESTADOS;

architecture Behavioral of ESTADOS is
    -- VARIABLES NECESARIAS PARA EL TEMPORIZADOR
    COMPONENT TEMPORIZADOR is
        port (
            clk : in std_logic;
            enable : in integer;
            tiempo : in integer;
            fin : out std_logic
        );
    END COMPONENT;
    signal enable : integer := 0; -- Sirve para saber en que estado me encuentro
    constant tiempo_tempo : integer := 3; -- Tiempo que permanece encendido el LED
    signal fin_temporizacion : std_logic := '0';

    -- VARIABLES NECESARIAS PARA EL DISPLAY DE 7 SEGMENTOS
    COMPONENT DECODER is
        generic (displays : positive; -- Numero total de displays que hay
                 segmentos : positive -- Segmentos que tiene cada display
               );
        port (
            clk : in std_logic;
            estado_actual : IN integer;
            displays_encendidos : out std_logic_vector(DISPLAYS - 1 downto 0);
            segmentos_encendidos : out std_logic_vector(SEGMENTOS - 1 downto 0)
        );
    END COMPONENT;

    -- VARIABLES NECESARIAS PARA LA MÁQUINA DE ESTADOS
    type state is( S0, S10, S20, S30, S40, S50, S60, S70, S80, S90, S100, S_mas);
    signal estado_actual, siguiente_estado : state := S0;
begin
    Inst_temporizador : TEMPORIZADOR
        port map(
            clk => CLK,
            enable => enable,
            tiempo => tiempo_tempo,
            fin => fin_temporizacion
        );
    Inst_decoder : DECODER
        generic map (DISPLAYS,SEGMENTOS)
        port map(
            clk => CLK,
            estado_actual => enable,
            displays_encendidos => DISPLAYS_ENCENDIDOS,
            segmentos_encendidos => SEGMENTOS_ENCENDIDOS
        );
    -- PROCESO PARA REGISTRO DE ESTADOS
    process(RESET,CLK)
    begin
        if rising_edge(clk) then

```

```

if RESET = '0' then
    estado_actual <= S0;
else
    estado_actual <= siguiente_estado;
case estado_actual is
    when S0 => enable <= 0;
    when S10 => enable <= 1;
    when S20 => enable <= 2;
    when S30 => enable <= 3;
    when S40 => enable <= 4;
    when S50 => enable <= 5;
    when S60 => enable <= 6;
    when S70 => enable <= 7;
    when S80 => enable <= 8;
    when S90 => enable <= 9;
    when S100 => enable <= 10;
    when others => enable <= 11;
end case;
end if;
end if;
end process;

-- PROCESO PARA CAMBIOS DE ESTADOS
cambios_estado : process(MONEDA, estado_actual, fin_temporizacion)
begin
    siguiente_estado <= estado_actual;
    if PRODUCTO /= "0000" then -- solo podemos cambiar de estado si hemos elegido producto
        case estado_actual is
            when S0 =>
                if MONEDA(0) = '1' then
                    siguiente_estado <= S10;
                elsif MONEDA(1) = '1' then
                    siguiente_estado <= S20;
                elsif MONEDA(2) = '1' then
                    siguiente_estado <= S50;
                elsif MONEDA(3) = '1' then
                    siguiente_estado <= S100;
                end if;
            when S10 =>
                if MONEDA(0) = '1' then
                    siguiente_estado <= S20;
                elsif MONEDA(1) = '1' then
                    siguiente_estado <= S30;
                elsif MONEDA(2) = '1' then
                    siguiente_estado <= S60;
                elsif MONEDA(3) = '1' then
                    siguiente_estado <= S_mas;
                end if;
            when S20 =>
                if MONEDA(0) = '1' then
                    siguiente_estado <= S30;
                elsif MONEDA(1) = '1' then
                    siguiente_estado <= S40;
                elsif MONEDA(2) = '1' then
                    siguiente_estado <= S70;
                elsif MONEDA(3) = '1' then
                    siguiente_estado <= S_mas;
                end if;
            when S30 =>
                if MONEDA(0) = '1' then
                    siguiente_estado <= S40;
                elsif MONEDA(1) = '1' then
                    siguiente_estado <= S50;
                elsif MONEDA(2) = '1' then
                    siguiente_estado <= S80;
                elsif MONEDA(3) = '1' then
                    siguiente_estado <= S_mas;
                end if;
            when S40 =>
                if MONEDA(0) = '1' then
                    siguiente_estado <= S50;
                elsif MONEDA(1) = '1' then
                    siguiente_estado <= S60;
                elsif MONEDA(2) = '1' then
                    siguiente_estado <= S90;
                end if;
        end case;
    end if;
end process;

```

```

        elsif MONEDA(3) = '1' then
            siguiente_estado <= S_mas;
        end if;
    when S50 =>
        if MONEDA(0) = '1' then
            siguiente_estado <= S60;
        elsif MONEDA(1) = '1' then
            siguiente_estado <= S70;
        elsif MONEDA(2) = '1' then
            siguiente_estado <= S100;
        elsif MONEDA(3) = '1' then
            siguiente_estado <= S_mas;
        end if;
    when S60 =>
        if MONEDA(0) = '1' then
            siguiente_estado <= S70;
        elsif MONEDA(1) = '1' then
            siguiente_estado <= S80;
        elsif MONEDA(2) = '1' or MONEDA(3) = '1' then
            siguiente_estado <= S_mas;
        end if;
    when S70 =>
        if MONEDA(0) = '1' then
            siguiente_estado <= S80;
        elsif MONEDA(1) = '1' then
            siguiente_estado <= S90;
        elsif MONEDA(2) = '1' or MONEDA(3) = '1' then
            siguiente_estado <= S_mas;
        end if;
    when S80 =>
        if MONEDA(0) = '1' then
            siguiente_estado <= S90;
        elsif MONEDA(1) = '1' then
            siguiente_estado <= S100;
        elsif MONEDA(2) = '1' or MONEDA(3) = '1' then
            siguiente_estado <= S_mas;
        end if;
    when S90 =>
        if MONEDA(0) = '1' then
            siguiente_estado <= S100;
        elsif MONEDA(1) = '1' or MONEDA(2) = '1' or MONEDA(3) = '1' then
            siguiente_estado <= S_mas;
        end if;
    when S100 =>
        if fin_temporizacion = '1' then
            siguiente_estado <= S0;
        end if;
    when others =>
        if fin_temporizacion = '1' then
            siguiente_estado <= S0;
        end if;
    end case;
end if;
end process;

-- PROCESO PARA SALIDAS EN FUNCION DEL ESTADO
salidas : process(estado_actual)
begin
    case estado_actual is
        when S100 =>
            if PRODUCTO = "0001" then
                LEDS <= "00001";
            elsif PRODUCTO = "0010" then
                LEDS <= "00010";
            elsif PRODUCTO = "0100" then
                LEDS <= "00100";
            elsif PRODUCTO = "1000" then
                LEDS <= "01000";
            end if;
        when S_mas =>
            LEDS <= "10000";
    end case;
end process;

```

```
when others =>
    LEDS <= (others => '0');
end case;
end process;
end Behavioral;
```

4.2 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)

Para comprobar el funcionamiento de la máquina de estados, no hemos realizado un testbench específico. Su correcto comportamiento será comprobado cuando expliquemos la entidad top.

5 DECODIFICADOR

Para que el usuario pueda conocer cuánto dinero ha introducido hasta el momento en la máquina, vamos a hacer uso de los displays de 7 segmentos.

5.1 REPRESENTACIÓN DE NÚMEROS EN EL DISPLAY DE 7 SEGMENTOS

	a	f	b							
a	1	0	1	0	0	1	0	0	0	0
b	0	1	0	0	0	0	1	1	0	0
c	0	0	1	0	0	0	0	0	0	0
d	0	1	0	0	1	0	0	1	0	0
e	0	1	0	1	1	1	0	1	0	1
f	0	1	1	1	0	0	0	1	0	0
g	1	1	0	0	0	0	0	1	0	0

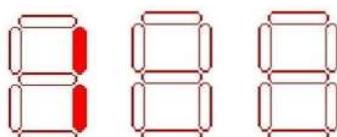
5.2 REPRESENTACIÓN DEL DINERO USANDO LOS DISPLAYS DE 7 SEGMENTOS

Anteriormente hemos indicado como representar cada uno de los números usando el display de 7 segmentos. Ahora queremos representar 3 números usando 3 displays de segmentos diferentes.

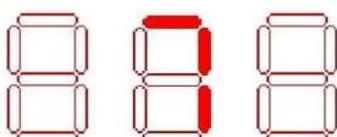
Tal y como está conectada nuestra placa, todos los displays comparten todos los segmentos. Es decir, el número representado en aquellos displays encendidos al mismo tiempo es el obligatoriamente el mismo. Para solucionar este problema, nos ayudaremos de la señal de reloj. Programaremos los displays de forma que, cada 3 ciclos de reloj, cada display solo se encienda en 1 de ellos.

Imaginemos que queremos representar el número 175. Procederemos de la siguiente forma:

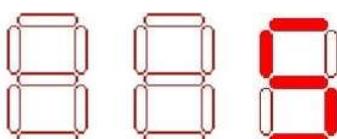
Primer ciclo de reloj



Segundo ciclo de reloj



Tercer ciclo de reloj



5.3 CÓDIGO VHDL

En cada ciclo de reloj, esta entidad nos va a devolver 2 cosas: el display que se enciende y el número que debe representar. Su código es:

```
library ieee;
use ieee.std_logic_1164.ALL;

entity DECODER is
    GENERIC (DISPLAYS : positive; -- Numero total de displays que hay
              SEGMENTOS : positive -- Segmentos que tiene cada display
            );
    PORT (
        CLK : in std_logic;
        ESTADO_ACTUAL : in integer;
        DISPLAYS_ENCENDIDOS : out std_logic_vector(DISPLAYS - 1 downto 0);
        SEGMENTOS_ENCENDIDOS : out std_logic_vector(SEGMENTOS - 1 downto 0)
    );
end entity DECODER;

architecture dataflow of DECODER is
    signal reloj_auxiliar : std_logic := '0';
begin
    begin
        process(CLK)
            subtype ciclos is integer range 0 to 10**8;
            variable contaje : ciclos;
        begin
            if rising_edge(CLK) then
                contaje := contaje + 1;
                if contaje = 10**5 - 1 then
                    contaje := 0;
                    reloj_auxiliar <= not reloj_auxiliar;
                end if;
            end if;
        end process;

        process(reloj_auxiliar) -- DETERMINAMOS QUE DISPLAY QUEREMOS ENCENDER EN CADA MOMENTO
            variable display : integer := 0;
            variable euros, centimos_1, centimos_2 : std_logic_vector(SEGMENTOS - 1 downto 0);
        begin
            if rising_edge(reloj_auxiliar) then
                case ESTADO_ACTUAL is
                    when 0 =>
                        euros :=      "00000001"; -- 0
                        centimos_1 := "10000001"; -- 0
                        centimos_2 := "10000001"; -- 0
                    when 1 =>
                        euros :=      "00000001"; -- 0
                        centimos_1 := "11001111"; -- 1
                        centimos_2 := "10000001"; -- 0
                    when 2 =>
                        euros :=      "00000001"; -- 0
                        centimos_1 := "10010010"; -- 2
                        centimos_2 := "10000001"; -- 0
                    when 3 =>
                        euros :=      "00000001"; -- 0
                        centimos_1 := "10000110"; -- 3
                        centimos_2 := "10000001"; -- 0
                    when 4 =>
                        euros :=      "00000001"; -- 0
                        centimos_1 := "11001100"; -- 4
                        centimos_2 := "10000001"; -- 0
                    when 5 =>
                        euros :=      "00000001"; -- 0
                        centimos_1 := "10100100"; -- 5
                        centimos_2 := "10000001"; -- 0
                end case;
            end if;
        end process;
    end;
end;
```

```

when 6 =>
    euros :=      "00000001"; -- 0
    centimos_1 := "10100000"; -- 6
    centimos_2 := "10000001"; -- 0
when 7 =>
    euros :=      "00000001"; -- 0
    centimos_1 := "10001111"; -- 7
    centimos_2 := "10000001"; -- 0
when 8 =>
    euros :=      "00000001"; -- 0
    centimos_1 := "10000000"; -- 8
    centimos_2 := "10000001"; -- 0
when 9 =>
    euros :=      "00000001"; -- 0
    centimos_1 := "10000100"; -- 9
    centimos_2 := "10000001"; -- 0
when 10 =>
    euros :=      "01001111"; -- 1
    centimos_1 := "10000001"; -- 0
    centimos_2 := "10000001"; -- 0
when others =>
    euros :=      "11111110"; --
    centimos_1 := "11111110"; --
    centimos_2 := "11111110"; --
end case;

display := (display + 1) mod 3; -- modificamos el display que vamos a representar
case display is
    when 0 =>
        DISPLAYS_ENCENDIDOS <= "11111011";
        SEGMENTOS_ENCENDIDOS <= euros;
    when 1 =>
        DISPLAYS_ENCENDIDOS <= "11111101";
        SEGMENTOS_ENCENDIDOS <= centimos_1;
    when others =>
        DISPLAYS_ENCENDIDOS <= "11111110";
        SEGMENTOS_ENCENDIDOS <= centimos_2;
    end case;
end if;
end process;
end architecture dataflow;

```

Si bien los displays tienen 7 segmentos, se puede observar como la variable SEGMENTOS_ENCENDIDOS tiene un tamaño de 8. Esto se debe a que, en el dígito de los euros, se ha querido representar también el punto.

5.4 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)

Para comprobar el funcionamiento del decodificador, hemos empleado el siguiente testbench:

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

entity tb_decoder is
end tb_decoder;

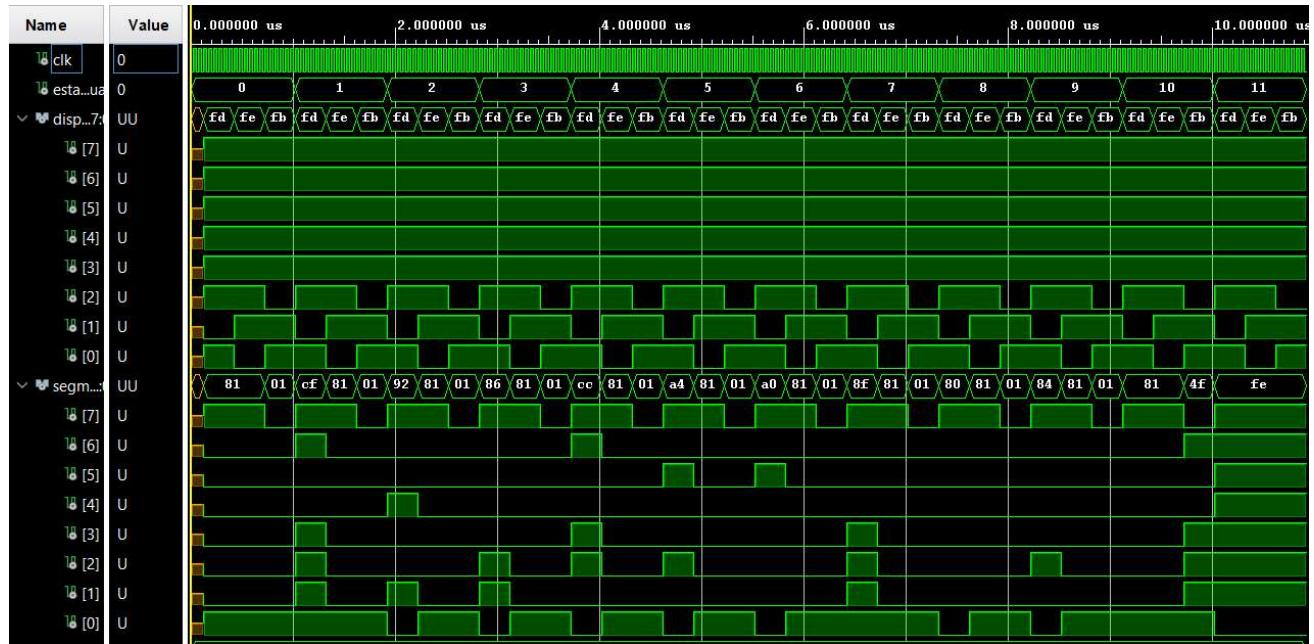
architecture Behavioral of tb_decoder is
COMPONENT DECODER is
    generic (DISPLAYS : positive; -- Numero total de displays que hay
             SEGMENTOS : positive -- Segmentos que tiene cada display
            );
    port (
        CLK : in std_logic;
        ESTADO_ACTUAL : in integer;
        DISPLAYS_ENCENDIDOS : out std_logic_vector(DISPLAYS - 1 downto 0);
        SEGMENTOS_ENCENDIDOS : out std_logic_vector(SEGMENTOS - 1 downto 0)
    );
END COMPONENT;
constant displays : positive := 8;
constant segmentos : positive := 8;
signal clk : std_logic;
signal estado_actual : integer := 0;
signal displays_encendidos : std_logic_vector(displays - 1 downto 0);
signal segmentos_encendidos : std_logic_vector(segmentos - 1 downto 0);
begin
process
begin
    clk <= '0';
    wait for 25 ns;
    clk <= '1';
    wait for 25 ns;
end process;

Inst_decoder : DECODER
generic map (DISPLAYS,SEGMENTOS)
port map(
    CLK => clk,
    ESTADO_ACTUAL => estado_actual,
    DISPLAYS_ENCENDIDOS => displays_encendidos,
    SEGMENTOS_ENCENDIDOS => segmentos_encendidos
);

process
begin
    wait for 125 ns;
    for i in 1 to 11 loop
        wait for 900 ns;
        estado_actual <= i;
    end loop;
    wait for 900ns;
    assert false
        report "Simulation finished."
        severity failure;
end process;
end Behavioral;
```

Para poder ver correctamente el testbench, hemos modificado el código original para que la señal de reloj auxiliar cambie cada 3 ciclos de reloj

Como resultado de la simulación obtuvimos:



Se puede observar como, de forma periódica y rotativa, se van modificando los displays a encender y la cantidad representada en cada momento. Además, la cantidad a representar en cada instante coincide tanto con el display encendido como con el estado en el que se encuentra.

6 TEMPORIZADOR

Como hemos dicho anteriormente, los cambios de estado de S10/S11 a S0 son por tiempo. Es decir, va a permanecer un tiempo en dichos estados (en nuestro caso 3 segundos) antes de cambiar de estado.

Para poder contar tiempo, nos basaremos en la señal de reloj. Teniendo en cuenta que nuestra placa tiene una frecuencia de 100 MHz, su periodo es de 10^{-8} segundos. Esto significa que, para que transcurra un segundo, tenemos que contar 10^8 flancos positivos de reloj. Contando 10^8 flancos tantas veces como segundos queremos que transcurran, obtendremos nuestro objetivo.

6.1 CÓDIGO VHDL

El código empleado es:

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

entity TEMPORIZADOR is
    PORT ( CLK : in std_logic;
            ENABLE : in integer; -- Solo quiero empezar a contar cuando esté en S100 o S_mas
            TIEMPO : in integer; -- Tiempo a contar en segundos
            FIN : out std_logic
        );
end TEMPORIZADOR;

architecture Behavioral of TEMPORIZADOR is
    signal salida_auxiliar : std_logic;
begin
    process(CLK)
        subtype ciclos is integer range 0 to 10**8;
        variable contaje : ciclos;
        variable segundos : integer;
    begin
        if rising_edge(CLK) then
            if ENABLE = 10 or ENABLE = 11 then
                contaje := contaje + 1;
                if contaje = 10**8 - 1 then
                    contaje := 0;
                    segundos := segundos + 1;
                end if;
                if segundos = TIEMPO then
                    salida_auxiliar <= '1';
                end if;
            else
                contaje := 0;
                segundos := 0;
                salida_auxiliar <= '0';
            end if;
        end if;
    end process;
    FIN <= salida_auxiliar; -- salida definitiva de la entidad
end Behavioral;
```

6.2 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)

Para comprobar el funcionamiento de la entidad top, hemos empleado el siguiente testbench:

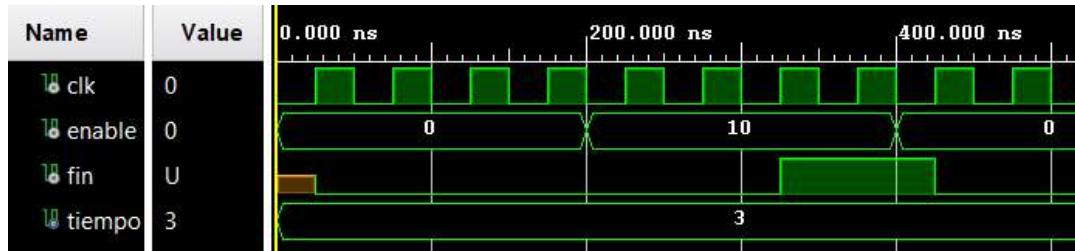
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_tempo is
end tb_tempo;

architecture Behavioral of tb_tempo is
COMPONENT TEMPORIZADOR is
    port (
        CLK : in std_logic;
        ENABLE : in integer;
        TIEMPO : in integer;
        FIN : out std_logic
    );
END COMPONENT;
constant tiempo : integer := 3;
signal clk : std_logic;
signal enable: integer;
signal fin : std_logic;
begin
    process
    begin
        clk <= '0';
        wait for 25ns;
        clk <= '1';
        wait for 25ns;
    end process;

    Inst_sincronizador : TEMPORIZADOR
    port map(
        CLK => clk,
        ENABLE => enable,
        TIEMPO => tiempo,
        FIN => fin
    );
    process
    begin
        ENABLE <= 0;
        wait for 200 ns;
        ENABLE <= 10;
        wait for 200 ns;
        ENABLE <= 0;
        wait for 200 ns;
        assert false
            report "Simulation finished."
            severity failure;
    end process;
end Behavioral;
```

Como resultado de la simulación obtuvimos:



Para mejorar la visibilidad en la simulación, hemos cambiado el tiempo para que solo cuente 3 ciclos

Como se puede observar, una vez se habilita el temporizador (lo que representa que nos encontramos en uno de los 2 últimos estados del sistema), la señal del temporizador se activa transcurrido el tiempo indicado. Al desactivarse dicha señal (lo que representa que hemos vuelto al estado final), la salida vuelve a 0.

7 ENTIDAD TOP

Se trata de la entidad central del proyecto. Desde ella, llamaremos a las entidades SYNCHRNZR y EDGEDTCTR en señales de un pulso (para evitar rebotes y problemas de metaestabilidad), así como a la entidad ESTADOS, para controlar la evolución del sistema.

7.1 CÓDIGO VHDL

El código de la entidad top es:

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

entity top is
    GENERIC (WITDH : positive := 4; -- Tenemos 4 tipos diferentes de monedas
             DISPLAYS : positive := 8; -- Existen 8 displays en la placa
             SEGMENTOS : positive := 8 -- Cada display tiene 7 segmentos y el punto
            );
    PORT ( CLK : in std_logic; -- Señal de reloj
           RESET : in std_logic; -- Señal de reset
           MONEDA : in std_logic_vector(WITDH - 1 downto 0); -- Entradas de las monedas
           PRODUCTO : in std_logic_vector(WITDH - 1 downto 0); -- Entradas de productos
           LEDS : out std_logic_vector(WITDH downto 0); -- Leds que indican que se ha terminado de introducir la cantidad
           DISPLAYS_ENCENDIDOS : out std_logic_vector(DISPLAYS - 1 downto 0); -- Sirve para encender los displays
           SEGMENTOS_ENCENDIDOS : out std_logic_vector(SEGMENTOS - 1 downto 0) -- Sirve para encender los segmentos de los displays
          );
end top;

architecture Behavioral of top is
    -- VARIABLES NECESARIAS PARA EL SINCRONIZADOR
    COMPONENT SYNCHRNZR is
        generic (withub : positive);
        port (
            clk : in std_logic;
            async_in : in std_logic_vector(withub - 1 downto 0);
            sync_out : out std_logic_vector(withub - 1 downto 0)
        );
    END COMPONENT;
    signal sincro_moneda : std_logic_vector(WITDH - 1 downto 0); -- salida del sincronizador

    -- VARIABLES NECESARIAS PARA EL GENERADOR DE PULSOS
    COMPONENT EDGEDTCTR is
        generic (withub : positive);
        port (
            clk : in std_logic;
            sync_in : in std_logic_vector(withub - 1 downto 0);
            edge : out std_logic_vector(withub - 1 downto 0)
        );
    END COMPONENT;
    signal pulso_moneda : std_logic_vector(WITDH - 1 downto 0); -- salida del generador de pulsos

    -- VARIABLES NECESARIAS PARA LA MÁQUINA DE ESTADOS
    COMPONENT ESTADOS is
        generic (withub : positive;
                 displays : positive;
                 segmentos : positive
                );
        port ( clk : in std_logic;
               reset : in std_logic;
               moneda : in std_logic_vector(withub - 1 downto 0);
               producto : in std_logic_vector(withub - 1 downto 0);
               leds : out std_logic_vector(withub downto 0);
               displays_encendidos : out std_logic_vector(displays - 1 downto 0);
               segmentos_encendidos : out std_logic_vector(segmentos - 1 downto 0)
              );
    END COMPONENT;
begin
    begin
        -- INSTANCIAS DE LOS COMPONENTES NECESARIOS
        Inst_sincronizador : SYNCHRNZR
            generic map (WITDH)
            port map(
                clk => CLK,
                async_in => MONEDA,
                sync_out => sincro_moneda
            );
    end;
```

```
Inst_generador_pulsos : EDGEDTCTR
    generic map (WITDH)
    port map(
        clk => CLK,
        sync_in => sincro_moneda,
        edge => pulso_moneda
    );
Inst_maquina_estados : ESTADOS
    generic map (WITDH, DISPLAYS, SEGMENTOS)
    port map (
        clk => CLK,
        reset => RESET,
        moneda => pulso_moneda,
        producto => PRODUCTO,
        leds => LEDS,
        displays_encendidos => DISPLAYS_ENCENDIDOS,
        segmentos_encendidos => SEGMENTOS_ENCENDIDOS
    );
end Behavioral;
```

7.2 COMPROBACIÓN DEL FUNCIONAMIENTO (TESTBENCH)

Para comprobar el funcionamiento de la entidad, partiremos del siguiente código, al cual le iremos realizando algunas modificaciones:

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

entity tb_top is
end tb_top;

architecture Behavioral of tb_top is
    -- VARIABLES NECESARIAS PARA EL DISPLAY DE 7 SEGMENTOS
    COMPONENT TOP is
        GENERIC (WITDH : positive := 4; -- Tenemos 4 tipos diferentes de monedas
                 DISPLAYS : positive := 8; -- Vamos a encender 3 displays
                 SEGMENTOS : positive := 8 -- Cada display tiene 7 segmentos
            );
        PORT ( CLK : in std_logic; -- Señal de reloj
                RESET : in std_logic; -- Señal de reset
                MONEDA : in std_logic_vector(WITDH - 1 downto 0); -- Entradas de las monedas
                PRODUCTO : in std_logic_vector(WITDH - 1 downto 0); -- Entradas de productos
                LEDS : out std_logic_vector(WITDH downto 0); -- Leds que indican que se ha terminado de introducir la cantidad
                DISPLAYS_ENCENDIDOS : out std_logic_vector(DISPLAYS - 1 downto 0); -- Sirve para encender los displays
                SEGMENTOS_ENCENDIDOS : out std_logic_vector(SEGMENTOS - 1 downto 0) -- Sirve para encender los segmentos de los displays
            );
    END COMPONENT;
    constant WITDH : positive := 4;
    constant DISPLAYS : positive := 8;
    constant SEGMENTOS : positive := 8;
    signal clk,reset : std_logic;
    signal moneda,producto : std_logic_vector(WITDH - 1 downto 0);
    signal leds : std_logic_vector(WITDH downto 0);
    signal displays_encendidos : std_logic_vector(DISPLAYS - 1 downto 0);
    signal segmentos_encendidos : std_logic_vector(SEGMENTOS - 1 downto 0);
begin
    process
    begin
        clk <= '0';
        wait for 25 ns;
        clk <= '1';
        wait for 25 ns;
    end process;

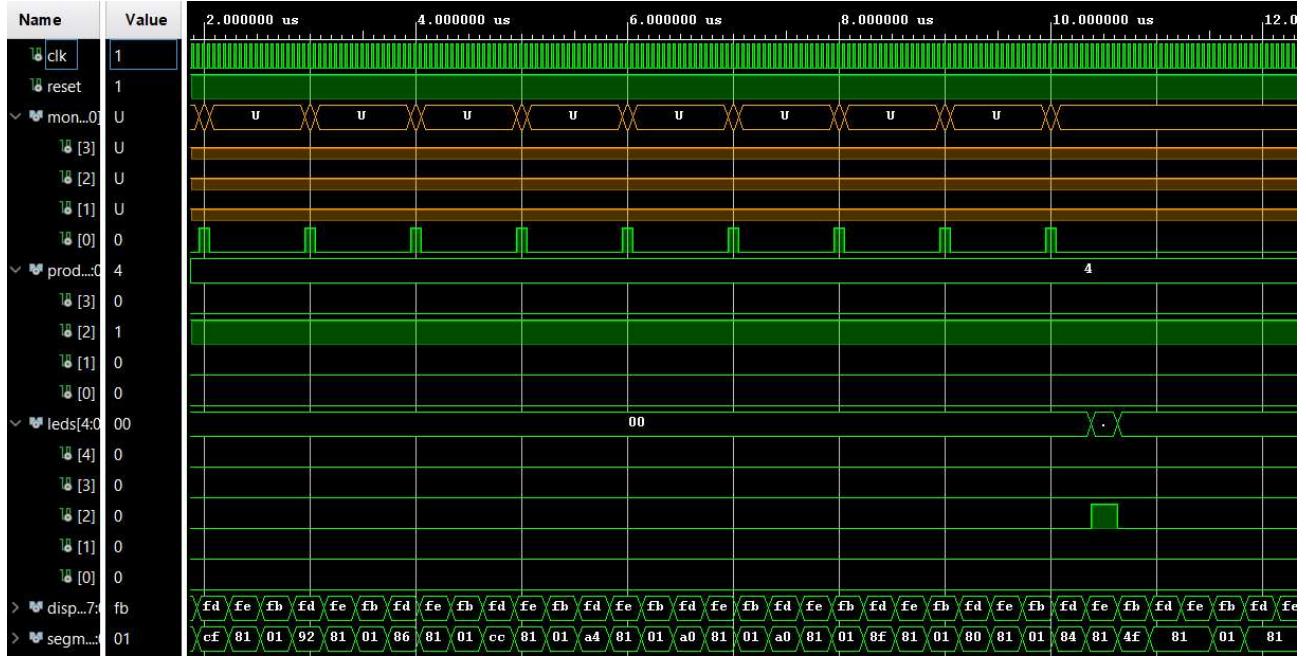
    Inst_maquina_refrescos : TOP
        generic map (WITDH,DISPLAYS,SEGMENTOS)
        port map(
            CLK => clk,
            RESET => reset,
            MONEDA => moneda,
            PRODUCTO => producto,
            LEDS => leds,
            DISPLAYS_ENCENDIDOS => displays_encendidos,
            SEGMENTOS_ENCENDIDOS => segmentos_encendidos
        );

    process
    begin
        reset <= '0';
        wait for 30 ns;
        reset <= '1';
        producto <= "0100";
        wait for 20 ns;
        for i in 1 to 10 loop
            wait for 900 ns;
            moneda(0) <= '1';
            wait for 100 ns;
            moneda(0) <= '0';
        end loop;
        wait for 4 ms;
        assert false
            report "Simulation finished."
            severity failure;
    end process;
end Behavioral;
```

Al igual que en las entidades Temporizador y Decoder, hemos modificado los valores de contejo de flancos positivos de la señal de reloj para que se puede visualizar correctamente su funcionamiento

Comprobación 1

Simulando el código mostrado, obtenemos:



Como se puede observar, si habilitamos el producto 2 e introducimos 10 monedas de 10 céntimos, al alcanzar el precio del producto (todos los productos valen 1 €) se enciende el led 2 (el correspondiente al producto 2).

Comprobación 2

Modificando el código original para asociar el valor “0000” a producto, obtenemos el siguiente resultado:



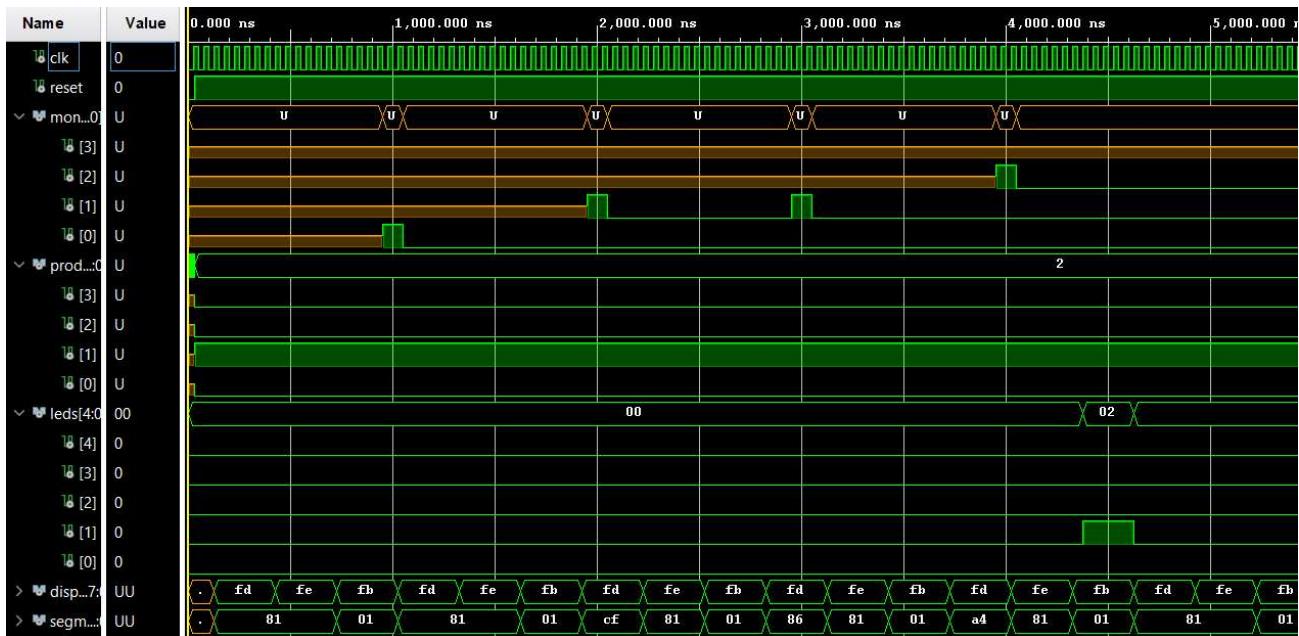
Como se puede observar, en este caso, no se enciende el LED, ya que la máquina de estados no va a evolucionar hasta que hayamos seleccionado 1 de los productos. Además, fijándonos en la variable segmentos_encendidos, no cambia.

Comprobación 3

Si modificamos el código para introducir 1 moneda de 10 céntimos, 2 de 20 y 1 de 50:

```
process
begin
    reset <= '0';
    wait for 30 ns;
    reset <= '1';
    producto <= "0010";
    wait for 920 ns;
    moneda(0) <= '1';
    wait for 100 ns;
    moneda(0) <= '0';
    for i in 1 to 2 loop
        wait for 900 ns;
        moneda(1) <= '1';
        wait for 100 ns;
        moneda(1) <= '0';
    end loop;
    wait for 900 ns;
    moneda(2) <= '1';
    wait for 100 ns;
    moneda(2) <= '0';
    wait for 4 ms;
    assert false
        report "Simulation finished."
        severity failure;
    end process;
end Behavioral;
```

Obtenemos el siguiente resultado:



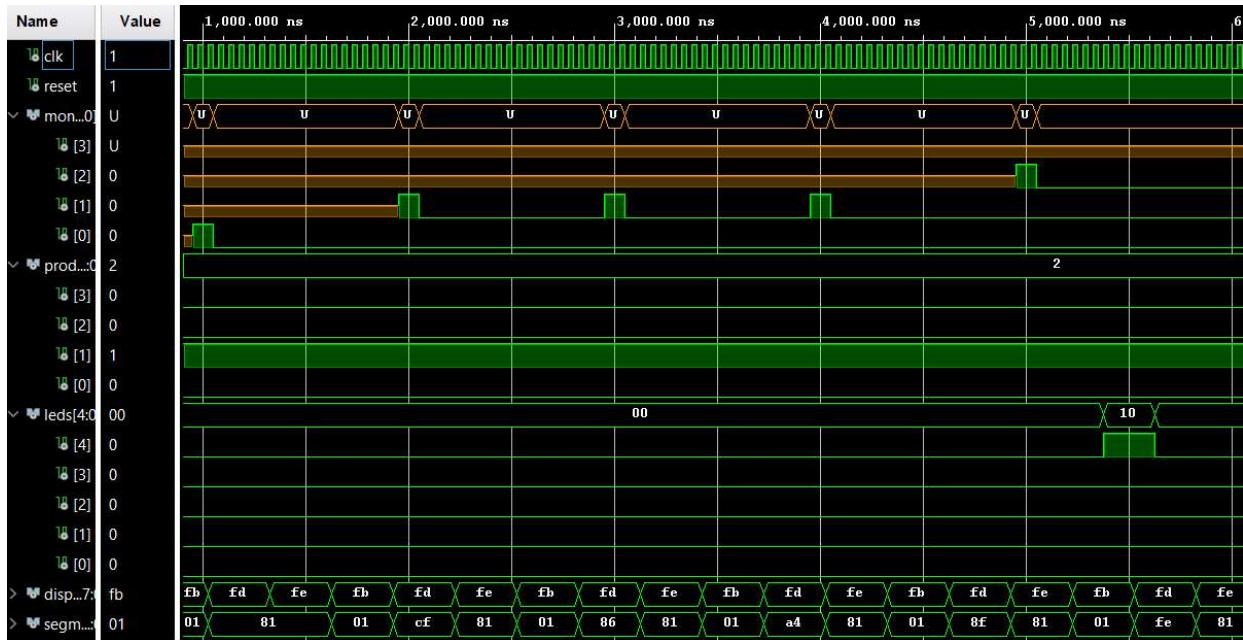
Como se puede observar, vuelve a iluminarse el LED correspondiente al producto. Queda comprobado que podemos introducir cualquier secuencia de monedas con tal de llegar a la cifra exacta de 1 euro.

Comprobación 4

Si modificamos el código para introducir 1 moneda de 10 céntimos, 3 de 20 y 1 de 50:

```
process
begin
    reset <= '0';
    wait for 30 ns;
    reset <= '1';
    producto <= "0010";
    wait for 920 ns;
    moneda(0) <= '1';
    wait for 100 ns;
    moneda(0) <= '0';
    for i in 1 to 3 loop
        wait for 900 ns;
        moneda(1) <= '1';
        wait for 100 ns;
        moneda(1) <= '0';
    end loop;
    wait for 900 ns;
    moneda(2) <= '1';
    wait for 100 ns;
    moneda(2) <= '0';
    wait for 4 ms;
    assert false
        report "Simulation finished."
        severity failure;
    end process;
end Behavioral;
```

Obtenemos el siguiente resultado:



En este caso, hemos introducido más dinero del que deberíamos. Por ello, en lugar en encenderse el LED correspondiente al producto, se enciende el LED 4, que simbolizado el “estado de error”.

8 MAPA DE RESTRICCIONES DE LA PLACA

Una vez diseñado el código, y comprobado su funcionamiento mediante los testbenches, vamos a introducir el código en la placa. Tenemos que modificar el mapa de restricciones:

```
## Clock signal
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports CLK]

##Switches

set_property -dict {PACKAGE_PIN J15 IOSTANDARD LVCMOS33} [get_ports {PRODUCTO[0]}]
set_property -dict {PACKAGE_PIN L16 IOSTANDARD LVCMOS33} [get_ports {PRODUCTO[1]}]
set_property -dict {PACKAGE_PIN M13 IOSTANDARD LVCMOS33} [get_ports {PRODUCTO[2]}]
set_property -dict {PACKAGE_PIN R15 IOSTANDARD LVCMOS33} [get_ports {PRODUCTO[3]}]

## LEDs

set_property -dict {PACKAGE_PIN H17 IOSTANDARD LVCMOS33} [get_ports {LEDS[0]}]
set_property -dict {PACKAGE_PIN K15 IOSTANDARD LVCMOS33} [get_ports {LEDS[1]}]
set_property -dict {PACKAGE_PIN J13 IOSTANDARD LVCMOS33} [get_ports {LEDS[2]}]
set_property -dict {PACKAGE_PIN N14 IOSTANDARD LVCMOS33} [get_ports {LEDS[3]}]
set_property -dict {PACKAGE_PIN R18 IOSTANDARD LVCMOS33} [get_ports {LEDS[4]}]

##7 segment display

set_property -dict {PACKAGE_PIN T10 IOSTANDARD LVCMOS33} [get_ports {SEGMENTOS_ENCENDIDOS[6]}]
set_property -dict {PACKAGE_PIN R10 IOSTANDARD LVCMOS33} [get_ports {SEGMENTOS_ENCENDIDOS[5]}]
set_property -dict {PACKAGE_PIN K16 IOSTANDARD LVCMOS33} [get_ports {SEGMENTOS_ENCENDIDOS[4]}]
set_property -dict {PACKAGE_PIN K13 IOSTANDARD LVCMOS33} [get_ports {SEGMENTOS_ENCENDIDOS[3]}]
set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMOS33} [get_ports {SEGMENTOS_ENCENDIDOS[2]}]
set_property -dict {PACKAGE_PIN T11 IOSTANDARD LVCMOS33} [get_ports {SEGMENTOS_ENCENDIDOS[1]}]
set_property -dict {PACKAGE_PIN L18 IOSTANDARD LVCMOS33} [get_ports {SEGMENTOS_ENCENDIDOS[0]}]

set_property -dict { PACKAGE_PIN H15    IOSTANDARD LVCMOS33 } [get_ports { SEGMENTOS_ENCENDIDOS[7] }];

set_property -dict {PACKAGE_PIN J17 IOSTANDARD LVCMOS33} [get_ports {DISPLAYS_ENCENDIDOS[0]}]
set_property -dict {PACKAGE_PIN J18 IOSTANDARD LVCMOS33} [get_ports {DISPLAYS_ENCENDIDOS[1]}]
set_property -dict {PACKAGE_PIN T9  IOSTANDARD LVCMOS33} [get_ports {DISPLAYS_ENCENDIDOS[2]}]
set_property -dict {PACKAGE_PIN J14 IOSTANDARD LVCMOS33} [get_ports {DISPLAYS_ENCENDIDOS[3]}]
set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMOS33} [get_ports {DISPLAYS_ENCENDIDOS[4]}]
set_property -dict {PACKAGE_PIN T14 IOSTANDARD LVCMOS33} [get_ports {DISPLAYS_ENCENDIDOS[5]}]
set_property -dict {PACKAGE_PIN K2  IOSTANDARD LVCMOS33} [get_ports {DISPLAYS_ENCENDIDOS[6]}]
set_property -dict {PACKAGE_PIN U13 IOSTANDARD LVCMOS33} [get_ports {DISPLAYS_ENCENDIDOS[7]}]

##Buttons

set_property -dict {PACKAGE_PIN C12 IOSTANDARD LVCMOS33} [get_ports RESET]
set_property -dict {PACKAGE_PIN M18 IOSTANDARD LVCMOS33} [get_ports {MONEDA[0]}]
set_property -dict {PACKAGE_PIN P17 IOSTANDARD LVCMOS33} [get_ports {MONEDA[1]}]
set_property -dict {PACKAGE_PIN M17 IOSTANDARD LVCMOS33} [get_ports {MONEDA[3]}]
set_property -dict { PACKAGE_PIN P18    IOSTANDARD LVCMOS33 } [get_ports { MONEDA[2] }]
```

9 COMPROBACIÓN DEL FUNCIONAMIENTO EN LA PLACA

Finalmente, insertamos algunas imágenes para corroborar el funcionamiento del sistema:

