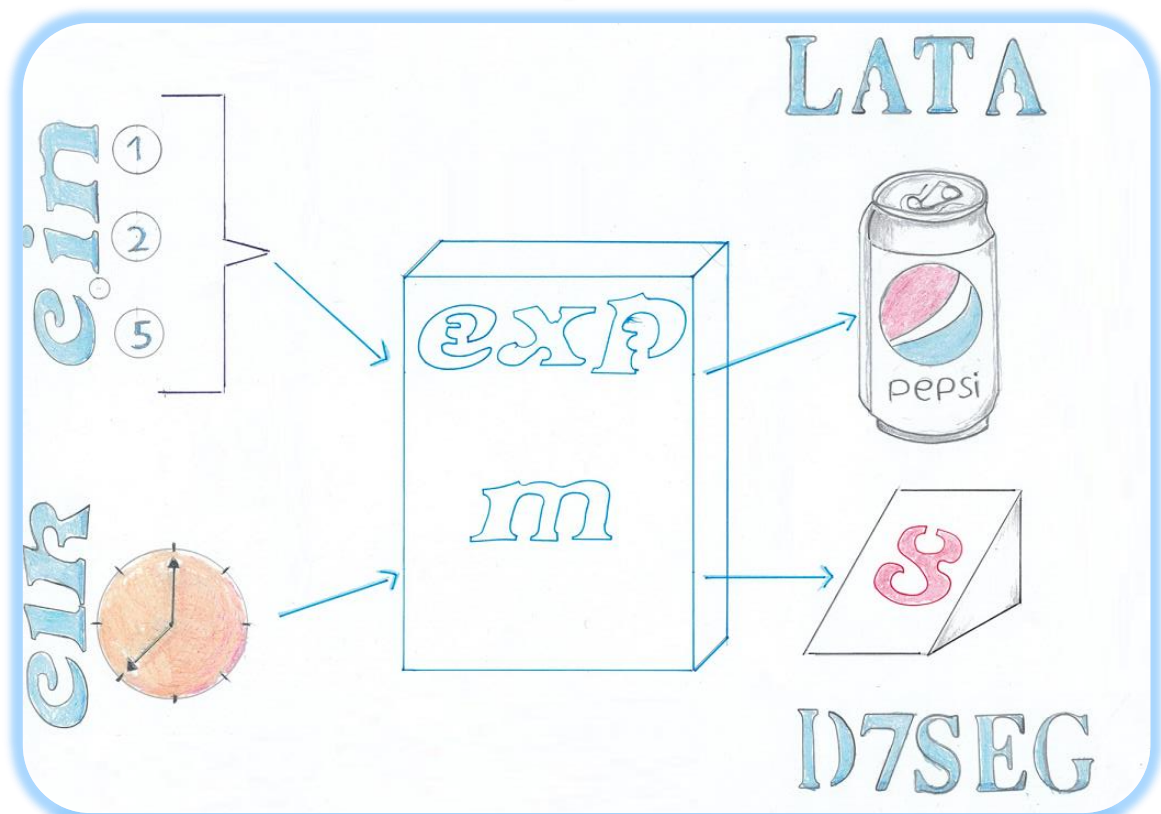




# MÁQUINA EXPENDEDORA



SÁNCHEZ HERNÁNDEZ, CARMEN  
MOLPECERES DÍAZ, SANTIAGO

## Contenido

Explicación del proyecto.....	2
Descripción del trabajo.....	3
Máquina expendedora.....	4
Decodificador 7 segmentos .....	7
Funcionamiento D7.....	8
Multiplexor 2 a 1 .....	9
Funcionamiento Multiplexor.....	10
Selector.....	11
Funcionamiento Selector .....	12
Contador.....	13
Funcionamiento del Contador .....	14
FSM .....	15
Funcionamiento FSM.....	16
Análisis y consumo.....	20
Timing .....	21
Testbench.....	22
Simulación 1 .....	23
Simulación 2 .....	24
Simulación 3 .....	25

## Explicación del proyecto

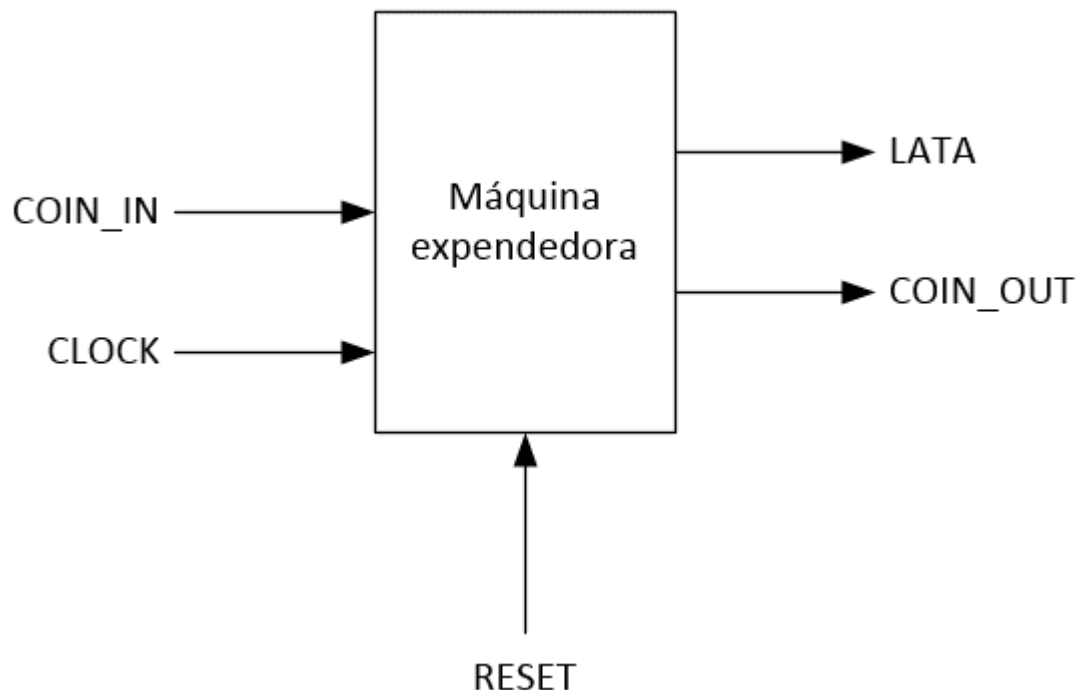
Este trabajo es el proyecto final de la asignatura de Tecnología de Computadores.

Con los conocimientos adquiridos durante el curso del lenguaje VHDL, se ha de codificar una máquina expendedora.

El trabajo consiste en diseñar el sistema de control de una máquina expendedora de bebidas que vende cada lata de refrescos a 2€. Se deben cumplir las siguientes características:

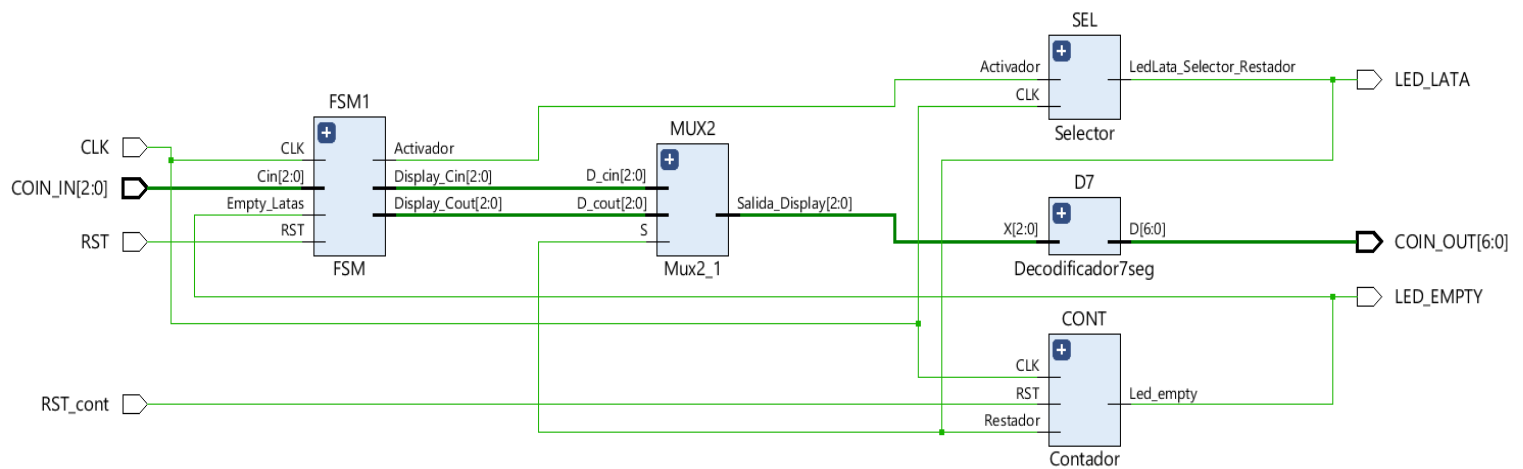
- La FPGA que vamos a usar para este proyecto es la Nexys 4 DDR.
- La máquina puede recibir monedas de 1€ y de 2€ y billetes de 5€.
- La máquina devuelve cambio si se introduce más dinero del que cuesta la lata.
- Como ampliación, se propone a los alumnos la realización de un sistema de contabilización del inventario restante en la máquina expendedora. Para ello, se deberá implementar un sistema de cuenta que active la señal EMPTY cuando se haya agotado el número de latas de bebida disponibles en la máquina. Suponer que inicialmente se carga la máquina expendedora con 3 latas y que el reponedor resetea esta señal al introducir 3 nuevas latas en la máquina.

Diseño:



## Descripción del trabajo

Vamos a proceder a la explicación de como el trabajo ha sido realizado. Para una mejor comprensión, siga el siguiente esquemático.



## Máquina expendedora

La máquina expendedora es un módulo con una arquitectura de tipo Structural, es decir, que se compone de más módulos o componentes. Entre ellos se encuentran:

- FSM: Un autómata finito que marca las etapas del proceso.
- Multiplexor2\_1: Es un pequeño seleccionador en el que decidiremos si mostrar el COIN\_IN o el COIN\_OUT.
- Selector: Condiciona la salida del multiplexor y enciende la señal de LED\_LATA.
- Decodificador 7 segmento: Decodifica la información proveniente del Multiplexor para mandarla al Display 7 segmentos de la placa.
- Contador: Nos dirá si hay o no Stock en nuestra máquina.

```

architecture Structural of Máquina_Expendedora is
component Mux2_1 is
    Port (D_cin,D_cout: in STD_LOGIC_VECTOR( 2 downto 0);
          S:in STD_LOGIC;
          Salida_Display: out STD_LOGIC_VECTOR (2 downto 0)
          );
end component;
component Decodificador7seg is
    Port (X:in STD_LOGIC_VECTOR(2 downto 0);
          D:out STD_LOGIC_VECTOR(6 downto 0)
          );
end component;
component FSM is
    Port (CLK,RST:in STD_LOGIC;
          Cin:in STD_LOGIC_VECTOR(2 downto 0);
          Empty_Latas:in STD_LOGIC;
          Activador: out STD_LOGIC;
          Display_Cin,Display_Cout:out STD_LOGIC_VECTOR(2 downto 0)
          );
end component;
component Selector is
    Port (CLK,Activador:in STD_LOGIC;
          LedLata_Selector_Restador:out STD_LOGIC);
end component;
component Contador is
    Port (CLK,RST,Restador:in STD_LOGIC;
          Led_empty:out STD_LOGIC);
end component;
signal SalidaD1,SalidaD2,SalidaMux:STD_LOGIC_VECTOR(2 downto 0);
Signal SalidaContador,Led_Selector,empty:STD_LOGIC;
begin
FSM1 : FSM port map(CLK,RST,COIN_IN,empty,SalidaContador,SalidaD1,SalidaD2);
SEL: Selector port map(CLK,SalidaContador,Led_Selector);
MUX2: Mux2_1 port map(SalidaD1,SalidaD2,Led_Selector,SalidaMux);
D7: Decodificador7seg port map(SalidaMux,COIN_OUT);
CONT: Contador port map(CLK,RST_cont,Led_Selector,empty);
LED_LATA<=Led_Selector;
LED_EMPTY<=empty;
end Structural;

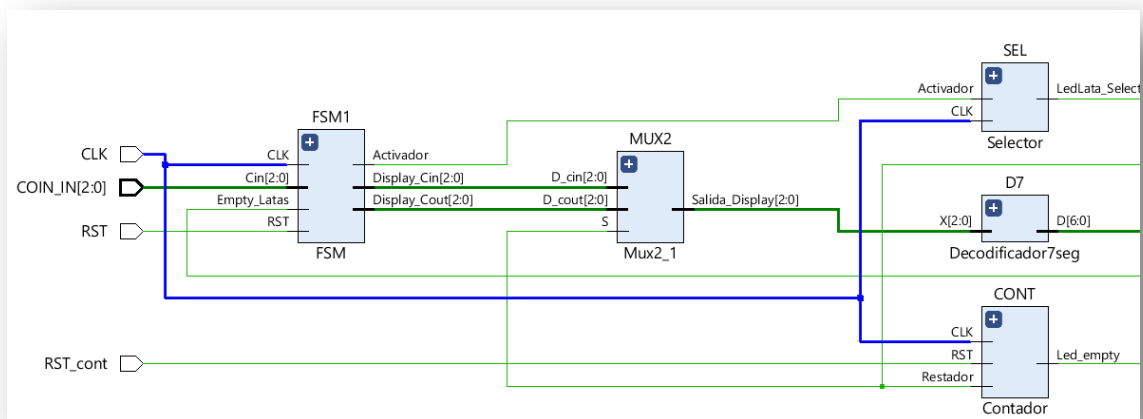
```

La máquina expendedora está definida como:

```
entity Máquina_Expendedora is
  Port (CLK,RST: in STD_LOGIC;
        RST_cont:in STD_LOGIC;
        COIN_IN:in STD_LOGIC_VECTOR(2 downto 0);
        COIN_OUT: out STD_LOGIC_VECTOR(6 downto 0);
        LED_LATA:out STD_LOGIC;
        LED_EMPTY:out STD_LOGIC
  );
end Máquina_Expendedora;
```

➤ Entradas

- CLK: Señal de reloj utilizada para sincronizar el funcionamiento de la máquina.



- RST: Señal de reinicio. También se utiliza para devolver cambio si se ha introducido 1€ por error y no se desea comprar la lata.
- COIN\_IN-Cin: Señal que indica el tipo de moneda/billete introducido.
- RST\_cont: Señal de reinicio. Resetea el stock de latas.

➤ Salidas:

- COIN\_OUT: Señal que indica el tipo de moneda devuelto.
- LED\_LATA: Señal que activa la trampilla para que el usuario pueda recoger la lata.
- LED\_EMPTY: Señal que indica que no hay stock.

Para un mayor entendimiento, profundizaremos primero en los módulos más sencillos, hasta los más complicados.

## Decodificador 7 segmentos

El decodificador 7 segmentos es un pequeño modulo en el que, según su input, el cual es un numero en binario en este trabajo, mandara las señales de activación a los leds del Display 7 segmentos de la placa.

El Decodificador de este trabajo se define como:

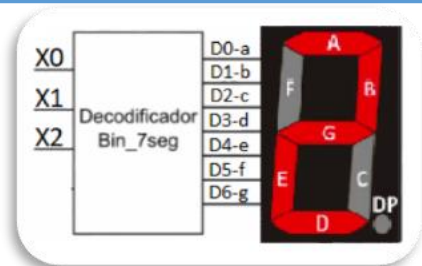
```
entity Decodificador7seg is
```

```
  Port (X:in STD_LOGIC_VECTOR(2 downto 0);
```

```
        D:out STD_LOGIC_VECTOR(6 downto 0)
```

```
  );
```

```
end Decodificador7seg;
```



➤ Entradas

○ X:

- Es un vector en de 3 bits dado que como máximo vamos a tener 6€, que en binario es “110”.

➤ Salidas

○ D:

- Es un vector de 7 bits. Este vector va directamente al display 7 segmentos de la placa, activando los segmentos pertinentes para representar el número.



## Funcionamiento D7

La arquitectura de este módulo es de tipo Behavioral, debido a que necesitamos un process para que opere. El tipo de process que hemos usado es implícito con el comando `with X select`.

D va a adoptar los valores asignados a cada entrada. Es decir,

por ejemplo:

- Si X="000" -> 0 en Binario
  - D=>"0000001".
- Si X="001" -> 1 en Binario
  - D=>"1001111"

D cambiará su información cada vez que X cambie, debido a que es un proceso asíncrono.

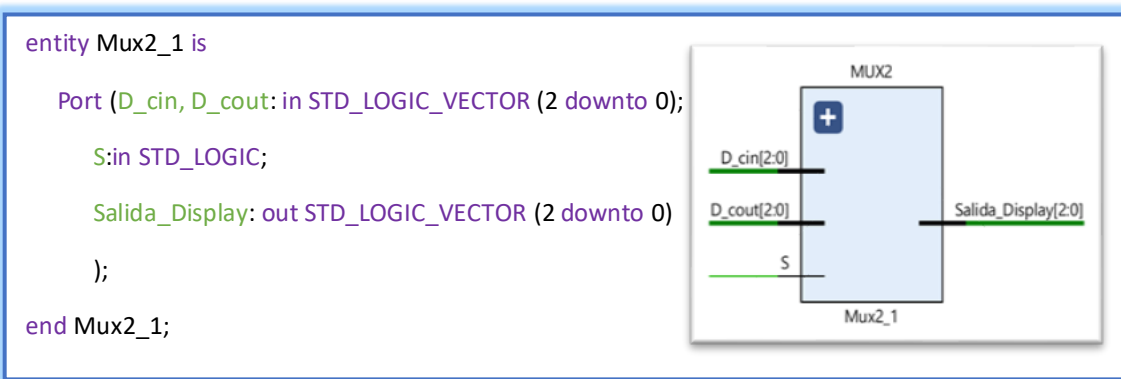
Si por algún error X no adopta los valores de (0,1,2,3,5,6), se apagarán todos los segmentos poniendo toda la salida D a 1, es decir D="1111111".

```
architecture Behavioral of Decodificador7seg is
begin
  with X select
    D<="0000001" when "000",--0
      "1001111" when "001",--1
      "0010010" when "010",--2
      "0000110" when "011",--3
      "0100100" when "101",--5
      "0100000" when "110",--6
      "1111111" when others;--todo apagado
  end Behavioral;
```

## Multiplexor 2 a 1

El multiplexor es un módulo que cuenta con una entrada especial que casi siempre se denomina “Data Select” o “S” con el cual se selecciona una de las diferentes entradas de información para que sea esta la que se traslade a la salida en cada momento.

El multiplexor de este trabajo se define como:



### ➤ Entradas

- D\_cin:
  - Es la señal de entrada que recibe la cantidad de dinero que ha sido introducido.
- D\_cout:
  - Es la cantidad de dinero que debe devolver la máquina.
- S:
  - Es el selector del multiplexor. Esta entrada, determinará si la salida ha de ser la información que está almacenada en la primera entrada, es decir, la posición 0 (D\_cin) o en la segunda, la cual le corresponde la posición 1 (D\_Cout).

### ➤ Salida

- Salida\_Display:
  - Obtendrá el valor determinado por S.

## Funcionamiento Multiplexor

En este módulo, repetimos la arquitectura Behavioral, dado que necesitamos un process para poder operar.

Este un módulo va a actuar independientemente del CLOCK(CLK) que lleva nuestra máquina de estados.

Solo nos enfocamos en la lista de sensibilidad del process es decir D\_cin, D\_cout y S. Cuando alguno de estos parámetros cambie, entraremos en el proceso.

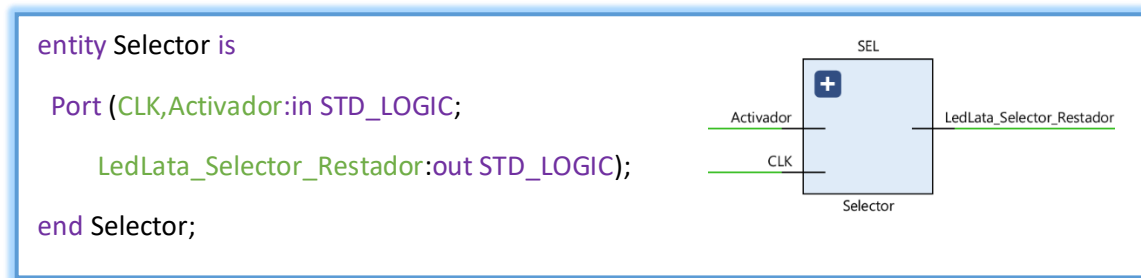
- Si S=0
  - Salida\_Display=D\_cin;
- Si S=1
  - Salida\_Display=D\_cout;

```
architecture Behavioral of Mux2_1 is
begin
  Process(D_cin, D_cout,S) is begin
    if(S='0') then
      Salida_Display<=D_cin;
    else
      Salida_Display<=D_cout;
    end if;
  end process;
end Behavioral;
```

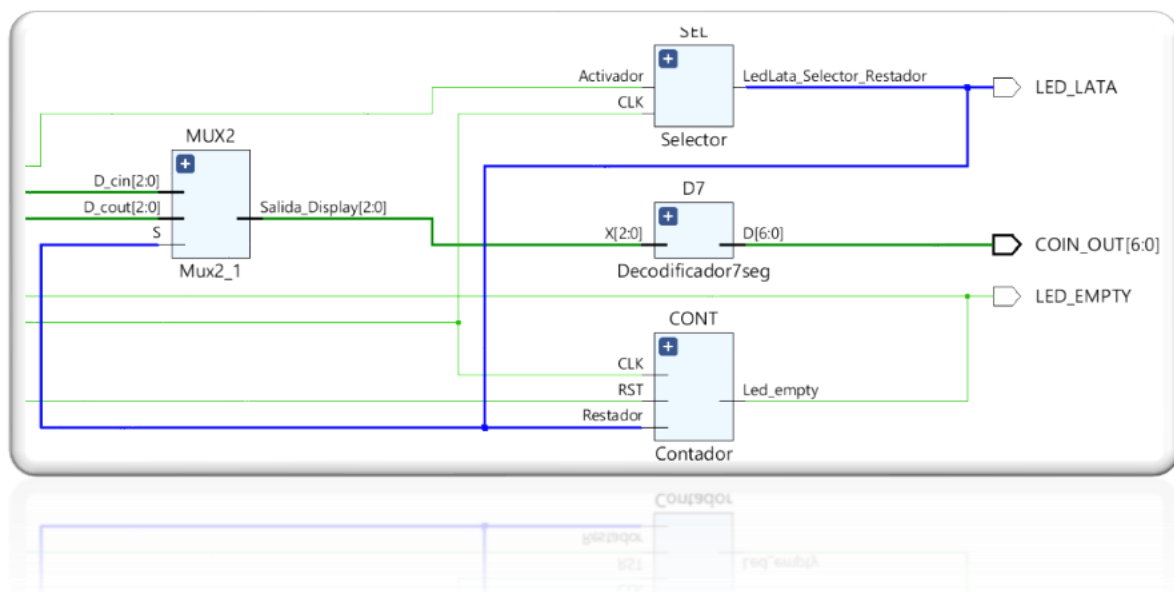
## Selector

Este módulo, es un pequeño componente que hemos diseñado para gestionar la entrada S del multiplexor.

El Selector de este trabajo se define como:



- Entradas:
  - CLK: Reloj que va a determinar un proceso síncrono.
  - Activador: Esta señal será mejor explicada en la explicación de la máquina de estados. Por el momento, nos vamos a quedar con la idea de que es una señal que nos indica que la máquina está preparada para dar la lata.
- Salidas:
  - LedLata\_Selector\_Restador: Esta salida va a obtener 3 funciones:
    - Va a determinar si el Led que nos indica que hay una lata en la bandeja se active y o no, “LedLata”.
    - Es el “Data Selection” del multiplexor, es decir, que marcará si el display muestra la cantidad de dinero introducida, o el dinero que debemos devolver.” Selector”.
    - Debido a que es una señal que nos indica que hemos entregado una lata, para la ampliación también nos indica que tenemos una lata menos, ”Restador”.



## Funcionamiento Selector

Este componente usa una arquitectura de tipo Behavioral ya que dependemos de un process.

El selector tiene un solo proceso cuyo rasgo a destacar es que es síncrono.

La lista de sensibilidad del proceso se compone por el CLK del circuito y la señal de entrada denominada “Activador”. Cuando alguno de estos dos parámetros cambie, entraremos al proceso.

Este proceso está muy encapsulado, por tanto, vamos a verlo por partes.

- <sup>1</sup> Si hay un flanco de bajada por parte del CLK
  - <sup>2</sup> Y si Activador=1
    - LedLata\_Selector\_Restador=1
      - Led-> se enciende.
      - Selector del multiplexor vale 1.
      - Restador =1-> Restamos.
  - <sup>2</sup> Si no vale 1 (Activador=0)
    - LedLata\_Selector\_Restador=0.
      - Led-> se apaga.
      - Selector del multiplexor vale 0.
      - Restador= 0-> no restamos.

```
architecture Behavioral of Selector is
begin
process(CLK,Activador) is begin
    if 1(falling_edge(CLK))then -- bajada
        if 2(Activador='1')then
            LedLata_Selector_Restador<='1';
        else 2
            LedLata_Selector_Restador<='0';
        end if;
    end if;
end process;
end Behavioral;
```

## Contador

Este componente forma parte de la ampliación.

Es un pequeño componente que actúa como inventario. Nos determinará si la máquina expendedora tiene stock o no.

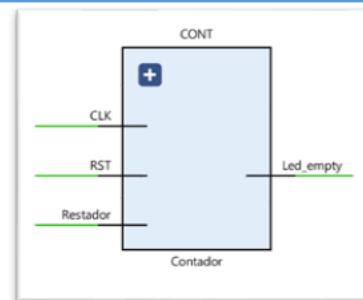
El contador de este trabajo se define como:

```
entity Contador is
```

```
Port (CLK,RST,Restador:in STD_LOGIC;
```

```
      Led_empty:out STD_LOGIC);
```

```
end Contador;
```

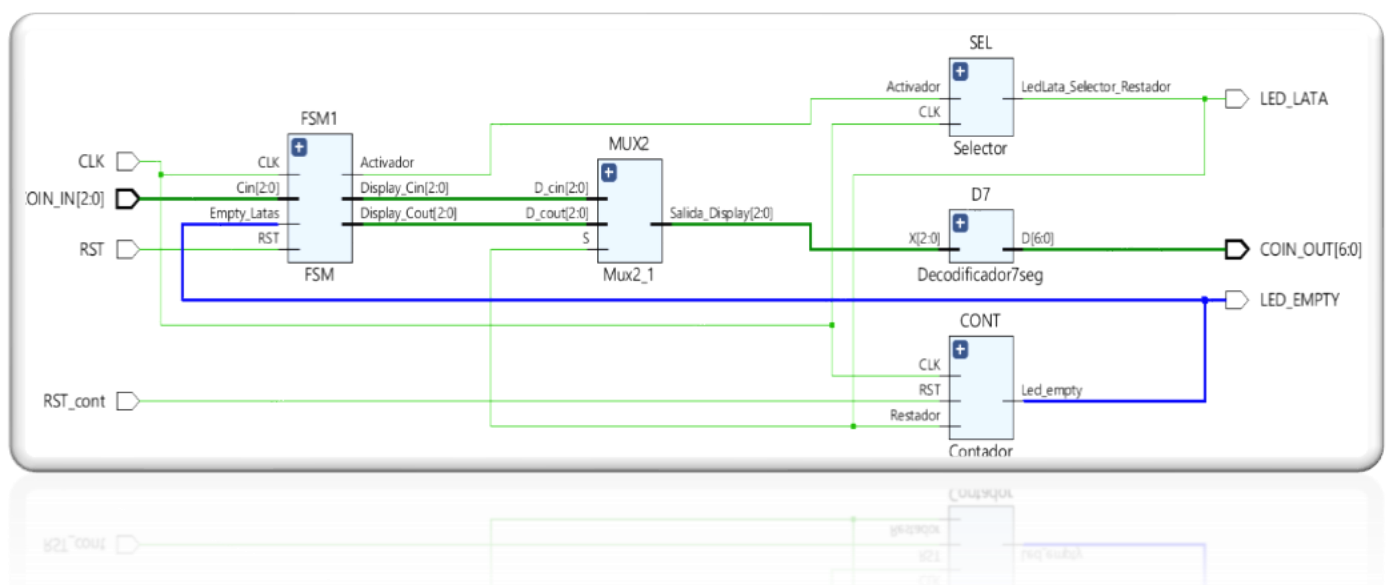


### ➤ Entradas

- CLK: Señal conectada al reloj (clock) del circuito, determinará nuestro proceso síncrono.
- RST: Señal que reiniciará nuestro stock a su máxima capacidad.
- Restador: Señal que nos indicará cuando hemos entregado una lata y hemos de registrar una pérdida en el inventario.

### ➤ Salidas

- Led\_empty: Tiene 2 funciones
  - Activa un Led que indica que la máquina no tiene Stock.
  - Indica a la FSM si la maquina tiene stock para poder funcionar.



## Funcionamiento del Contador

La arquitectura de este componente es de tipo Behavioral ya que dependemos de un process.

Primero hemos de declarar una señal de tipo Integer (número entero) que va a adoptar los valores desde 0 a 3. Lo que quiere decir que vamos a tener máximo 3 latas.

La lista de sensibilidad va determinada por un CLOCK, un RESET y la señal de Activador. Si alguna de estas señales cambia, entraremos en el proceso.

En primer lugar, tenemos un RST asíncrono. En el que, si la señal está a 1, esta subirá el contador de latas al máximo, es decir, 3.

Si el RST está a nivel bajo (0), entramos en la parte síncrona del process.

Si estas condiciones se cumplen

- Hay un flanco de subida
- Restador = 1
- Cuenta > 0



RESTAMOS 1

```
architecture Behavioral of Contador is
    signal cuenta: integer range 0 to 3;
begin
    process(CLK,RST,Activador) is begin
        if 1(RST='1')then
            cuenta<=3;
        elsif 2(rising_edge(CLK)) then
            if 3(Restador='1')then
                if 4(cuenta>0)then
                    cuenta<=cuenta-1;
                end if;
            end if;
        end if;
    end process;
    Led_empty<='1' when (cuenta=0) else '0';
end Behavioral;
```

Luego tenemos un process implícito.

Led\_Empty valdrá 1 cuando cuenta=0 si no, permanecerá a 0.

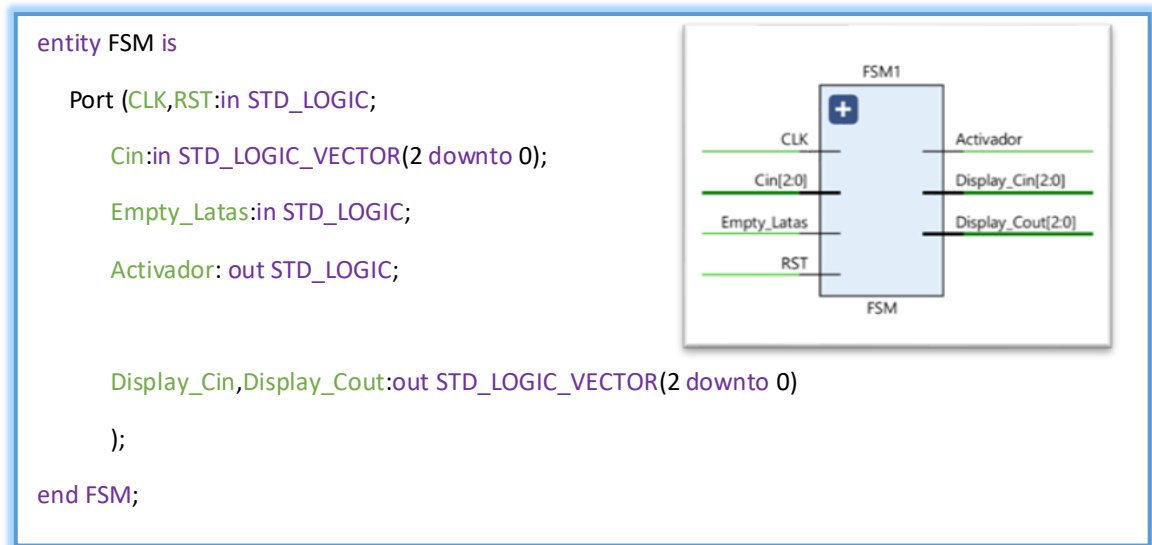
## FSM

Para todo el proceso en el que se tratan los datos de entrada y de salida, hemos diseñado una máquina de estados (FSM – Finite State Machine).

Esta máquina de estados es de Moore debido a que sus salidas no dependen de las entradas, es decir, sus salidas solo dependen del estado en el que nos encontremos.

La señal de Empty\_Latas, aunque sea una salida proveniente de Contador, es una entrada más a la FSM.

La FSM de este trabajo se define como:



### ➤ Entradas

- CLK: La máquina de estados va a tener un proceso de carácter síncrono, es decir, va a entrar en la ejecución de un proceso cuando haya un flanco de subida del reloj.
- RST: Esta entrada es el botón reset que hemos establecido como predeterminado, reiniciara la FSM para ponerla en su estado inicial.
- Cin: Esta entrada está asociada a la variable principal COIN\_IN. Se corresponde a la moneda o billete que hemos introducido.
- Empty\_Latas: Es una señal que proviene del contador de latas, esta indicará a la FSM si hay stock.

### ➤ Salidas

- Activador: Es una señal que indica al selector que ya tenemos los 2€ mínimo para poder dar la lata.
- Display\_Cin: Señal que indica la cantidad que ha sido introducida.
- Display\_Cout: Señal que indica la cantidad que la maquina ha de devolver.



## Funcionamiento FSM

Este componente utiliza una arquitectura Behavioral dado que necesitamos varios procesos para poder actuar.

architecture Behavioral of FSM

### Señales internas

Para poder declarar los estados en VHDL necesitamos las siguientes líneas de código.

```
type State_type is (S0,S1,S2,S3,S4,S5);  
signal CurrentState,NextState:State_type;
```

Estas líneas sirven para definir los estados de S0, S1, S2, S3, S4, S5.

Para ello necesitamos la palabra reservada “type”.

State	Binario	Significado en el trabajo	Combinación
S0	000	No hay dinero/ Esperamos dinero	0€
S1	001	Hay 1€ en la maquina	1€
S2	010	Hay 2€ en la maquina	2€
S3	011	Hay 3€ en la maquina	1€+2€
S4	101	Hay 5€ en la maquina	5€
S5	110	Hay 6€ en la maquina	1€+5€

A continuación, declaramos las señales CurrentState y NextState que son las señales que van a adoptar los valores de la variable State\_type.

DATO IMPORTANTE: Los datos de entrada están pensados para que sea un fácil manejo cuando se implemente en la FPGA. Por tanto:

Tipos de entrada	
Binario para el programa	Cantidad que representa
000	0
001	1
010	2
100	5

## Process Síncrono

La máquina de estados es un componente síncrono. Para actualizar los estados, precisamos de un clock.

Este process tiene dos partes:

➤ Parte asíncrona

- El reset (RST) actúa de forma independiente al reloj. De este modo cuando queramos que nos devuelva el dinero, lo hará sin tener que esperar el flanco de reloj.

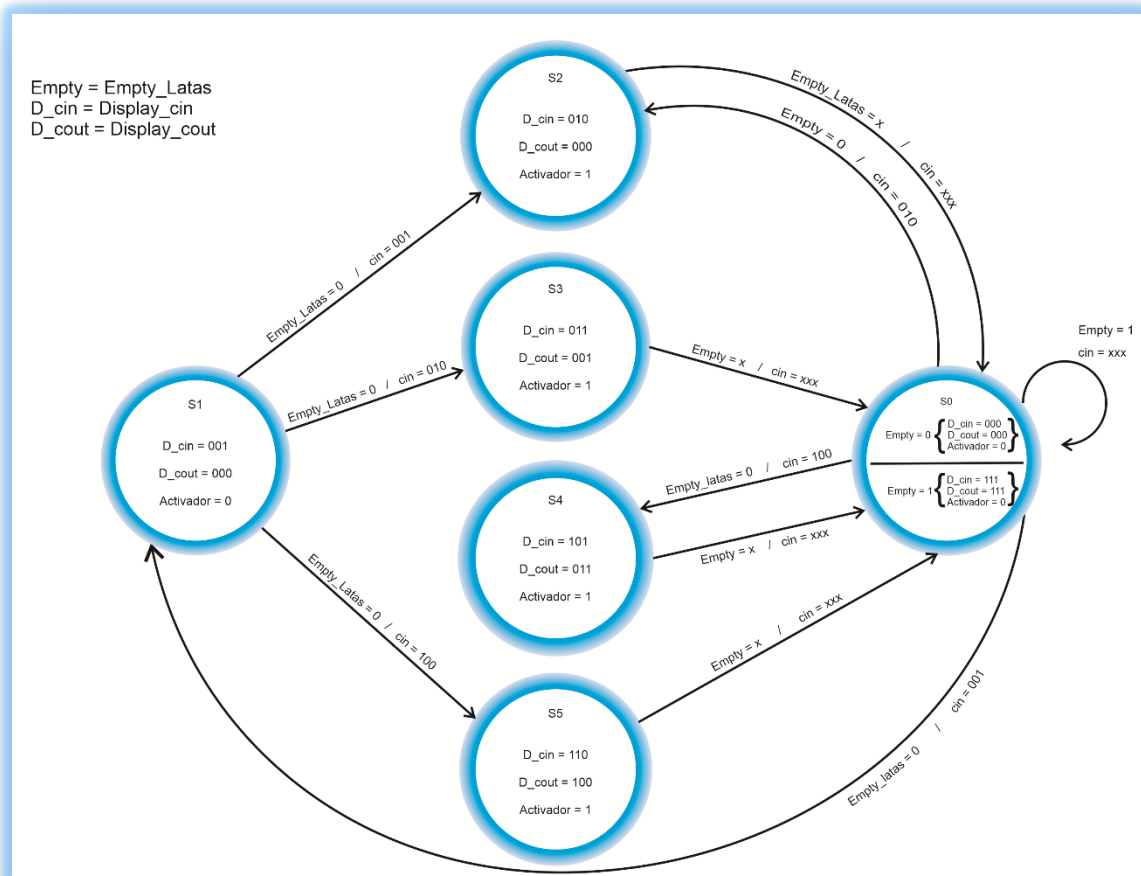
➤ Parte síncrona

Es la parte que precisa el reloj para actuar. Cuando tenemos un flanco de subida.

- El estado actual será el estado siguiente.  
El estado se tiene que estar actualizando aun que las entradas no varíen, y quede en el mismo estado.

```
process (CLK , RST) is begin
    if(RST='1') then
        CurrentState<= S0;
    elsif(rising_edge(CLK))then
        CurrentState<=NextState;
    end if;
end process;
```

Diagrama de estados



## Process Asíncrono

En la tabla de sensibilidad se encuentran CurrentState, Cin, EmptyLatas, que si varía alguna de estas variables, entraremos en el proceso.

Empty\_Latas, determina a la máquina de estados si tenemos stock. Cuando vale 1 entramos en el "if" indicando que el led que afirma que no hay latas se ha activado, por tanto, apagaremos el display 7 segmentos y no nos moveremos del estado S0 hasta que haya stock.

Cuando estamos en S0, Mandamos al multiplexor que el dinero que llevamos introducido es 0 y el que necesitamos devolver es 0. Y como no hemos llegado a un estado en el que tenemos 2€ mínimo. Activador se mantiene a 0 pues no hemos llegado al denominado "estado de aceptación".

Este if es en el que interviene Cin, que es la moneda o billete que le introducimos a la máquina. Por tanto,

- Si Cin=001 -> 1€
  - El siguiente estado será S1
- Si Cin=010 -> 2€
  - El siguiente estado será S2
- Si Cin=100 -> 5€
  - El siguiente estado será S4
- Si es distinto de 1€, 2€ y 5€, implica que hemos metido monedas no validas o que no hemos ingresado nada. Por tanto, el siguiente estado es en el que estamos
- **NextState<=CurrentState;**

Cuando nos encontramos en el S1, hemos de actualizar la información que va al multiplexor. Por tanto, D\_cin=001 -> 1€ y D\_cout=000 -> 0 dado que de momento no ha llegado a los 2€ necesarios.

Y como no hemos llegado a un estado de aceptación. Activador se mantiene a 0.

```
process (CurrentState,Cin,Empty_Latas)is begin
    if(Empty_Latas='1') then
        NextState<=S0;
        Display_Cin<="111";
        Display_Cout<="111";
    else
        case(CurrentState) is
            -- Cuando varíe CurrentState, entraremos a ver los casos en los que estamos
            when S0=>
                Display_Cin<="000";
                Display_Cout<="000";
                Activador<='0';
                if(Cin="001")then --la moneda es 1
                    NextState<=S1;--tenemos 1 moneda
                elsif(Cin="010") then --la moneda es 2
                    NextState<=S2;--tenemos 2e
                elsif(Cin="100") then -- la moneda es 5
                    NextState<=S4; --tenemos 5e
                else
                    NextState<=CurrentState;
                end if;
                when S1=> --ya interpretamos que tenemos 1e
                    Display_Cin<="001";
                    Display_Cout<="000";
                    Activador<='0';
                    if(Cin="001")then --la moneda es 1
                        NextState<=S2;--tenemos 2 monedas
                    elsif(Cin="010") then --la moneda es 2
                        NextState<=S3; --tenemos 3 monedas
                    elsif(Cin="100") then-- la moneda es 5
                        NextState<=S5;-- tenemos 6 e;
                    else
                        NextState<=CurrentState;
                    end if;
        end case;
end process;
```

////////////////////////////////// siguiente pagina

En esta parte del código, encontramos los estados de aceptación.

Para este caso, S3 => tenemos 3 €. Ya podemos dar la lata. Por tanto,

- Display\_Cin=011 -> 3
  - Que se mostrará un periodo de tiempo.
- Display\_Cout=001-> ya que como la lata cuesta 2€, le tenemos que devolver 1€, y la cantidad se mostrará en el display un periodo de tiempo.

(Recordad que las salidas Display\_Cout, Display\_Cin, van al multiplexor)

- Activador =1  
Hay que recordar que va al selector esta salida, por tanto.
  - Activará el LED\_LATA.
  - Enviará al restador la señal para eliminar una lata del inventario.
  - Dira al multiplexor que dato ha de aparecer en el display.

Todo se basa en el funcionamiento del Selector. (pag.10).

- Volvemos al estado S0, porque regresamos a “la espera de monedas”.

El caso “when others” surge cuando hay un tipo de error en la máquina, y nos salimos de los estados predeterminados. Por tanto, actúa como un RESET que volvemos al estado S0.

when S2=>

```
Display_Cin<="010";
Display_Cout<="000";
Activador<='1';
NextState<=S0;
```

when S3=>

```
Display_Cin<="011";
Display_Cout<="001";
Activador<='1';
NextState<=S0;
```

when S4=>

```
Display_Cin<="101";
Display_Cout<="011";
Activador<='1';
NextState<=S0;
```

when S5=>

```
Display_Cin<="110";
Display_Cout<="100";
Activador<='1';
NextState<=S0;
```

when others=>

```
NextState<=S0;
```

end case;

end if;

end process;

## Análisis y consumo

El análisis del área es el método de contabilizar el número de recursos, es el análisis de potencia de la lista de redes que hemos implementado.

El consumo estático es el consumo adjunto de la FPGA debido a los transistores que la forman. Depende de la temperatura ambiente y la tensión de la alimentación.

En nuestro caso, el consumo estático ha concluido con un 0.097W, lo que quiere decir que es un 97% en nuestro análisis.

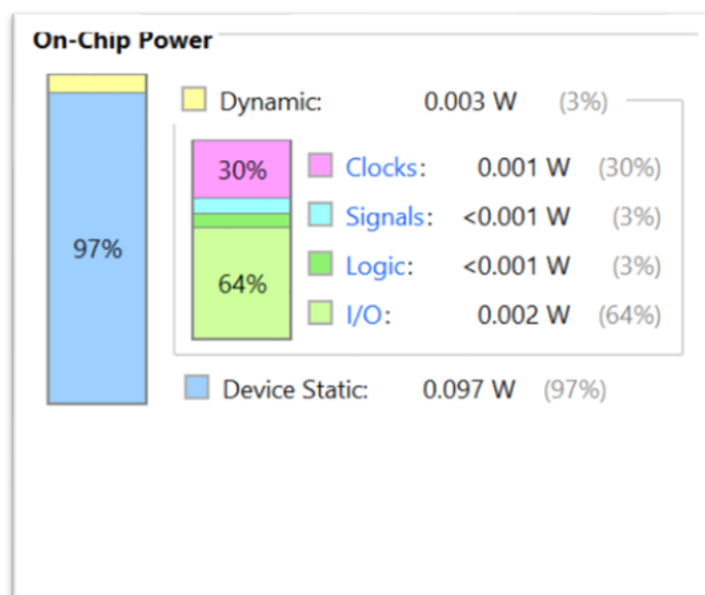
Por otra parte, el consumo dinámico, es el que depende de los valores de tensión, frecuencia, y los recursos activos en cada momento específico del análisis.

En nuestro caso, el consumo dinámico se valora en un 3%, el cual ha concluido con un consumo de 0.003W. Nuestro consumo dinámico incluye el clock, las señales, que implican data, clock enable, set y reset respectivamente; logic, y I/O.

- Clocks -> 0.001W
- Signals < 0.001W
- Data < 0.001W
- Clock enable < 0.001W
- Set < 0.001W
- Reset < 0.001W
- Logic < 0.001W
- I/O -> 0.002W

Este es nuestro consumo dinámico de la máquina expendedora de bebidas.

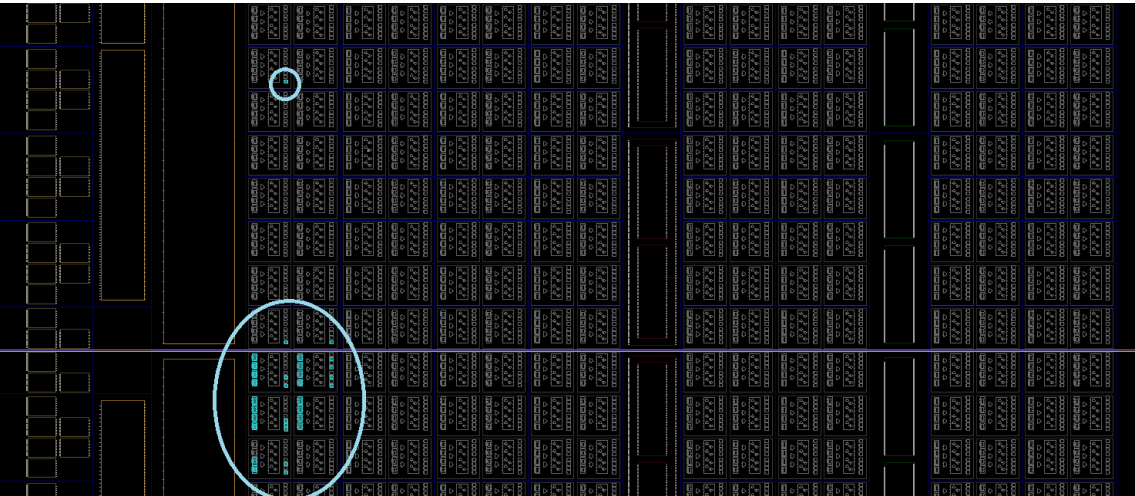
Concluyendo, en consumo On-Chip Power es la suma de los consumos anteriormente explicados.



Es decir, nuestro análisis, nuestro consumo On-Chip, concluiría con un 0.1W.

Total On-Chip Power:	0.1 W
Design Power Budget:	Not Specified
Power Budget Margin:	N/A
Junction Temperature:	25,5°C
Thermal Margin:	59,5°C (12,9 W)
Effective $\theta$ JA:	4,6°C/W
Power supplied to off-chip devices:	0 W

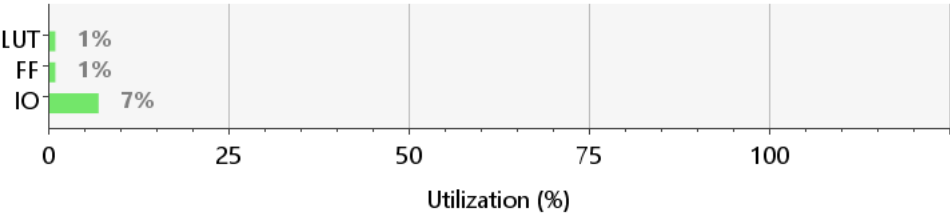
Analizando el consumo por área, vemos que realmente usamos muy pocos recursos de la FPGA.



Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	Block RAM Tile (135)	Bonded IPADs (2)	BUFIO (24)
Maquina_Expendedora	18	14	8	18	14	15	1
CONT (Contador)	3	2	2	3	0	0	0
FSM1 (FSM)	15	10	5	15	0	0	0
SEL (Selector)	0	2	2	0	0	0	0

Summary

Resource	Utilization	Available	Utilization %
LUT	18	63400	0.03
FF	14	126800	0.01
IO	15	210	7.14



Timing

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 2,723 ns	Worst Hold Slack (WHS): 0,268 ns	Worst Pulse Width Slack (WPWS): 4,500 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 10	Total Number of Endpoints: 10	Total Number of Endpoints: 8

All user specified timing constraints are met.

## Testbench

El testbench o también llamado banco de pruebas es una simulación de comportamiento que nos permite verificar que el circuito se comporta según lo esperado.

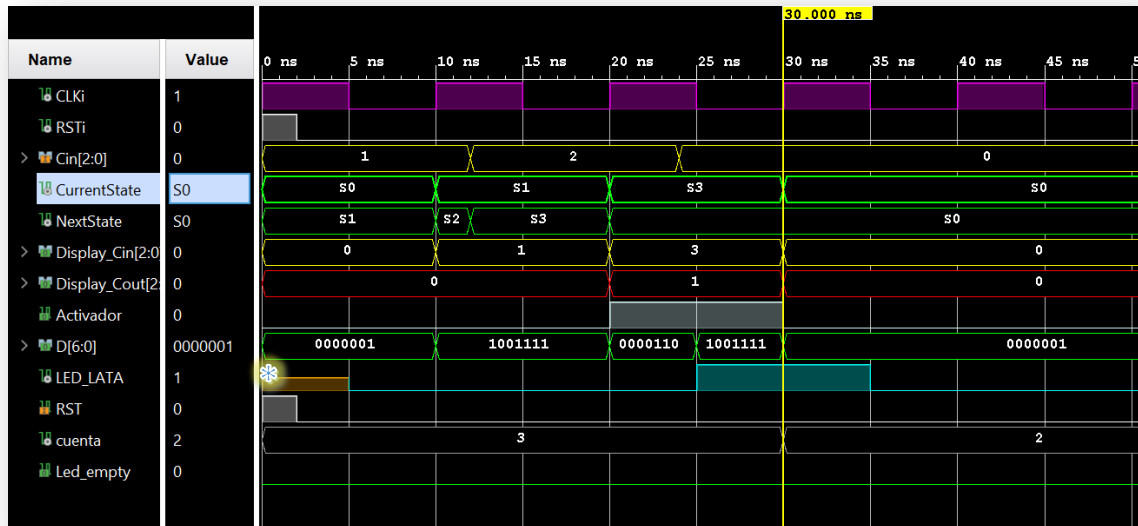
Para ello, Vivado, el software con el que hemos realizado el trabajo nos genera un fichero específico para testear el programa.

LEYENDA	
Variable	Significado
CLKi	Reloj del circuito
RSTi	Botón de reset de FSM
Cin	Dinero introducido
CurrentState	Estado en el que nos encontramos
NextState	Estado siguiente según la entrada Cin
Display_cin	Cantidad acumulada de dinero
Display_Cout	Cantidad de devolución
Activador	Señal que indica un estado de aceptación
D	Lo que sale por el display. (Mirar la tabla pg.6)
LED_LATA	Indicador de que la lata ha sido entregada
RST	Reset por parte del contador
Cuenta	Numero de latas en el Stock
Led_Empty	Señal que indica que no hay latas


## Simulación 1

Simulación en la que introducimos 2 monedas seguidas.

Esta secuencia se termina a los 30ns (ns= nanosegundos)



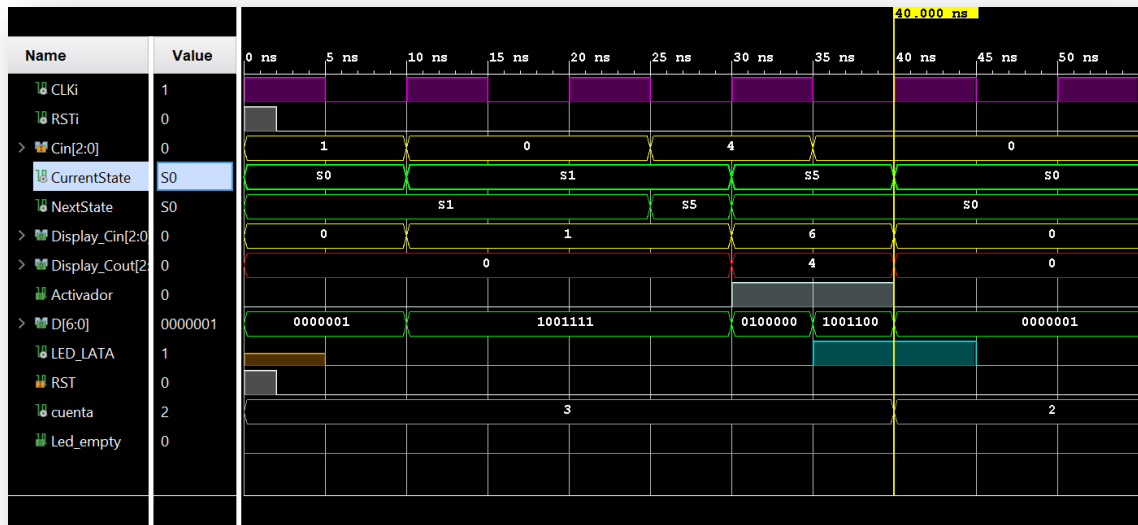
### Aspectos que observar

- Esta simulación termina a los 30n.
- El CLK actúa con un periodo de 10ns.
- Los Reset de la FSM y del contador, se activan al principio para poder poner todo a su estado inicial.
- A los 25 ns de empezar, veremos que la variable D hace un cambio breve. Este se corresponde a el cambio entre la cantidad de dinero acumulada y la devolución.
-  Led\_Lata, tiene un periodo de 5ns en el que se muestra marrón. Led\_Lata obtiene su valor a través de los flancos de bajada, puesto que el simulador empieza con un flanco de subida, Led\_Lata se mantiene como "U" (valor no definido) hasta el primer flanco de bajada.
- A los 30ns volvemos al estado S0 y cuenta desciende 1 nivel.



## Simulación 2

En este caso introducimos 1€ y al cabo de un rato introducimos un billete de 5€.

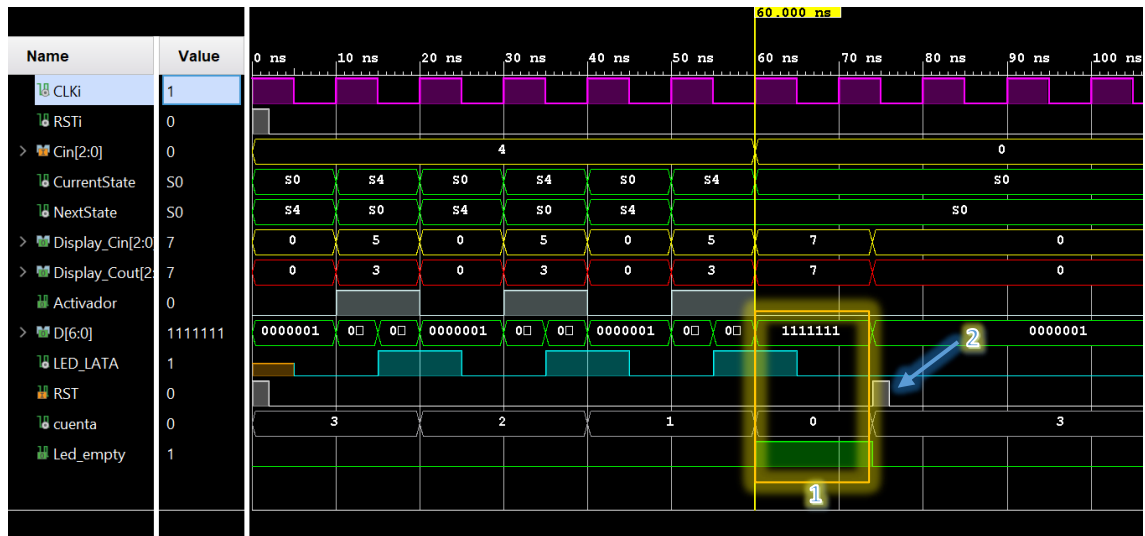


Aspectos que observar:

- Esta simulación termina a los 40 ns.
- Procedimiento:
  - Primero metemos 1€.
  - Luego dejamos que el tiempo pase para corroborar que, si no hacemos nada, la máquina se mantiene constante en ese estado.
  - Metemos 5€ (recordar que para este trabajo “100” es el billete de 5, pero debido a que el simulador está en binario, esta muestra 4). Y nos mantenemos hasta llegar al estado de aceptación.

### Simulación 3

Este caso es cuando nos quedamos sin stock.



- Esta Simulación finaliza a los 75ns.
- En esta simulación solo hay que centrarse en los estados y en la variable Cuenta.
- Mantenemos introducidos 5€ constantes, para que el proceso sea más rápido.
- Cuando llegamos a los 60 ns:
  - 1 Nos damos cuenta de que el contador se pone a cero, encendiendo Led\_empty, lo que nos lleva a que D este a 1111111 (Apaga todo). Esto quiere decir que hemos entrado en un punto en el que la FSM deja de funcionar.
  - 2 Cuando RST se activa, vemos que el contador vuelve a 3 y D=0000001 es decir (0) lo que implica que vuelve a funcionar.