

# SISTEMAS ELECTRÓNICOS DIGITALES

---

## Trabajo

VHDL: Máquina expendedora de refrescos

---

INTEGRANTES DEL GRUPO	
Apellidos, Nombre	Matrícula
Kamenov Iliev, Simeon	55309
López Montero, Jorge	55322
Navarro Bados, Cristina	55380

Grupo Teoría	A403 y EE403
Grupo Trabajo	17
Profesor	Rubén Núñez

**FECHA DE ENTREGA: 09/01/2023**

# ÍNDICE

<b>1. Descripción de la funcionalidad.....</b>	<b>1</b>
<b>2. Entidades: atribución de responsabilidades .....</b>	<b>1</b>
2.1. Diagrama de bloques.....	2
<b>3. Entidad SYNCHRNZR.....</b>	<b>6</b>
<b>4. Entidad DEBOUNCER .....</b>	<b>6</b>
<b>5. Entidad EDGEDTCTR .....</b>	<b>8</b>
<b>6. Entidad COIN_INPUT .....</b>	<b>9</b>
<b>7. Entidad FSM_SELECTION .....</b>	<b>9</b>
<b>8. Entidad COUNTER.....</b>	<b>11</b>
<b>9. Entidad FSM_PRICE .....</b>	<b>13</b>
<b>10. Entidad FSM_DISPLAY .....</b>	<b>16</b>
Controlador .....	16
Decodificador.....	18
<b>11. Entidad TOP .....</b>	<b>23</b>
<b>12. Simulaciones.....</b>	<b>24</b>
COUNTER_tb.....	24
FSM_SELECTION_tb .....	29
FSM_PRICE_tb .....	31

## 1. Descripción de la funcionalidad

Se ha implementado la funcionalidad de una máquina expendedora empleando el entorno de programación hardware VIVADO. Siguiendo el documento correspondiente a las ofertas de trabajos, tenemos los siguientes requisitos mínimos:

- La máquina debe admitir monedas de 10c, 20c, 50c y 1€
- El precio de todos los productos es de 1€
- La máquina detecta como error si se introduce más dinero al importe exacto

Basándonos en los requisitos mínimos que debe cumplir el sistema, hemos modificado algunos apartados de la anterior lista con el fin de aumentar los casos de uso de la máquina. Estas modificaciones son:

- El precio de las bebidas puede variar. Se ha escogido arbitrariamente 1€, 1.2€ , 1.5€ y 1.8€ para el precio de las bebidas.
- La venta del producto se producirá tanto si se excede el importe exacto como si no, de tal manera que se indicará por pantalla el cambio en caso de excederse el precio.
- Se mostrará en el display de 7 segmentos el precio de cada producto, así como la cantidad de dinero introducida en cada momento.
- Una vez realizada la venta, la máquina entrará en un estado de standby para que se limpien los interruptores de selección de bebida. Para iniciar un nuevo proceso de compra es necesario un rearme manual mediante un interruptor.

En este documento presentaremos cada una de las entidades que conforman el proyecto y su jerarquía, es decir cómo se relacionan. Junto a la descripción funcional de cada entidad se incluirá el código VHDL correspondiente, o en algunos casos solamente fragmentos debido a la extensión de la entidad. No obstante, el código completo del proyecto está disponible en el repositorio de GitHub.

Por último, mencionar que el proyecto se ha cargado en la placa NEXYS 4 DDR.

## 2. Entidades: atribución de responsabilidades

Primeramente, concretaremos la jerarquía de las entidades y a continuación, iremos mostrándolas de menor a mayor relevancia, terminando con la entidad TOP que engloba a todas las anteriores.

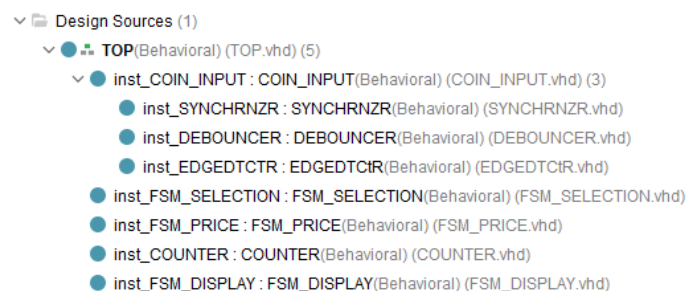


Imagen 1: Pestaña Sources

La funcionalidad se puede dividir en los siguientes bloques: Detección de introducción de moneda; Detección de selección de bebida; Registro del dinero introducido, así como el cálculo del posible cambio; Registrar una venta y poner al sistema en standby; y Mostrar la información por el display.

La detección de moneda la lleva a cabo la entidad **COIN\_INPUT**, que se auxilia de tres entidades más con finalidad de tratar la señales y evitar falsas detecciones. Estas entidades son: **SYNCHRNZR**, **DEBOUNCER** y **EDGEDTCTR**, y se encargan de sincronizar la señal, evitar los rebotes y detectar el flanco.

La detección de la bebida se comprueba con la entidad **FSM\_SELECTION**. Esta habilita una señal cuando se selecciona una bebida y permite cambiar la elección. Cuando la máquina se encuentra en standby no se puede seleccionar ningún producto hasta que se produce el rearme manual.

Por otro lado, la cantidad de dinero introducida se calcula con la entidad **COUNTER**, la cual devuelve la cuenta (en binario) y el cambio (como número natural). Solo se realiza la cuenta de dinero una vez se ha seleccionado una bebida, de modo que si se introduce una moneda antes de seleccionar la bebida se produce la devolución de la misma. Cuando se registra una venta se resetea la cuenta, pero no el cambio, el cual lo hace una vez realizado el rearme manual.

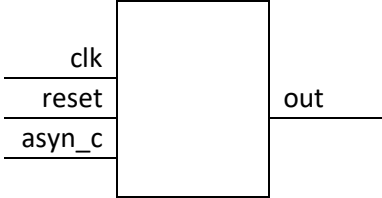
La entidad **COUNTER** trabaja conjuntamente con la entidad **FSM\_PRICE**, que detecta la venta de un producto si el dinero introducido es de mayor o igual importe al precio de la bebida. **FSM\_PRICE** habilita una señal al producirse la venta y vuelve al estado inicial tras el rearme.

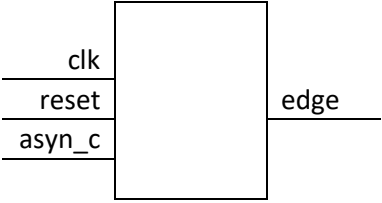
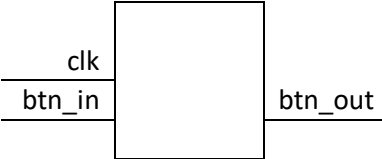
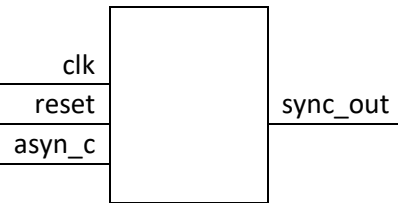
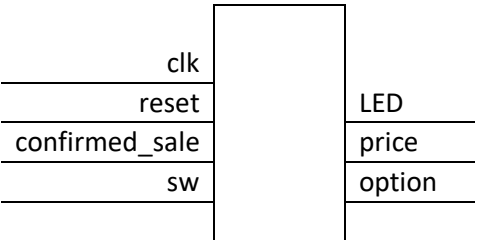
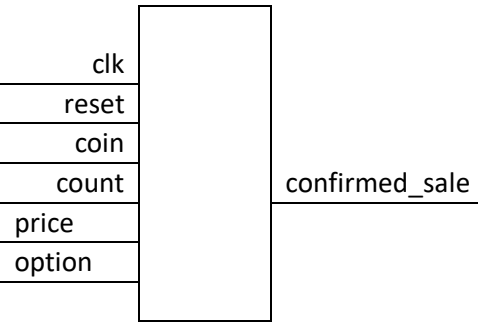
Por último, la entidad **FSM\_DISPLAY** controla los segmentos del display. Esta entidad está compuesta por tres estados, donde el primero corresponde a la pantalla de inicio que indica la necesidad de seleccionar bebida; el segundo muestra, a la izquierda, el precio de la bebida seleccionada y , a la derecha, la cantidad de dinero introducida; y el tercer estado corresponde al estado de standby, mostrándonos el cambio e informándonos del reseteo manual.

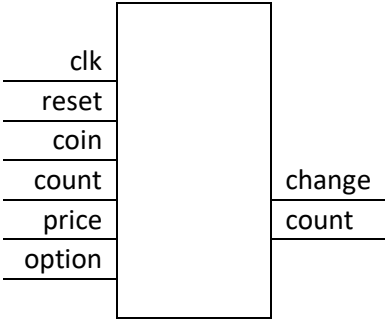
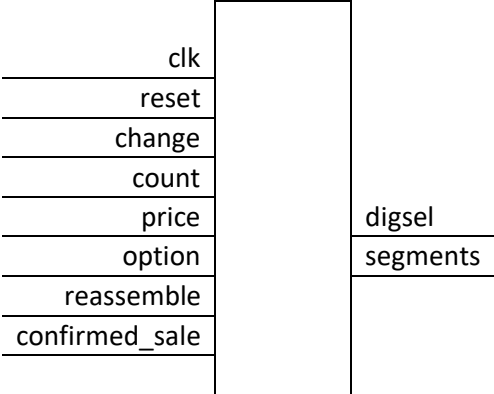
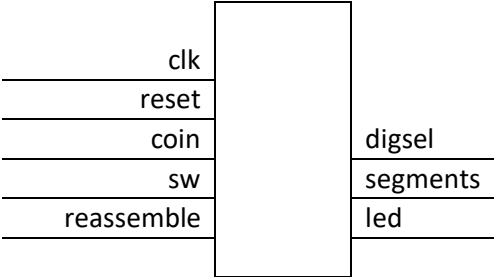
A continuación, se mostrará el diagrama de bloques del sistema y una descripción más detallada de cada entidad junto a su código.

### 2.1. Diagrama de bloques

En la siguiente tabla se muestra la interfaz de cada entidad:

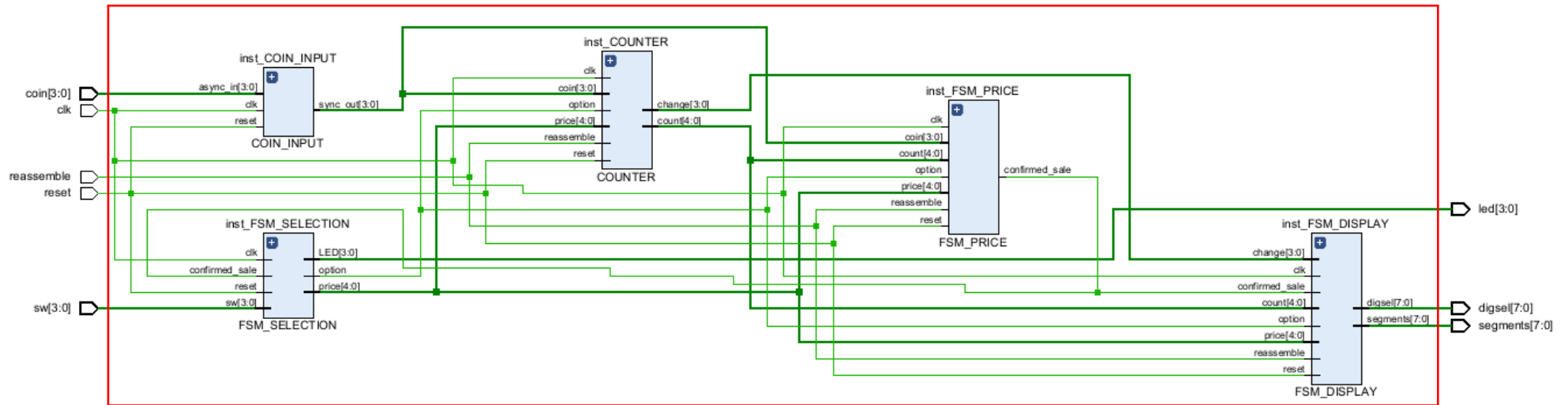
ENTIDAD	DIAGRAMA DE BLOQUES
<b>SYNCHRNZR</b> clk: in reset: in async_in: in[3:0] syn_out: out[3:0]	 <pre>graph LR     clk[clk] --&gt; block     reset[reset] --&gt; block     async_c[async_c] --&gt; block     block --&gt; out[out]</pre>

<p align="center"><u>EDGETCTR</u></p> <p>clk: in  reset: in  sync_in: in[3:0]  edge: out[3:0]</p>	
<p align="center"><u>DEBOUNCER</u></p> <p>clk: in  btn_in_in: in[3:0]  btn_out: out[3:0]</p>	
<p align="center"><u>COIN_INPUT</u></p> <p>clk: in  reset: in  asyn_in: in[3:0]  sync_out: out[3:0]</p>	
<p align="center"><u>FSM_SELECTION</u></p> <p>clk: in  reset: in  confirmed_sale: in  sw: in[3:0]  LED: out[3:0]  price: out[4:0]  option: out</p>	
<p align="center"><u>FSM_PRICE</u></p> <p>clk: in  reset: in  coin: in[3:0]  count: in[4:0]  price: in[4:0]  option: in  reassemble: in  confirmed_sale: out</p>	

<p align="center"><u>COUNTER</u></p> <p>clk: in  reset: in  coin: in[3:0]  price: in[4:0]  option: in  reassemble: in  change: out[3:0]  count: out[4:0]</p>	
<p align="center"><u>FSM_DISPLAY</u></p> <p>clk: in  reset: in  change: in[3:0]  price: in[4:0]  count: in[4:0]  reassemble: in  option: in  confirmed_sale: in  digsel: out[7:0]  segments: out[7:0]</p>	
<p align="center"><u>TOP</u></p> <p>clk: in  reset: in  coin: in[3:0]  sw: in[3:0]  reassemble: in  digsel: out[7:0]  segments: out[7:0]  led: out[3:0]</p>	

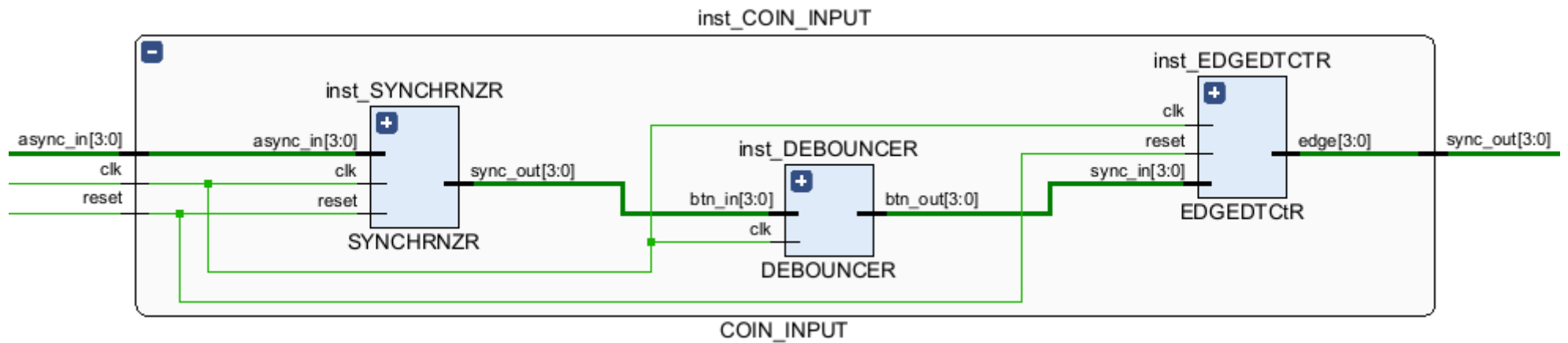
Como se puede observar de la tabla, la entidad TOP no es la entidad con el mayor número de componentes en su interfaz. Esto es así porque se han definido señales intermedias para almacenar el diferentes valores o detectar eventos. En la siguiente página aparece el diagrama esquemático del conjunto proporcionado por el entorno de VIVADO, donde se aprecian las entradas y salidas de cada entidad y las relaciones entre estas.

ENTIDAD TOP



TOP

ENTIDAD COIN\_INPUT



### 3. Entidad SYNCHRNZR

La entidad SYNCHRNZR se encarga de devolver una señal en sincronismo con el reloj del sistema para evitar posibles cambios en la señal de la entrada durante la lectura. Dicha tarea se consigue con dos memorias que guardan los valores intermedios (ya estables), por lo que la salida siempre se retrasará dos ciclos de reloj.

Respecto al código, se ha empleado el sincronizador dado en los guiones de prácticas de la asignatura, no obstante, se ha modificado levemente para que sea capaz de operar con mas de una señal simultáneamente. El código de la entidad es el siguiente:

```
entity SYNCHRNZR is
  Port (
    clk: in std_logic; -- clock
    reset: in std_logic; -- Asynchronous reset
    async_in: in std_logic_vector(3 downto 0); -- Asynchronous input
    sync_out: out std_logic_vector(3 downto 0) -- Synchronous output
  );
end SYNCHRNZR;

architecture Behavioral of SYNCHRNZR is
  -- DECLARACION DE SEÑALES
  type matrix_sreg is array(3 downto 0) of std_logic_vector(1 downto 0);
  signal sreg: matrix_sreg;

begin
  process (reset, clk)
  begin
    if (reset='0') then
      for i in 0 to 3 loop
        sync_out(i) <= '0';
        sreg(i) <= "00";
      end loop;
    elsif rising_edge(clk) then
      for i in 0 to 3 loop
        sync_out(i) <= sreg(i)(1);
        sreg(i) <= sreg(i)(0) & async_in(i);
      end loop;
    end if;
  end process;
end architecture Behavioral;
```

Se ha definido una señal de tipo matricial para poder operar con todas las señales simultáneamente empleando un bucle “for” en el process. Se podría generalizar el código para cualquier número de entradas estableciendo un genérico en la definición de la entidad.

### 4. Entidad DEBOUNCER

Una vez se tiene una señal de entrada síncrona, se pasará dicha señal por un debouncer para filtrar la señal de posibles rebotes pues usamos botones como entrada. El código referente al debouncer es:



```

use IEEE.numeric_std.ALL;

entity DEBOUNCER is
    generic(
        -- Ancho del vector de conteo
        WIDTH : positive:= 9
    );
    port (
        clk      : in std_logic;  -- clock
        btn_in   : in std_logic_vector(3 downto 0); -- Synchronous input from SYNCHRNZR
        btn_out  : out std_logic_vector(3 downto 0) -- Stable value output
    );
end entity DEBOUNCER;

architecture Behavioral of DEBOUNCER is

    -- DECLARACION E INICIALIZACION DE SEÑALES
    -- btn_prev sirve para comprobar que la señal sea estable
    signal btn_prev: std_logic_vector(3 downto 0);
    -- counter almacena los ciclos en los que la señal no ha cambiado de valor
    type matrix_counter is array (3 downto 0) of unsigned(WIDTH downto 0);
    signal counter      : matrix_counter := ( (others => '0') ,
                                              (others => '0') ,
                                              (others => '0') ,
                                              (others => '0') );

begin
    process(clk)
    begin
        if (clk'event and clk='1') then
            for i in 0 to 3 loop
                if (btn_prev(i) xor btn_in(i)) = '1' then
                    counter(i) <= (others => '0');
                    btn_prev(i) <= btn_in(i);
                elsif (counter(i)(WIDTH) = '0') then
                    counter(i) <= counter(i) + 1;
                else
                    btn_out(i) <= btn_prev(i);
                end if;
            end loop;
        end if;
    end process;
end architecture Behavioral;

```

Como condición para evitar los rebotes, se establece un número mínimo de veces en las que la señal debe de presentar el mismo valor para considerarla estable. La cuenta de las veces que se repite la señal se guarda en la señal “counter” y en caso de no coincidir el valor en dos ciclos de reloj consecutivos se vuelve a empezar la cuenta.

El tamaño de la cuenta la variamos con el genérico WIDTH, el cual lo hemos dejado con un valor de 9 pues hemos comprobado en la práctica que no se producen rebotes. El tiempo que necesita la señal para considerarse estable se puede obtener con la frecuencia de reloj y con el tamaño de la variable “counter”. Se tiene:

$$t = \frac{2^{WIDTH}}{f_{clk}} \quad (1)$$

Introduciendo datos:

$$t = \frac{2^9}{100[MHz]} = 5.1[\mu s]$$

Si fuese necesario aumentar el tiempo de detección únicamente modificamos el valor de WIDTH.

## 5. Entidad EDGEDTCTR

Como último paso en el tratamiento de la señal de entrada, tenemos la entidad EDGEDTCTR que detecta los flancos de subida. Como en las anteriores entidades, se ha definido una señal matricial para tratar todos los botones simultáneamente:

```
entity EDGEDTCTR is
  Port (
    clk: in std_logic; -- clock
    reset: in std_logic; -- Asynchronous reset
    sync_in: in std_logic_vector(3 downto 0); -- Input from DEBOUNCER
    edge: out std_logic_vector(3 downto 0)
  );
end EDGEDTCTR;

architecture Behavioral of EDGEDTCTR is

  -- DECLARACION DE SEÑALES
  type matrix_sreg is array(3 downto 0) of std_logic_vector(2 downto 0);
  signal sreg : matrix_sreg;
  signal edge_s: std_logic_vector(3 downto 0) := "0000";

begin
  -- DETECCION DE FLANCO
  process (reset, clk)
  begin
    if (reset = '0') then
      for i in 0 to 3 loop
        sreg(i) <= "000";
      end loop;
    elsif rising_edge(clk) then
      for i in 0 to 3 loop
        sreg(i) <= sreg(i)(1 downto 0) & sync_in(i);
      end loop;
    end if;
  end process;

  -- ACTUALIZACION DEL VALOR DE EDGE_S
  process (reset, sreg)
  begin
    if (reset = '0') then
      edge_s <= "0000";
    else
      for i in 0 to 3 loop
        if (sreg(i) = "100") then
          edge_s(i) <= '1';
        else
          edge_s(i) <= '0';
        end if;
      end loop;
    end if;
  end process;

  -- ASIGNACION DE LA SEÑAL AL PUERTO
  edge <= edge_s;
end Behavioral;
```

## 6. Entidad COIN\_INPUT

La entidad COIN\_INPUT representa la interfaz que engloba las tres entidades anteriores, con la finalidad de presentar un código más compacto y funcional. A continuación, se mostrará un fragmento del código, pues la arquitectura solo se compone de la declaración de los diferentes componentes y de su instanciación.

```
entity COIN_INPUT is
    -- se podría generalizar para mas monedas con generic(WIDTH(...))
    port (
        clk: in std_logic; -- clock
        reset: in std_logic; -- Asynchronous reset
        async_in: in std_logic_vector(3 downto 0); -- Asynchronous coin input
        sync_out: out std_logic_vector(3 downto 0) -- Synchronous coin output
    );
end COIN_INPUT;

architecture Behavioral of COIN_INPUT is

    -- DECLARACION DE SEÑALES
    signal sync_output: std_logic_vector(3 downto 0);
    signal btn_out: std_logic_vector(3 downto 0);

    -- DECLARACION DE COMPONENTES
    --SYNCHRNZR nos devuelve una señal sincrona
    component SYNCHRNZR is
        Port (
            clk: in std_logic;
            reset: in std_logic;
            async_in: in std_logic_vector(3 downto 0);
            sync_out: out std_logic_vector(3 downto 0)
        );
    end component SYNCHRNZR;
    --Otras declaraciones

begin

    -- INICIALIZACION DE COMPONENTES
    inst_SYNCHRNZR: SYNCHRNZR port map(
        clk => clk ,
        reset => reset ,
        async_in => async_in ,
        sync_out => sync_output
    );
    --Otras inicializaciones

end architecture Behavioral;
```

## 7. Entidad FSM\_SELECTION

Como se ha adelantado anteriormente, FSM\_SELECTION permite detectar si se ha seleccionado un producto mediante 4 interruptores diferentes, donde cada interruptor representa una bebida. FSM\_SELECTION está estrechamente relacionada con FSM\_PRICE puesto que su habilitación depende de la salida “confirmed\_sale” de esta segunda entidad. Esta condición se visualiza en el process “next\_state\_decoder”:

```

entity FSM_SELECTION is
  Port (
    -- INPUTS
    clk: in std_logic; -- clk
    reset: in std_logic; -- Asynchronous reset
    sw: in std_logic_vector(3 downto 0); -- switch input: 0=J15, 1=L16, 2=M13, 3=T18
    confirmed_sale: in std_logic; -- 1 if an item is sold ("external reset")
    -- OUTPUTS
    LED: out std_logic_vector(3 downto 0); -- Shows the selection
    option: out std_logic; -- 1 if any product is selected
    price: out natural range 0 to 20 -- Drink price
  );
end entity FSM_SELECTION;

architecture Behavioral of FSM_SELECTION is

  --DECLARACION DE SEÑALES
  -- definimos las bebidas de las que disponemos
  type BEBIDA is (S0, agua, fanta, nestea, redbull); -- S0 estado de reposo
  -- creamos las señales para la maquina de estados
  signal present_state: BEBIDA := S0;
  signal next_state: BEBIDA;

begin

  state_register: process (clk, reset)
  begin
    if (reset = '0') then
      present_state <= S0;
    elsif (rising_edge(clk)) then
      present_state <= next_state;
    end if;
  end process;

  next_state_decoder: process(present_state, sw, confirmed_sale)
  begin
    next_state <= present_state;
    -- Tras la venta de una bebida FSM_PRICE envia la informacion
    -- Se necesita un rearme manual para desactivar confirmed_sale
    if (confirmed_sale = '1') then
      -- S0=1 -> option=0 (FSM_SELECTION.vdh)-> count="00000" (COUNTER.vhd)
      -- De esta forma se realiza una limpieza de las variables
      next_state <= S0;
    else
      case present_state is
        when S0 =>
          if (sw(0) = '1') then
            next_state <= agua;
          elsif (sw(1) = '1') then
            next_state <= fanta;
          elsif (sw(2) = '1') then
            next_state <= nestea;
          elsif (sw(3) = '1') then
            next_state <= redbull;
          end if;
        when agua | fanta | nestea | redbull=>
          if (sw = "0000") then
            next_state <= S0;
          end if;
        when others =>
          next_state <= S0;
      end case;
    end if;
  end process;

  output_decoder: process(present_state)

```

```

begin
  case present state is
    when S0 =>
      option <= '0';
      price <= 0;
      LED <= "0000";
    when agua =>
      option <= '1';
      price <= 10;
      LED(0) <= '1';
    when fanta =>
      option <= '1';
      price <= 12;
      LED(1) <= '1';
    when nestea =>
      option <= '1';
      price <= 15;
      LED(2) <= '1';
    when redbull =>
      option <= '1';
      price <= 18;
      LED(3) <= '1';
    when others =>
      option <= '0';
      price <= 0;
      LED <= "0000";
    end case;
  end process;
end Behavioral;

```

Mientras no se detecte que la venta se ha realizado se puede volver al estado inicial y elegir otra bebida a la anterior seleccionada. En este caso, se supone que el dinero introducido (de haberlo hecho) se devolvería al usuario pues se produce la limpieza del contador como se verá más adelante en la entidad **COUNTER**.

Mencionar, que a cada bebida se le ha adjudicado un correspondiente led para verificar visualmente nuestra selección, y además un precio en formato decimal (cada precio está multiplicado por 10 para evitar usar decimales).

## 8. Entidad COUNTER

La entidad COUNTER va actualizando el almacenador del dinero introducido, al mismo tiempo que calcula el cambio en caso de excederse el precio fijado para cada bebida. La señal “count” se transmite en binario, mientras que “change” en formato decimal (pese a que internamente se realice la conversión a binario como se aprecia en el diagrama de bloques). Estos formatos son una cuestión de comodidad a la hora de realizar comparaciones, no obstante, la conversión entre uno y otro no representa un mayor inconveniente como se verá en posteriores entidades.

Dicho esto, el código es el siguiente:

```

entity COUNTER is
  Port (
    --INPUTS
    clk: in std_logic; -- clock
    reset: in std_logic; --Asynchronous reset
    coin: in std_logic_vector(3 downto 0); -- Coins input

```

```

    option: in std_logic; -- 1 if any product is selected
    price: in natural range 0 to 20; -- Drink price
    reassemble: in std_logic; -- Manual restart
    --OUTPUTS
    count: out unsigned(4 downto 0); -- Money count
    change: out natural range 0 to 10 -- Money change
);
end COUNTER;

architecture Behavioral of COUNTER is

    -- DECLARACION DE SEÑALES
    -- usamos count_s para poder leer el valor acumulado
    signal count_s: unsigned(4 downto 0) := "00000";

begin

    coin_sum: process(clk, reset)
    variable cuenta: unsigned(4 downto 0) := "00000";
    begin
        if (reset = '0') then
            cuenta := "00000";
        elsif (option = '0') then
            cuenta := "00000";
        elsif rising_edge(clk) then
            if(coin(0) = '1') then
                cuenta := cuenta + "00001";
            elsif(coin(1) = '1') then
                cuenta := cuenta + "00010";
            elsif(coin(2) = '1') then
                cuenta := cuenta + "00101";
            elsif(coin(3) = '1') then
                cuenta := cuenta + "01010";
            end if;
        end if;
        count_s <= cuenta;
    end process;

    money_change: process(reset , clk, reassemble)
    variable count_s_N: natural range 0 to 30;
    variable cambio: natural range 0 to 10 := 0;
    begin
        count_s_N := to_integer(count_s);
        if (reset = '0') then
            cambio := 0;
        elsif(count_s_N > price) then
            cambio := count_s_N - price;
        elsif(reassemble = '0' and cambio /= 0) then
            cambio := cambio;
        else
            cambio := 0;
        end if;
        change <= cambio;
    end process;

    -- ASIGNACION DE PUERTOS
    count <= count_s;

end Behavioral;

```

Implementamos 2 process diferentes, uno para la señal “count” y otro para “change”, en ambos hemos definido variables con la finalidad de acumular el valor y evitar crear latches innecesarios, así como posibles bucles en los que se realimenta la salida.

La variable “count” se limpia continuamente hasta que se selecciona una bebida, esto se observa con la condición *option = '0'*. Por otra parte, change únicamente se limpia con el rearme manual puesto que queremos leer el valor que contiene y mostrarlo en la pantalla de standby.

### 9. Entidad FSM\_PRICE

FSM\_PRICE trabaja conjuntamente con la entidad COUNTER pues recibe la cuenta del dinero con lo cual debe determinar si se ha producido la venta del producto seleccionado. FSM\_PRICE se ha implementado como una máquina de estados, aunque, en retrospectiva, no hacía falta puesto que la señal de salida “confirmed\_sale” se puede obtener con la simple comparación del precio y del contador.

Pese a la innecesaria complejidad de la entidad, esta resulta funcional por lo que presentamos el trabajo con ella. La entidad se compone de un total de 20 estados, siendo S0 el estado de reposo (sin introducir dinero); S1, S2, ..., S18 los estados que representan la cantidad de dinero introducida; y S19 el estado de venta segura (tanto por exceso de la cuenta o valor justo).

No se mostrará todo el código puesto que las condiciones de franqueo de estados se repiten constantemente y no aporta información. Como ejemplo de la evolución de los estados se mostrará el siguiente ejemplo:

- 1) Seleccionamos un producto cuyo valor es de 1.5€ (price = 15).
- 2) Se habilita la máquina de estados y se empieza en el estado S0.
- 3) Tenemos la posibilidad de introducir monedas de 10c, 20c, 50c y 1€.
  - a. Si introducimos la moneda de 10c pasamos al estado S1.
  - b. Si introducimos la moneda de 20c pasamos al estado S2
  - c. Si introducimos la moneda de 50c pasamos al estado S5
  - d. Si introducimos la moneda de 1€ pasamos al estado S10
- 4) Volvemos a repetir el punto 3) desde el nuevo estado al que hemos llegado.
- 5) El punto 4) será posible siempre y cuando:
  - a. No se haya alcanzado el estado S15 (para este concreto caso).
  - b. No se haya alcanzado un estado entre S15 y S19, siendo la cuenta mayor al precio.
- 6) Si no es posible el punto 4) se salta automáticamente al estado S19, en caso de no estar en él.
- 7) Desde el estado S19 se pasa al estado de reposo S0 y se deshabilita la máquina de estados.
- 8) Realizamos el rearme manual y volvemos al punto 1) de selección de producto.

Dicho esto, los fragmentos más significativos del código son:

```
entity FSM_PRICE is
  Port (
    --INPUTS
    clk: in std_logic; -- clock
    reset: in std_logic; --Asynchronous reset
    coin: in std_logic_vector(3 downto 0); -- Coins input
    option: in std_logic; -- 1 if any product is selected
```

```

    price: in natural range 0 to 20; -- Drink price
    count: in unsigned(4 downto 0); -- Money count
    reassemble: in std_logic; -- Manual restart
    --OUTPUTS
    confirmed_sale: out std_logic -- 1 if an item is sold
);
end entity FSM_PRICE;

architecture Behavioral of FSM_PRICE is
-- DECLARACION DE SEÑALES
-- definimos un estado por cada combinacion de monedas
    type COUNT_STATE is (S0, S1, S2, S3, S4, S5,
                          S6, S7, S8, S9, S10, S11,
                          S12, S13, S14, S15, S16,
                          S17, S18, S19); -- S0 estado de reposo
                                          -- Cada estado representa una combinacion de dinero
                                          -- S19 representa el estado de venta asegurada
                                          -- (se llega bien excediendo el precio o con el importe
justo)
    signal present_state: COUNT_STATE := S0;
    signal next_state: COUNT_STATE;
    -- utilizamos confirmed_sale_s para realizar asignacion dentro del process (next_state_decoder)
    signal confirmed_sale_s: std_logic;

begin

    state_register: process(clk, reset)
    begin
        if (reset = '0') then
            present_state <= S0;
        elsif (rising_edge(clk)) then
            present_state <= next_state;
        end if;
    end process;

    -- Se definen las condiciones de franqueo de cada estado
    -- No tiene complejidad pero es extenso
    -- Los estados en los que se produce la venta son: S10, S12, S15, S18 Y S19
    -- (y todos los intermedios si el contador supera el precio)
    -- Como maximo pueden introducir 2.7[€] pero a partir de 1.9 lo consideramos todo igual
    -- Para estos casos el resto lo calcula la entidad COUNTER y se guarda en change
    next_state_decoder: process(clk, present_state, coin)
    variable count_N: natural range 0 to 30;
    begin
        count_N := to_integer(count);
        -- Solo se aceptaran monedas si se detecta un producto seleccionado
        -- Tras detectar una venta se deshabilita la introudcion de monedas pues option=0
(FSM_SELECTION.vhd)
        if(option = '1') then
            next_state <= present_state;
            case present_state is
                when S0 =>
                    if (coin(0) = '1') then
                        next_state <= S1;
                    elsif (coin(1) = '1') then
                        next_state <= S2;
                    elsif (coin(2) = '1') then
                        next_state <= S5;
                    elsif (coin(3) = '1') then
                        next_state <= S10;
                    end if;

                    -- Más estados sin posible venta

                when S10 =>
                    if (price = count_N) then
                        next_state <= S19;

```



```
        else
            if (coin(0) = '1') then
                next_state <= S11;
            elsif (coin(1) = '1') then
                next_state <= S12;
            elsif (coin(2) = '1') then
                next_state <= S15;
            elsif (coin(3) = '1') then
                next_state <= S19;
            end if;
        end if;

    when S11 =>
        if (price < count_N) then
            next_state <= S19;
        else
            if (coin(0) = '1') then
                next_state <= S12;
            elsif (coin(1) = '1') then
                next_state <= S13;
            elsif (coin(2) = '1') then
                next_state <= S16;
            elsif (coin(3) = '1') then
                next_state <= S19;
            end if;
        end if;

    -- Más estados intermedios con posible venta

    when S18 =>
        next_state <= S19;

    when S19 =>
        next_state <= S0;

    when others =>
        next_state <= S0;
    end case;
end if;
end process;

output_decoder: process(present_state, reassemble)
variable venta: std_logic := '0';
begin
    if (reset = '0') then
        confirmed_sale_s <= '0';
        venta := '0';
    elsif (present_state = S19) then
        confirmed_sale_s <= '1';
        venta := '1';
    elsif (venta = '1' and reassemble = '0') then
        confirmed_sale_s <= '1';
    else
        confirmed_sale_s <= '0';
        venta := '0';
    end if;
end process;

-- ASGNACION DE PUERTOS
confirmed_sale <= confirmed_sale_s;

end architecture Behavioral;
```

Cuando se llega al estado S19 es cuando se detecta la venta, tal y como se ve en las condiciones del process “output\_decoder”. También mencionar el uso de la variable “venta” para evitar latches.

### 10. Entidad FSM\_DISPLAY

La entidad FSM\_DISPLAY realiza la función de controlador y decodificador del display de 7 segmentos de la placa. Primeramente, debemos de gestionar la frecuencia a la que se alimentan las líneas de control de los ánodos de cada dígito y posteriormente podremos realizar la decodificación.

#### Controlador

Para el control de los ánodos de selección reduciremos la frecuencia del reloj del sistema empleando una señal, a la que hemos denominado “counter\_1ms”. Cada vez que counter\_1ms llegue a su valor máximo se reiniciará su valor y se aumentará el valor de la señal “digit\_ctrl”, la cual guarda valores entre 0 y 7 (debido al total de 8 dígitos).

El tamaño de “counter\_1ms” se ha escogido teniendo en cuenta la frecuencia a la que se debe refrescar la imagen para evitar el parpadeo en el display. Viendo la hoja de características se nos informa que para evitar el parpadeo el tiempo entre encendidos de un mismo dígito puede estar comprendido entre 1 a 16 [ms]. Teniendo esto en cuenta, hemos optado por refrescar la imagen cada 8 [ms].

Partiendo de la premisa de los 8 [ms], es fácil obtener el tamaño de “counter\_1ms” si seguimos la siguiente expresión:

$$f_{resultante} = \frac{f_{clk}}{(prescaler + 1)} \quad (2)$$

Siendo:

- *prescaler* : tamaño de la señal counter\_1ms.

Introduciendo y despejando en la ecuación (2) se obtiene un valor de 9999. Por otro lado, también se ha definido un reloj con un periodo de 2 [s], para mostrar alternamente en la pantalla final dos mensajes diferentes. El valor del prescaler para el reloj de frecuencia 0.5 [Hz] es 199999999.

A continuación, se mostrará el código que hace referencia a la interfaz de la entidad, así como los process que corresponden al control de los ánodos, dejando para más adelante el código del decodificador:

```
entity FSM_DISPLAY is
  Port (
    --INPUTS
    clk: in std_logic; -- clock
    reset: in std_logic; --Asynchronous reset
```

```

    reassemble: in std_logic; -- Manual restart
    count: in unsigned(4 downto 0); -- Money count
    price: in natural range 0 to 20; -- Drink price
    change: in natural range 0 to 10; -- Money chang
    option: in std_logic; -- 1 if any product is selected
    confirmed_sale: in std_logic; -- 1 if an item is sold
    --OUTPUTS
    segments: out std_logic_vector(7 downto 0); -- Active digit segments
                                           -- Ver constraints

    digsel: out std_logic_vector(7 downto 0) -- Active digit selection
                                           -- Ver constraints
);
end entity FSM_DISPLAY;

architecture Behavioral of FSM_DISPLAY is

-- DECLARACION DE SEÑALES
-- definimos un estado por cada pantalla a mostrar
type DISPLAY_STATE is (S0, S1, S2); -- S0 elegir producto
                                   -- S1 introducir monedas
                                   -- S2 rearme manual

signal present_state: DISPLAY_STATE := S0;
signal next_state: DISPLAY_STATE;

-- counter_1ms nos permiten modificar la frecuencia del reloj
-- tendremos 100[MHz]/(99999+1) = 1[kHz] que sera a la que cambien los digitos
-- cada digito tardara 1[ms] en cambiar y se volvera a encender dentro de 8[ms]
-- 8[ms] esta dentro del margen de 1-16[ms] proporcionado en el datasheet
-- se asegura de esta manera que no hay parpadeo
signal counter_1ms: natural range 0 to 99999 := 0;
-- counter_2s permite cambiar la pantalla final cada 2 segundos
-- tendremos 100[MHz]/(199999999+1) = 0.5[Hz]
signal counter_2s: natural range 0 to 199999999 := 0;
-- digit_ctrl nos permite alternar el digito que queremos activo
signal digit_ctrl: natural range 0 to 7 := 0;
-- final_ctrl nos permite alterar la pantalla final
signal final_ctrl: natural range 0 to 1 := 0;

begin

-- Generamos una señal que cambia cada milisegundo (digit_ctrl)
reloj_1ms: process(clk)
begin
    if (rising_edge(clk)) then
        counter_1ms <= counter_1ms + 1;
        if (counter_1ms >= 99999) then
            counter_1ms <= 0;
            digit_ctrl <= digit_ctrl + 1;
            if (digit_ctrl > 7) then
                digit_ctrl <= 0;
            end if;
        end if;
    end if;
end process;

-- Generamos una señal que cambia cada 2 segundos (final_ctrl)
reloj_2s: process(clk)

-- Asociamos los anodos de los diodos a la señal de un 1[ms]
digit_control: process(digit_ctrl)
begin
    case (digit_ctrl) is
        when 0 =>
            -- Significa que queremos el primer digito activo (activo a nivel bajo)
            digsel <= "01111111";
        when 1 =>

```

```
        digsel <= "10111111";
    when 2 =>
        digsel <= "11011111";
    when 3 =>
        digsel <= "11101111";
    when 4 =>
        digsel <= "11110111";
    when 5 =>
        digsel <= "11111011";
    when 6 =>
        digsel <= "11111101";
    when 7 =>
        digsel <= "11111110";
    end case;
end process;

-- Control de estado para la maquina de estados
state_register: process(clk, reset)

    next_state_decoder: process(present_state, reassemble)

        -- Establecemos los patrones de los segmentos en funcion del estado
        output_decoder: process(present_state, reset, digit_ctrl, final_ctrl)

end Behavioral;
```

### Decodificador

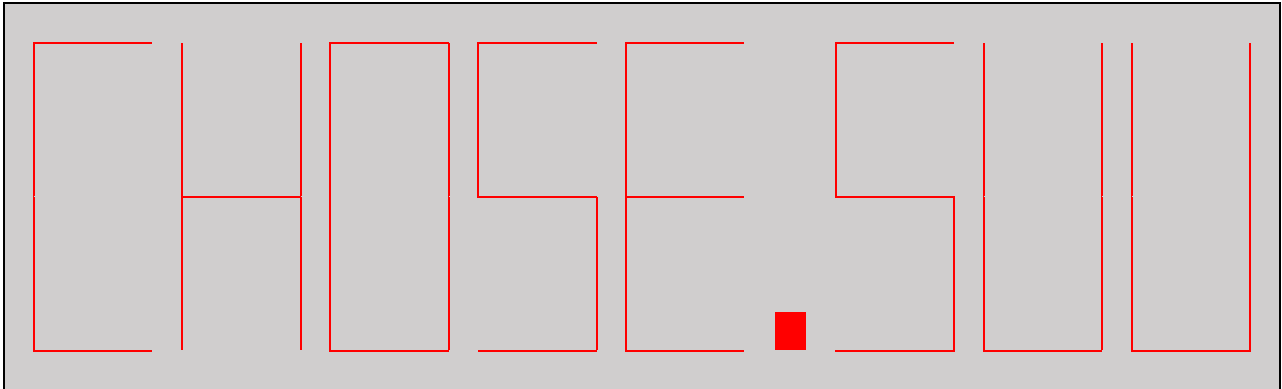
Una vez que tenemos el control de los ánodos de selección, podemos pasar a la máquina de estados que modela el decodificador. Para cada estado mostraremos la salida por pantalla y el código del decodificador. No se mostrará el process “state\_register” al ser igual que en el resto de las máquinas de estado, pero sí se mostrará el process “next\_state\_decoder”:

```
next_state_decoder: process(present_state, reassemble)
    begin
        next_state <= present_state;
        case (present_state) is
            when S0 =>
                if (option = '1') then
                    next_state <= S1;
                end if;
            when S1 =>
                if (option = '0') then
                    next_state <= S0;
                elsif (confirmed_sale = '1') then
                    next_state <= S2;
                end if;
            when S2 =>
                if (reassemble = '1') then
                    next_state <= S0;
                end if;
            when others =>
                next_state <= S0;
        end case;
    end process;
```

Al igual que en FSM\_SELECTION, si se elimina la selección de un producto, se vuelve al estado inicial de reposo.

- Estado S0

Tiene la finalidad de informar al usuario que debe seleccionar un producto. Como el espacio del display es reducido (y no hemos hecho que el texto se desplace), se ha intentado poner mensajes cortos y abreviados. En este estado la salida por pantalla es:



El mensaje es CHOSE.SW, haciendo alusión a los interruptores de selección de bebida. El código para producir dicha salida es el siguiente:

```
output_decoder: process(present_state, reset, digit_ctrl, final_ctrl)
variable count_N: natural range 0 to 30;
begin
    count_N := to_integer(count);
    segments <= "11111111";
    if (reset = '0') then
        segments <= "11111111";
    else
        case (present_state) is
            when S0 =>
                -- Mostraremos un mensaje CHOSE.SW
                if (digit_ctrl = 0) then
                    segments <= "10110001"; -- C
                elsif (digit_ctrl = 1) then
                    segments <= "11001000"; -- H
                elsif (digit_ctrl = 2) then
                    segments <= "10000001"; -- O/O
                elsif (digit_ctrl = 3) then
                    segments <= "10100100"; -- S/5
                elsif (digit_ctrl = 4) then
                    segments <= "00110000"; -- E.
                elsif (digit_ctrl = 5) then
                    segments <= "10100100"; -- S/5

                -- No podemos hacer una W pero podemos concatenar 2 U
                elsif (digit_ctrl = 6) then
                    segments <= "11000001"; -- U
                elsif (digit_ctrl = 7) then
                    segments <= "11000001"; -- U
                end if;

            when S1 => (...)

            when S2 => (...)

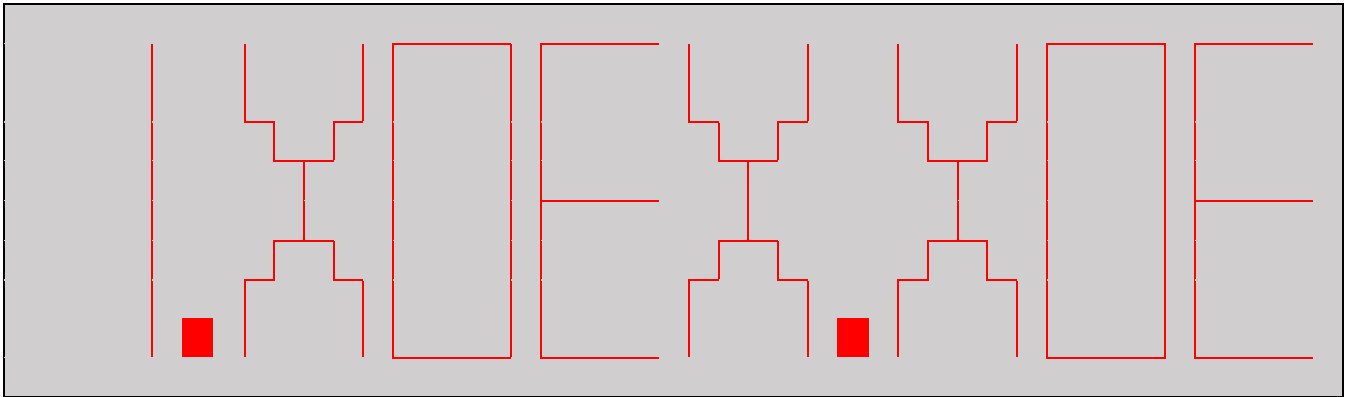
        end case;
    end if;
end process;

end Behavioral
```

Los casos S1 y S2 se comentarán a continuación, cuyo funcionamiento es prácticamente igual al S1 salvo pequeñas diferencias. Para mostrar por pantalla el dígito deseado tenemos que saber qué ánodo se está alimentando en cada momento, para lo cual usamos digit\_ctrl como argumento de la cadena de "if".

- **Estado S1**

Se muestra por pantalla tanto el dinero introducido, a la derecha, como el precio de la bebida, a la izquierda. La salida presenta esta apariencia:



Donde la "X" representa que puede tomar varios valores dependiendo del precio de la bebida y de la cantidad de dinero introducida. El código de S1 es:

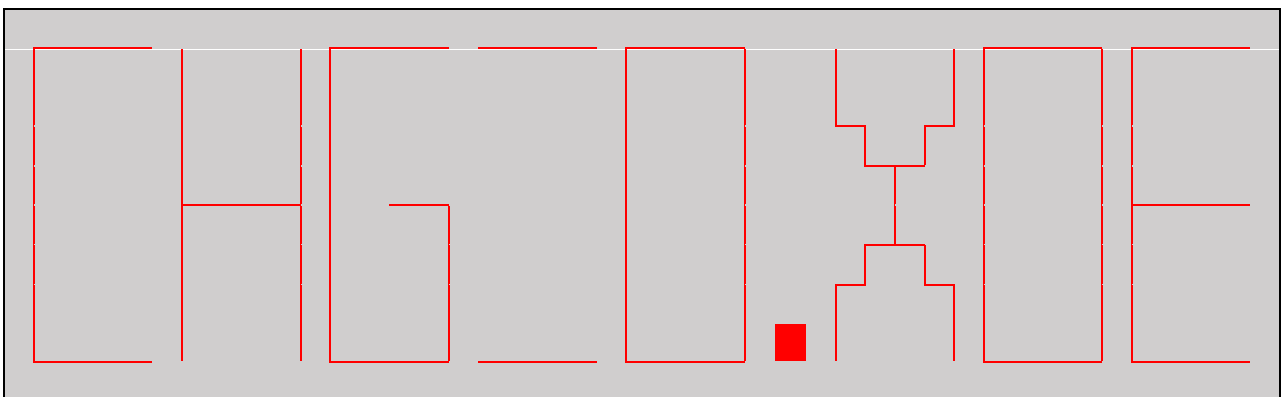
```
when S1 =>
  -- Mostramos el precio a la izquierda y a la derecha las monedas introducidas
  -- 1.X0E X.X0E
  -- 4 DIGITOS DE LA IZQUIERDA
  if (digit_ctrl = 0) then
    segments <= "01001111"; -- 1.
  elsif (digit_ctrl = 1) then
    case (price) is
      when 10 =>
        segments <= "10000001"; -- 0
      when 12 =>
        segments <= "10010010"; -- 2
      when 15 =>
        segments <= "10100100"; -- 5
      when 18 =>
        segments <= "10000000"; -- 8
      when others =>
        segments <= "11111111"; -- No se puede dar el caso
    end case;
  elsif (digit_ctrl = 2) then
    segments <= "10000001"; -- 0
  elsif (digit_ctrl = 3) then
    segments <= "10110000"; -- E
  -- 4 DIGITOS DE LA DERECHA
  elsif (digit_ctrl = 4) then
    if (count_N < 10) then
      segments <= "00000001"; -- 0.
    elsif (count < 20) then
      segments <= "01001111"; -- 1.
    else
      segments <= "00010010"; -- 2.
```

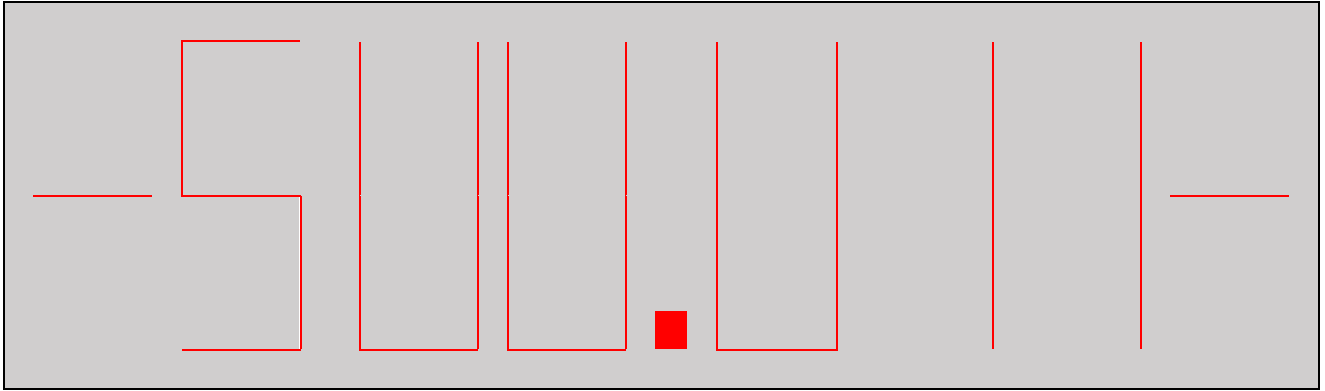
```
end if;
elsif (digit_ctrl = 5) then
  case (count_N mod 10) is
    when 0 =>
      segments <= "10000001"; --0
    when 1 =>
      segments <= "11001111"; --1
    when 2 =>
      segments <= "10010010"; --2
    when 3 =>
      segments <= "10000110"; --3
    when 4 =>
      segments <= "11001100"; --4
    when 5 =>
      segments <= "10100100"; --5
    when 6 =>
      segments <= "10100000"; --6
    when 7 =>
      segments <= "10001111"; --7
    when 8 =>
      segments <= "10000000"; --8
    when 9 =>
      segments <= "10000100"; --9
    when others =>
      segments <= "11111111"; --No debería darse
  end case;
elsif (digit_ctrl = 6) then
  segments <= "10000001"; --0
elsif (digit_ctrl = 7) then
  segments <= "10110000"; --E
end if;
```

Cabe destacar el uso del operador **mod** cuando *digit\_ctrl* = 5, esto nos devuelve el primer valor de los decimales independientemente de la parte entera.

- **Estado S2**

Este estado nos indica que la máquina se encuentra en standby y necesita un rearme manual para volver a comprar un producto. Por pantalla se mostrará el cambio que nos devuelve la máquina y además se alternará cada 2 segundos con un aviso que nos indica el interruptor del rearme manual. Por tanto, las 2 pantallas que se mostrarán son:





La segunda pantalla nos indica que hemos seleccionado el interruptor U11 de la hoja de constraints. El código es:

```
when S2 =>
    -- Mostramos el cambio y cambiamos la pantalla cada 2 segundo para indicar el
    rearme manual
    if (final_ctrl = 0) then
        --Mostraremos CHG=0.X0E
        if (digit_ctrl = 0) then
            segments <= "10110001"; --C
        elsif (digit_ctrl = 1) then
            segments <= "11001000"; --H
        elsif (digit_ctrl = 2) then
            segments <= "10100000"; --6/G
        elsif (digit_ctrl = 3) then
            segments <= "10110111"; --=
        elsif (digit_ctrl = 4) then
            segments <= "00000001"; --0.
        elsif (digit_ctrl = 5) then
            case (change) is
                when 0 =>
                    segments <= "10000001"; --0
                when 1 =>
                    segments <= "11001111"; --1
                when 2 =>
                    segments <= "10010010"; --2
                when 3 =>
                    segments <= "10000110"; --3
                when 4 =>
                    segments <= "11001100"; --4
                when 5 =>
                    segments <= "10100100"; --5
                when 6 =>
                    segments <= "10100000"; --6
                when 7 =>
                    segments <= "10001111"; --7
                when 8 =>
                    segments <= "10000000"; --8
                when 9 =>
                    segments <= "10000100"; --9
                when others =>
                    segments <= "11111111"; -- No puede darse el caso
            end case;
        elsif (digit_ctrl = 6) then
            segments <= "10000001"; --0
        elsif (digit_ctrl = 7) then
            segments <= "10110000"; --E
```



```

        end if;
    elsif (final_ctrl = 1) then
        -- Mostraremos -SW.U11 porque hemos asociado el reseteo a ese interruptor
        if (digit_ctrl = 0) then
            segments <= "11111110"; --
        elsif (digit_ctrl = 1) then
            segments <= "10100100"; -- S/5
        elsif (digit_ctrl = 2) then
            segments <= "11000001"; -- U
        elsif (digit_ctrl = 3) then
            segments <= "01000001"; -- U.
        elsif (digit_ctrl = 4) then
            segments <= "11000001"; -- U
        elsif (digit_ctrl = 5) then
            segments <= "11001111"; -- 1
        elsif (digit_ctrl = 6) then
            segments <= "11001111"; -- 1
        elsif (digit_ctrl = 7) then
            segments <= "11111110"; --
        end if;
    end if;
end if;

```

## 11. Entidad TOP

Por último, la entidad TOP se encarga de instanciar todas las entidades anteriores (a excepción de las instanciadas en COIN\_INPUT), y conectarlas mediante señales. Además, la entidad TOP representa la interfaz física entre la placa y el usuario, pues posee todas las entradas que dependen del usuario.

No se mostrará todo el código, solo un ejemplo la declaración e instanciación del primer componente:

```

entity TOP is
    Port (
        -- INPUTS
        clk: in std_logic; -- Clock (100 MHz)
        reset: in std_logic; -- Asynchronous Reset Signal. TRUE when reset=0
        coin: in std_logic_vector(3 downto 0); -- Coin input: 0=10cent, 1=20cent, 2=50cent,
3=1€
        -- Modelado con botones (ver Constraints)
        sw: in std_logic_vector(3 downto 0); -- switch input: 0=J15, 1=L16, 2=M13, 3=T18
        reassemble: in std_logic; -- Manual restart

        -- OUTPUTS
        segments: out std_logic_vector(7 downto 0); -- Active digit segments
        digsel: out std_logic_vector(7 downto 0); -- Active digit selection
        led: out std_logic_vector(3 downto 0); -- Item selection ligh
    );
end entity TOP;

architecture Behavioral of TOP is

    -- DECLARACION DE SEÑALES
    -- sync_out guarda la pulsacion de los botones y se pasa a las demas entidades
    signal sync_out: std_logic_vector(3 downto 0);
    -- option nos indica si se ha seleccionado una bebida

```

```
signal option: std_logic;
-- price contiene el valor de la bebida multiplicado por 10
signal price: natural range 0 to 20;
-- count contiene el valor almacenado al introducir monedas
signal count: unsigned(4 downto 0);
-- change contiene el valor del cambio
signal change: natural range 0 to 10;
-- confirmed_sale nos informa si se ha producido la venta (count > price)
signal confirmed_sale: std_logic;

-- DECLARACION DE COMPONENTES
-- COIN_INPUT gestiona los botones de introducir moneda
component COIN_INPUT is
  port (
    clk: in std_logic;
    reset: in std_logic;
    async_in: in std_logic_vector(3 downto 0);
    sync_out: out std_logic_vector(3 downto 0)
  );
end component COIN_INPUT;

-- Resto de declaraciones

begin

-- INICIALIZACION DE COMPONENTES
inst_COIN_INPUT: COIN_INPUT port map (
  clk => clk ,
  reset => reset ,
  async_in => coin ,
  sync_out => sync_out
);

-- Resto de instanciaciones

end architecture Behavioral;
```

## 12. Simulaciones

Para asegurarnos del correcto funcionamiento del código antes de cargarlo en la placa, hemos realizado una serie de simulaciones, sobre todo en las entidades que presentan mayor complejidad lógica. Es por eso que no se ha realizado la simulación de COIN\_INPUT ni de FSM\_DISPLAY. No obstante, se han comprobado que en la práctica funcionan correctamente.

Mostraremos el código y la pantalla generada en la simulación para cada entidad en la que se ha llevado a cabo la simulación.

### COUNTER\_tb

Para probar la funcionalidad de la entidad generaremos un tren de pulsos que introduzca monedas de diferentes valores. Además, iremos variando el precio del producto y cuando nos pasemos del precio realizaremos un reseteo para volver al inicio. El código se que se mostrará en esta primera ocasión será el referente a todo el fichero de simulación, pero en los restantes no se mostrarán las partes comunes como la generación del reloj. Dicho esto, el código es:

```

entity COUNTER_tb is
end COUNTER_tb;

architecture Behavioral of COUNTER_tb is

-- DECLARACION DE COMPONENTES
    component COUNTER is
        Port(
            clk: in std_logic;
            reset: in std_logic;
            coin: in std_logic_vector(3 downto 0);
            option: in std_logic;
            price: in natural range 0 to 20;
            reassemble: in std_logic;
            count: out unsigned(4 downto 0);
            change: out natural range 0 to 10
        );
    end component COUNTER;

-- DECLARACION DE CONSTANTES
    constant ClkPeriod : time := 1000 ns;
    constant numero_monedas : positive := 4;
    constant Delay : time := 0.1 * ClkPeriod;

-- DECLARACION DE SEÑALES
    signal clk: std_logic;
    signal reset: std_logic;
    signal i: std_logic;
    signal coin: std_logic_vector(3 downto 0);
    signal option: std_logic;
    signal price: natural range 0 to 20;
    signal reassemble: std_logic;
    signal count: unsigned(4 downto 0);
    signal change: natural range 0 to 10;

begin

-- INSTANCIACION DE COMPONENTES
    inst_COUNTER: COUNTER port map(
        clk => clk ,
        reset => reset ,
        coin => coin ,
        option => option ,
        price => price ,
        reassemble => reassemble ,
        count => count ,
        change => change
    );

-- GENERACION DE RELOJ
    clk_gen : process
    begin
        wait for 0.5 * Clkperiod;
        clk <= '0';
        wait for 0.5 * Clkperiod;
        clk <= '1';
    end process;

-- RESETEO INICIAL Y POSTERIORES
    -- La funcionalidad se testeara a partir del segundo pulso de reloj
    reset <= '0' after 0.25*ClkPeriod,
            '1' after 0.75*ClkPeriod,
            -- Reseteo para limpieza de count (intervalos tomados empiricamente)
            '0' after 8.75*ClkPeriod,
            '1' after 8.75*ClkPeriod + Delay,
            '0' after 17.75*ClkPeriod,
            '1' after 17.75*ClkPeriod + Delay,
            '0' after 26.75*ClkPeriod,

```

```

        '1' after 26.75*ClkPeriod + Delay;

reset_check : process
begin
    -- Se espera hasta que se produzca el reseteo
    wait until reset = '0';
    -- Se concede un periodo para que salte el error
    i <= '0', '1' after Delay;
    if i /= '1' then
        assert count = "0000"
            report "[FAIL]: No se ha producido reseteo inicial"
            severity failure;
    end if;
end process;

-- CONDICIONES INICIALES DE LAS ENTRADAS
-- DETECCION DE BEBIDA SELECCIONADA
-- Resetea count, no hace falta modificar change
    option <= '1' ;
        --'0' after 13.75*ClkPeriod;
-- REARME MANUAL (implementado con un switch)
-- Resetea change, no afecta a count
    reassemble <= '0' ;
        --'1' after 13.75*ClkPeriod;

-- FUNCIONALIDAD

FSM_SELECTION_behaviour : process
variable k: natural range 0 to 3 := 0;
begin
    coin <= "0000";
    wait until reset = '1';
    -- Suma hasta 1[€]
    price <= 10;
    while (count < "01010") loop
        k := k mod numero_monedas;
        coin(k) <= '1';
        wait for ClkPeriod;
        coin(k) <= '0';
        wait for ClkPeriod;
        k := k + 1;
    end loop;
    --Suma hasta 1.2[€]
    wait for ClkPeriod;
    price <= 12;
    while (count < "01100") loop
        k := k mod numero_monedas;
        coin(k) <= '1';
        wait for ClkPeriod;
        coin(k) <= '0';
        wait for ClkPeriod;
        k := k + 1;
    end loop;
    --Suma hasta 1.5[€]
    wait for ClkPeriod;
    price <= 15;
    while (count < "01111") loop
        k := k mod numero_monedas;
        coin(k) <= '1';
        wait for ClkPeriod;
        coin(k) <= '0';
        wait for ClkPeriod;
        k := k + 1;
    end loop;
    --Suma hasta 1.8[€]
    wait for ClkPeriod;
    price <= 18;
    while (count < "10010") loop

```

```
k := k mod numero_monedas;
coin(k) <= '1';
wait for ClkPeriod;
coin(k) <= '0';
wait for ClkPeriod;
k := k + 1;
end loop;
-- Finalizamos la simulacion
assert false
    report "[PASS]: testbench passed."
    severity failure;
end process;

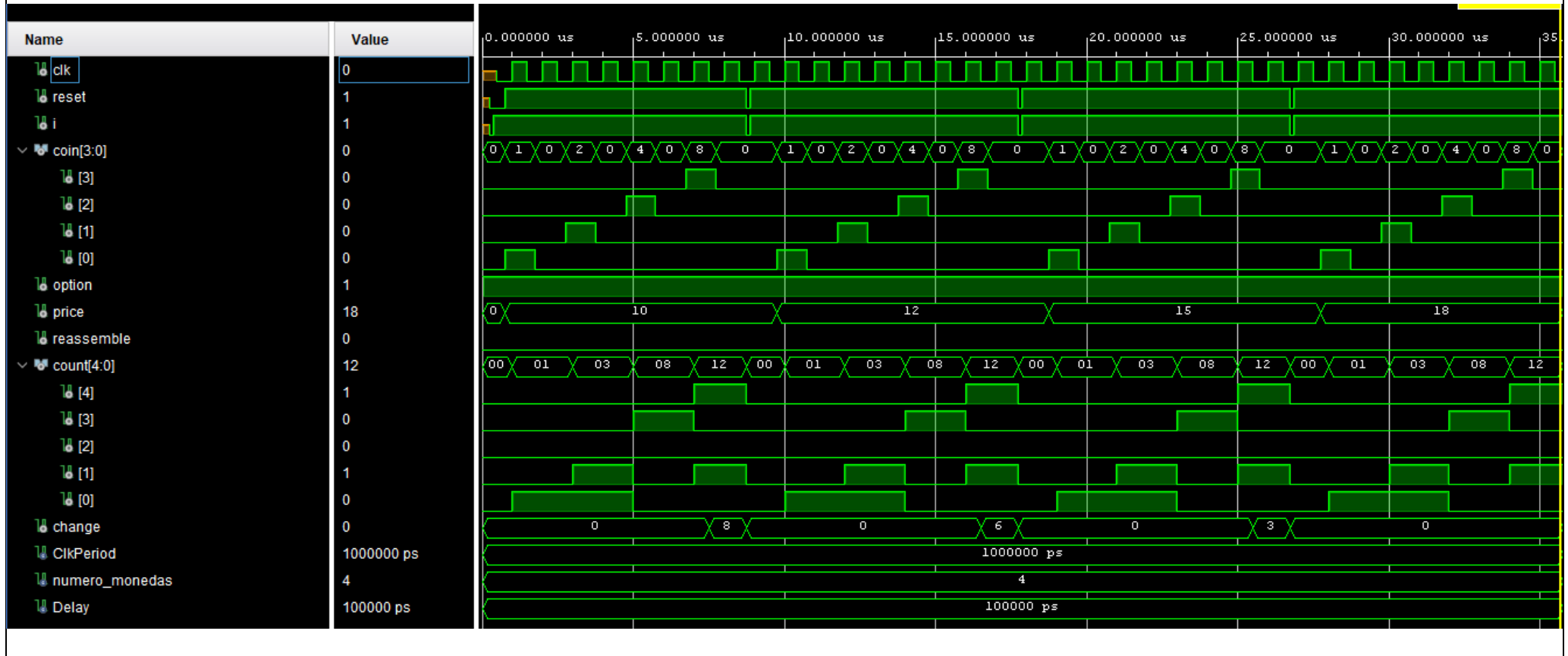
end Behavioral;
```

Del código mostrado cabe mencionar el apartado de “CONDICIONES INICIALES DE LAS ENTRADAS” en el cual se incluyen las señales que no dependen internamente de la propia entidad. Es por esto que en la asignación de estas señales se ha dejado una línea comentada para que el usuario pueda decidir cuando quiere cambiar el valor de estas señales externas.

También es importante destacar el uso de “assert” y “report”. Si la simulación no se desarrolla como esperamos se nos avisará mediante un mensaje en la pestaña de la consola, al igual que si la simulación termina correctamente.

La pantalla de simulación se mostrará en una hoja independiente debido a su tamaño:

## SIMULACIÓN ENTIDAD COUNTER



- Se comprueba que, al superar el valor del precio, se guarda en “change” la diferencia entre la cuenta y el precio.
- También se aprecia que el reseteo funciona correctamente, pues se produce la limpieza de las señales.
- Por último, si durante la simulación activásemos “reassemble” se produciría de nuevo un reseteo y si desactivamos “option” se deshabilitaría la introducción de monedas

### FSM\_SELECTION\_tb

En este caso, generaremos un tren de pulsos para ir cambiando el estado de los interruptores de selección de bebida. También se comprobará que mientras no se realiza el reseteo manual después de una venta, no se puede seleccionar una opción.

El código que se mostrará es un fragmento del total, en el que se ha quitado las partes comunes y el desarrollo de instanciaciones y declaraciones.

```
entity FSM_SELECTION_tb is
end FSM_SELECTION_tb;

architecture Behavioral of FSM_SELECTION_tb is

-- DECLARACION DE COMPONENTES
    component FSM_SELECTION is

-- DECLARACION DE CONSTANTES
        (...)

-- DECLARACION DE SEÑALES
        (...)
    begin

-- INSTANCIACION DE COMPONENTES
        inst_FSM_SELECTION: FSM_SELECTION

-- GENERACION DE RELOJ
        (...)

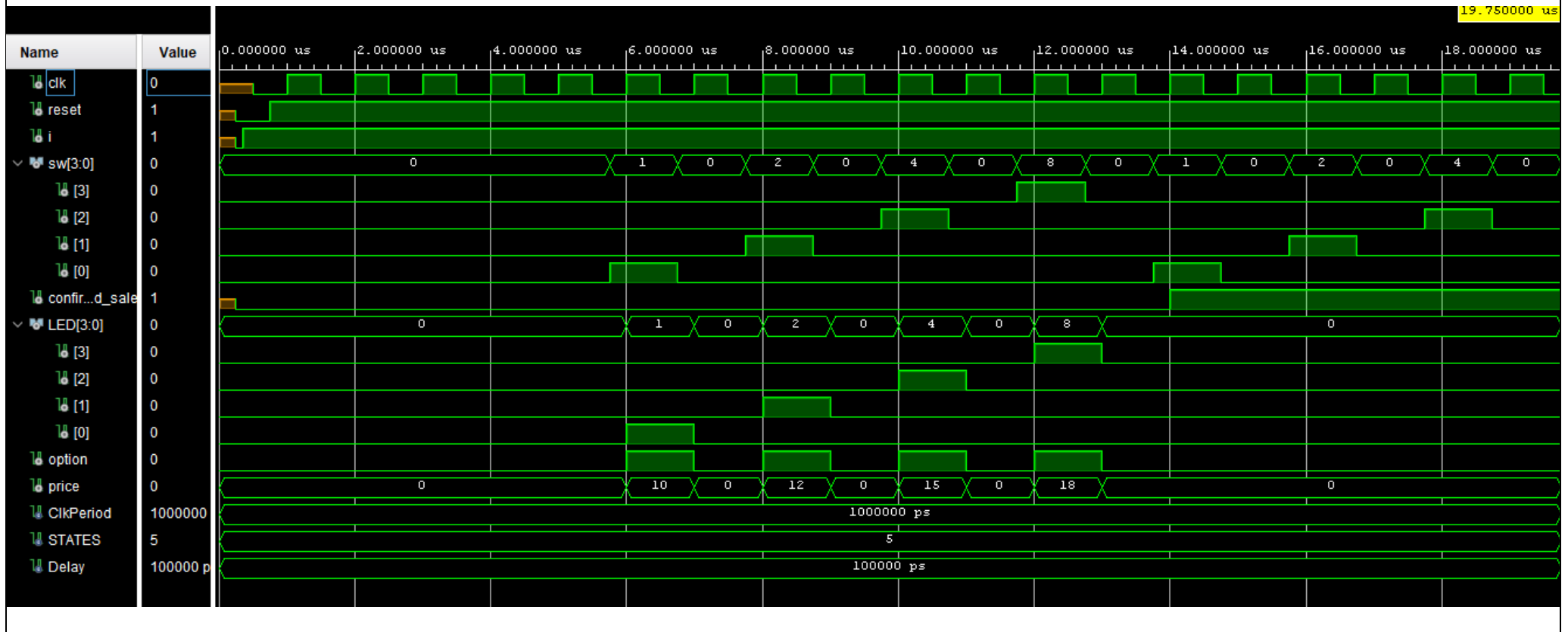
-- RESETEO INICIAL
        (...)

-- DETECCION DE BEBIDA VENDIDA
        confirmed_sale <= '0' after 0.25*ClkPeriod,
                        '1' after 14*ClkPeriod;

-- FUNCIONALIDAD

        FSM_SELECTION behaviour : process
            variable k: natural range 0 to 3;
        begin
            sw <= "0000";
            wait until reset = '1';
            wait for 5*ClkPeriod;
            for i in 0 to STATES+1 loop
                k := i mod 4;
                sw(k) <= '1';
                wait for ClkPeriod;
                sw(k) <= '0';
                wait for ClkPeriod;
            end loop;
            -- Finalizamos la simulacion
            assert false
                report "[PASS]: testbench passed."
                severity failure;
        end process;
    end Behavioral;
```

## SIMULACIÓN ENTIDAD FSM\_SELECTION



- Se comprueba que al seleccionar una opción se activa el LED correspondiente y “price” pasa a tener le valor de la bebida.
- También se aprecia que al estar activo “confirmed\_sale” no se ilumina ningún LED pese a cambiar el valor de los interruptores



## FSM\_PRICE\_tb

Se generará un pulso de monedas que pasaremos a la entidad COUNTER, la cual necesitamos para que nos devuelva la señal “count” y poder evaluar las condiciones de franqueo dentro de la entidad FSM\_PRICE. Se pretende comprobar que al llegar la cuenta al precio establecido se produce la venta y se guarda el valor del cambio en la señal “change”. El código es:

```
entity FSM_SELECTION_tb is
end FSM_SELECTION_tb;

architecture Behavioral of FSM_SELECTION_tb is

-- DECLARACION DE COMPONENTES
    component FSM_PRICE is
    component COUNTER is

-- DECLARACION DE CONSTANTES

-- DECLARACION DE SEÑALES

begin

-- INSTANCIACION DE COMPONENTES
    inst_FSM_SELECTION: FSM_PRICE
    inst_COUNTER: COUNTER

-- GENERACION DE RELOJ

-- RESETEO INICIAL

-- DETECCION DE BEBIDA VENDIDA
    reassemble <= '0' ,
                '1' after 15*ClkPeriod;

-- FUNCIONALIDAD

    FSM_SELECTION_behaviour : process
    variable k: natural range 0 to 3 := 0;
    begin
        coin <= "0000";
        wait until reset = '1';
        price <= 10;

        while (reassemble = '0') loop
            -- Si se produce la venta se deshabilita la maquina de estados
            -- No se puede introducir monedas mientras esta deshabilitada y no se modifica el valor de
change
            if (confirmed_sale = '1') then
                option <= '0';
            else
                option <= '1';
            end if;

            k := k mod numero_monedas;
            coin(k) <= '1';
            wait for ClkPeriod;
            coin(k) <= '0';
            wait for ClkPeriod;
            k := k + 1;

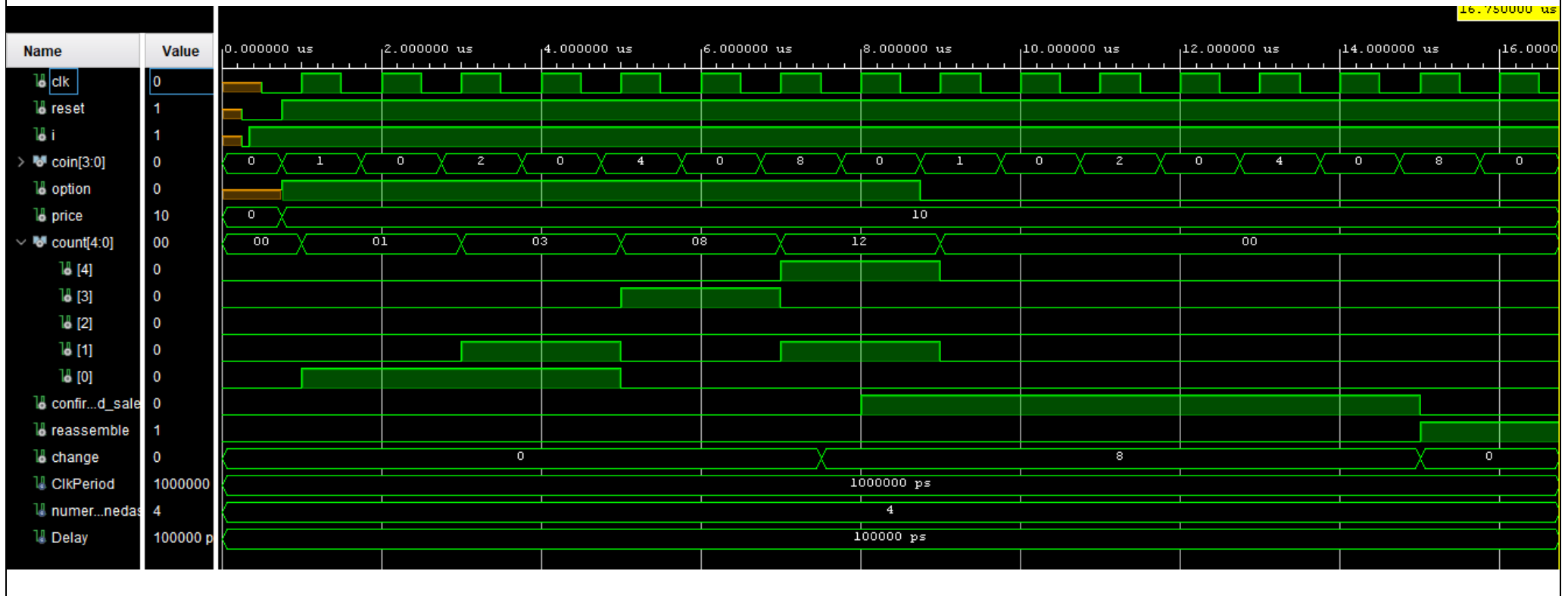
        end loop;

        -- Finalizamos la simulacion
```

```
        assert reassemble = '0'
        report "[PASS]: testbench passed."
        severity failure;
    end process;
end Behavioral;
```

Se ha incorporado como entrada manual “reassemble” que lo activaremos una vez se ha producido la venta. Se verá en la ventana de simulación que al activar “reassemble” se produce el reseteo de la máquina de la señal “count” y la máquina de estados vuelve a estar lista para operar.

## SIMULACIÓN ENTIDAD FSM\_PRICE



- Al superarse el precio ( $0 \times 12 = 18$ ) se guarda el cambio en “change” ( $18 - 10 = 8$ ) y se mantiene en ese estado mientras “confirmed\_sale” esté activo.
- Mientras “confirmed\_sale” está activo, la máquina de estados ignora la entrada de las monedas.
- Una vez se activa el “reassemble” el sistema se limpia y vuelve a estar funcional.