TECNOLOGÍA DE COMPUTADORES

Máquina Expendedora.

Por Miguel Oliver y Guillermo Herranz.

Índice

Introducción

Según los conocimientos adquiridos en las prácticas de la asignatura “Tecnología de Computadores”, se ha realizado el diseño de una máquina expendedora para poner en práctica dichos conocimientos. El desafío principal radica en la implementación de un mecanismo que permita manejar monedas y billetes de distintos valores, el retorno del cambio y la disponibilidad de las latas, siendo esta última una tarea opcional.

Descripción del trabajo

En este trabajo hemos realizado una máquina expendedora en código VHDL que funciona de manera que acepta 3 tipos de monedas o billetes (1, 2 o 5 euros) y cuando detecta que se han introducido 2 euros dispensa una lata de refresco. Si detecta que se han introducido más euros de lo que cuesta la lata dispensa dicha lata y a continuación devuelve el dinero restante, así como si detecta que se ha introducido solo 1 euro y se pulsa el botón de reset ésta devuelve el dinero introducido.

Como parte del trabajo opcional hemos añadido un contador que va disminuyendo a medida que se compran latas y si el propietario pulsa el botón de rellenar se rellena el stock de latas. Está por defecto en 3 por facilidad de testeo, pero el número se podría cambiar para acomodarse al stock de una máquina de verdad.

Explicación del código y simulación

Diseño:

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


entity m_expendedora is

    port(
        coin_in : in std_logic_vector(1 downto 0);
        rst, clk : in std_logic;
        coin_out : out std_logic_vector(1 downto 0);
        lata: out std_logic
    );

end m_expendedora;
```

En esta primera parte podemos observar las librerías utilizadas (IEEE) y la entidad de la máquina expendedora que se compone de un vector de 2 bits para la entrada de la moneda, un bit de entrada tanto para el reset y el reloj (clk), un vector de salida de 2 bits para la devolución de la moneda y un bit de salida para la salida de la lata de refresco.

```
architecture Behavioral of m_expendedora is

type state_type is (S0, S1);
signal state, nxstate : state_type;

begin

    process(clk, rst) begin
        if(rising_edge(clk)) then
            if(rst = '1') then
                state <= S0;
            else
                state <= nxstate;
            end if;
         end if;
    end process;
```

Aquí empezamos a declarar la arquitectura empezando por las señales. Hacemos un tipo de señal que utilizamos para los estados y hacemos dos señales de ese tipo, S0 y S1.

Cuando empezamos el funcionamiento de la arquitectura hacemos un process en el que vamos a hacer la asignación de los estados y el reset. En la lista de sensibilidad del process ponemos el clk y el reset y cuando el clk este en un flanco de subida ponemos el estado en S0 si el reset está activo (a nivel alto) o asignamos al estado actual el next state asignado en el anterior flanco de reloj.

```
process(state, coin_in) begin

    case state is

        when S0 =>

            coin_out <= "00";
            lata <= '0';

            if(coin_in = "01") then
                nxstate <= S1;

            elsif coin_in = "10" then
                lata <= '1';
                nxstate <= S0;

            elsif coin_in = "11" then
                lata <= '1';
                coin_out <= "10";
                nxstate <= S0;

            elsif coin_in = "00" then
                nxstate <= S0;

            end if;
```

Índice de monedas de entrada:
00 = 0 euros
01 = 1 euros
10 = 2 euros
11 = 5 euros

Índice de monedas de salida:
00 = 0 euros
01 = 1 euro
10 = 3 euros
11 = 4 euros

En el segundo process utilizamos como lista de sensibilidad el estado y la introducción de moneda.
Cuando estamos en el estado S0 reseteamos la devolución de moneda y la salida de la lata.
Si el usuario introduce una moneda de 1 euro pasa al estado S1.
Si introduce una moneda de 2 euros devolvemos una lata y nos quedamos en el estado S0.
Si se introducen 5 euros devolvemos una lata, devolvemos 3 euros y nos quedamos en el estado S0.
Por último, si no se introduce dinero nos quedamos en S0.

```
            when S1 =>

                if (rst = '1') then -- Devuelve 1$ si se pulsa reset (la persona al final no quiere comprar la lata).
                    coin_out <= "01";
                    nxstate <= S0;

                elsif(coin_in = "01") then
                    lata <= '1';
                    nxstate <= S0;

                elsif coin_in = "10" then
                    lata <= '1';
                    coin_out <= "01";
                    nxstate <= S0;

                elsif coin_in = "11" then
                    lata <= '1';
                    coin_out <= "11";
                    nxstate <= S0;

                elsif coin_in = "00" then -- En el caso de que no se meta nada, te quedas en S1 esperando otra moneda
                    nxstate <= S1;

                end if;

        end case;

    end process;

end Behavioral;
```

Cuando nos encontramos en el estado S1, si se pulsa el botón de reset devolvemos 1 euro ya que solo entramos en este estado si se ha introducido una moneda de 1 euro previamente.
Si se introduce otra moneda de 1 euro sale la lata y volvemos a S0.
Si se introducen 2 euros sale la lata, devolvemos 1 euro y volvemos a S0.
Si se introducen 5 euros sale la lata, devolvemos 4 euros y volvemos a S0.
Por último, si no se introduce nada nos mantenemos en S1.

Simulación:

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

entity tb_expendedora is
end tb_expendedora;

architecture Behavioral of tb_expendedora is
    constant clk_periodo : time := 10 ns;
    signal clk, rst : std_logic := '0';
    signal coin_in : std_logic_vector(1 downto 0);
    signal coin_out : std_logic_vector(1 downto 0);
    signal lata : std_logic;

    -- Componente a testear
    component m_expendedora is
        port (
            coin_in : in std_logic_vector(1 downto 0);
            rst, clk : in std_logic;
            coin_out : out std_logic_vector(1 downto 0);
            lata : out std_logic
        );
    end component;
```

Para comenzar la simulación primero declaramos las señales de cada entrada y salida y posteriormente declaramos el componente máquina expendedora.

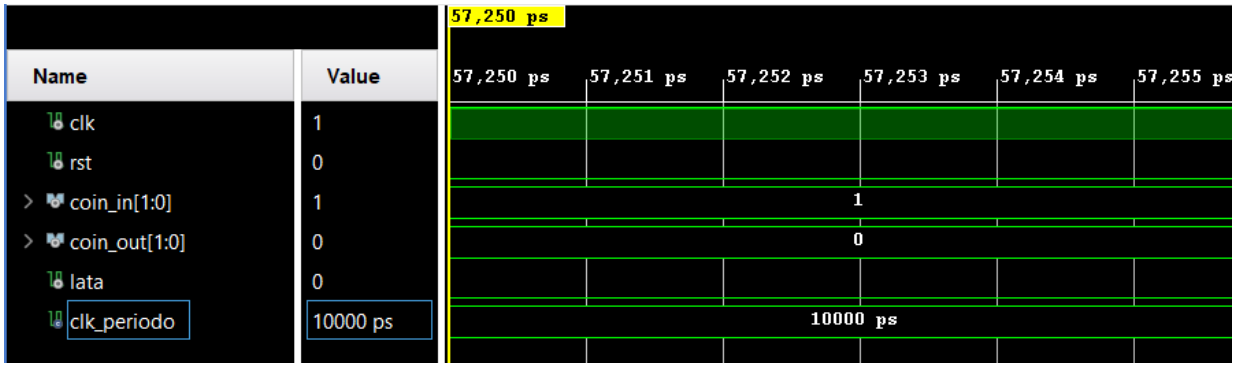
```
begin

   dut: m_expendedora port map (
        coin_in => coin_in,
        rst => rst,
        clk => clk,
        coin_out => coin_out,
        lata => lata
   );

   clk_process: process
   begin
        while now < 200 ns loop
            wait for clk_periodo / 2;
            clk <= not clk;
        end loop;
        wait;
   end process;
```

Asignamos las señales que hemos declarado al principio a las entradas y salidas del componente según corresponda y hacemos un process que vamos a utilizar para indicar cuanto tiempo se va a mantener el reloj funcionando durante la simulación.

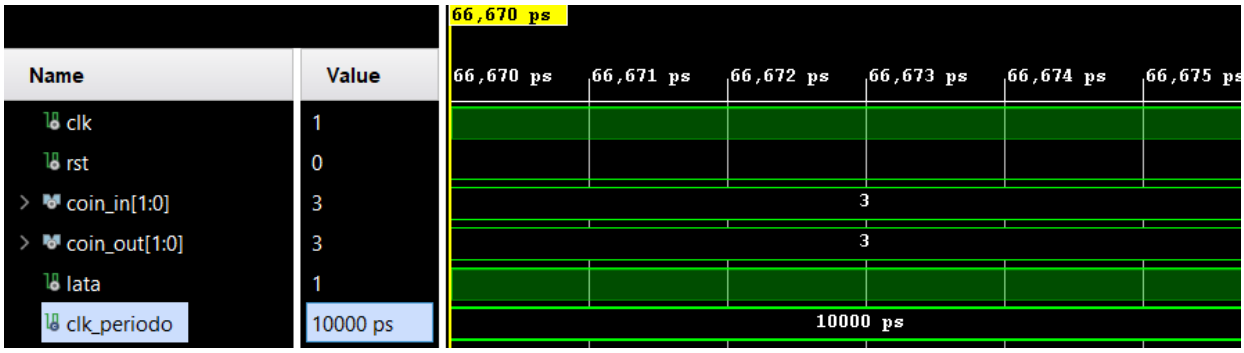
```
    test_process: process
    begin
        wait for clk_periodo / 4; -- Espera inicial

        -- No ingresas nada
        wait until rising_edge(clk);
        coin_in <= "00";

        -- Ingreso de 1$ despues de haber metido 1$
        wait until rising_edge(clk);
        coin_in <= "01";

        wait until rising_edge(clk);
        coin_in <= "01";

        -- Ingreso de 2$ despues de haber metido 1$
        wait until rising_edge(clk);
        coin_in <= "01";

        wait until rising_edge(clk);
        coin_in <= "10";

      -- Ingreso de 5$ despues de haber metido 1$
      wait until rising_edge(clk);
      coin_in <= "01";

      wait until rising_edge(clk);
      coin_in <= "11";

      -- No ingreso de moneda despues de haber metido 1$. Ingreso de 1$ tras
      -- comprobar que continúa la operación.

      wait until rising_edge(clk);
      coin_in <= "01";

      wait until rising_edge(clk);
      coin_in <= "00";

      wait until rising_edge(clk);
      coin_in <= "01";

        -- Pulso de reset despues de haber metido 1$

        wait until rising_edge(clk);
        coin_in <= "01";
        wait until rising_edge(clk);
        rst <= '1';
        wait for clk_periodo / 2;
        rst <= '0';
        wait for clk_periodo / 2;

        -- Ingresas 2$
        wait until rising_edge(clk);
        coin_in <= "10";

        -- Ingresas 5$
        wait until rising_edge(clk);
        coin_in <= "11";

        wait;
    end process;

end Behavioral;
```

En este último process lo que hacemos es asignar las entradas para comprobar que las salidas son correctas en base al funcionamiento definido cuando se realiza la simulación. Se testean varios de los casos posibles para comprobar que el funcionamiento es correcto.

Ejemplo de funcionamiento:



Aquí podemos observar que cuando la entrada es 1 no sale ni lata ni devolución.



Aquí podemos ver que si se meten 5 euros después del euro anterior sale la lata y se devuelven 4 euros.

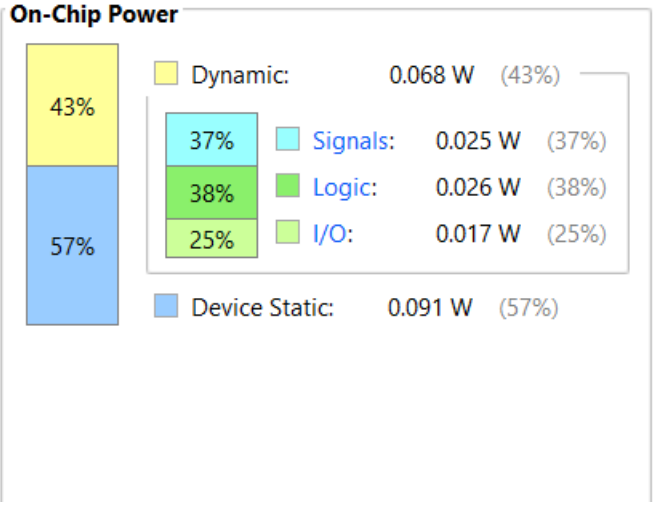
Análisis de tiempo, área y consumo

Consumo:

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.159 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25,7°C**
Thermal Margin:	59,3°C (12,9 W)
Effective ϑJA:	4,6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low



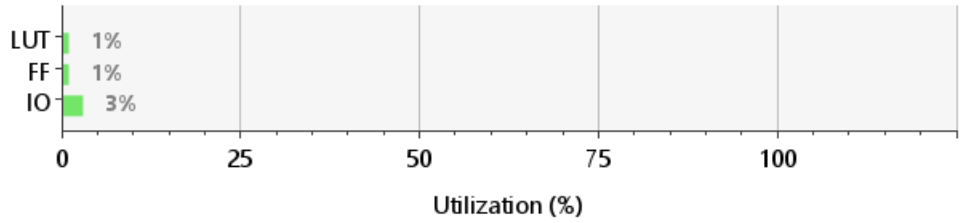
En el test de consumo podemos observar que, del consumo del chip, un 43% (0.068 W) es dinámico, dividido en 37% las señales (0.0025 W), 38% la lógica/componentes (0.0026 W) y un 25% los inputs y outputs (0.017 W). El otro 57% del consumo del chip es estático (0.091 W).

Sumando todo esto el consumo total del chip es de 0.159 W y la temperatura es de 25,7 grados.

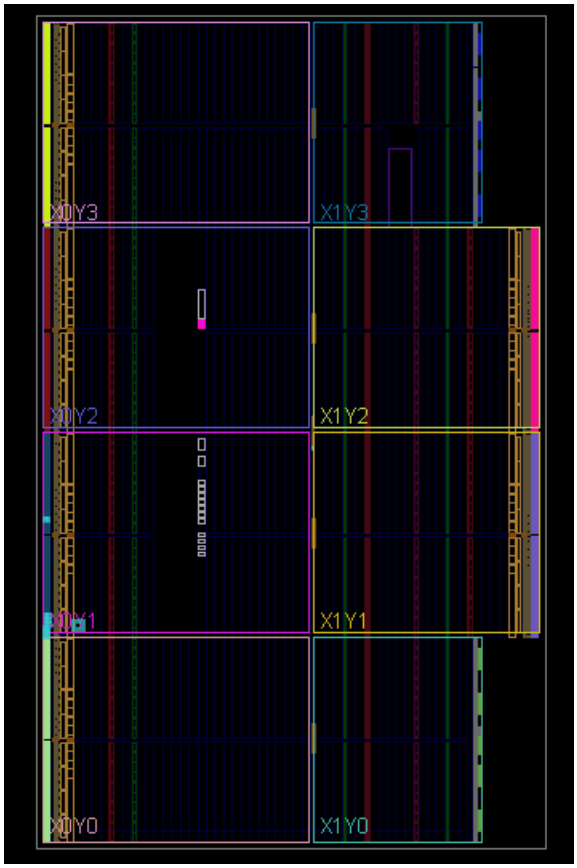
Área:

Name 1	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
N m_expendedora	8	5	4	8	7	1

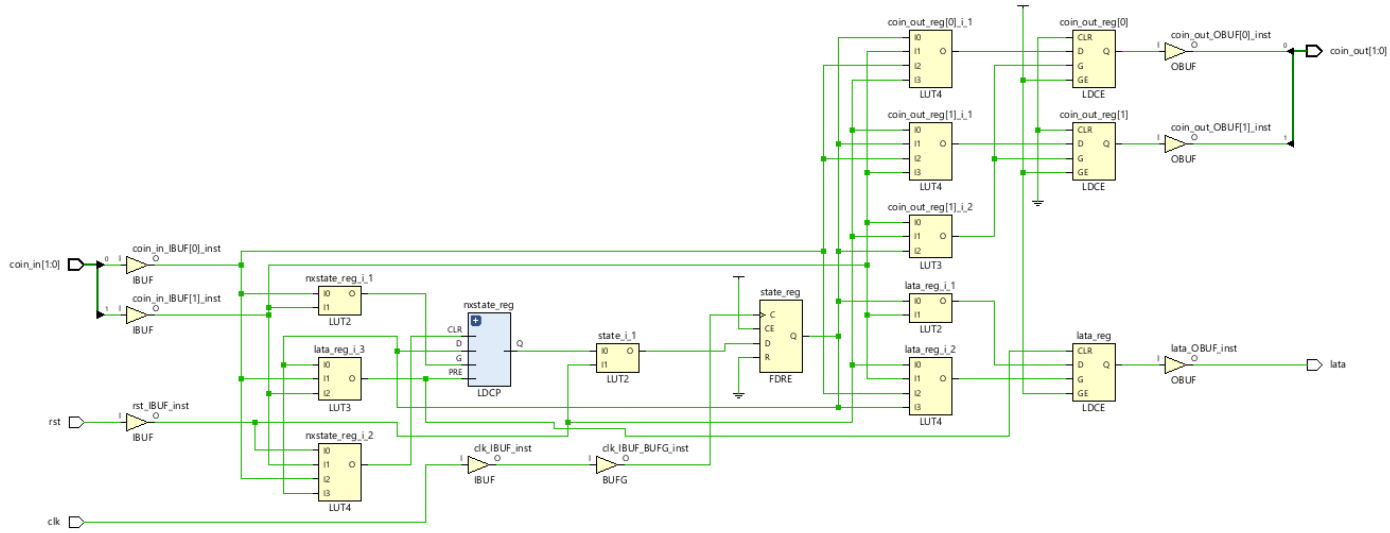
Resource	Utilization	Available	Utilization %
LUT	8	63400	0.01
FF	5	126800	0.00
IO	7	210	3.33



Device:



Esquemático:



Tiempo:

Setup

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	2	1	1	coin_out_reg[1]/G	coin_out[1]	5.085	3.411	1.674	∞				0.000
Path 2	∞	2	1	1	lata_reg/G	lata	5.031	3.206	1.825	∞				0.000
Path 3	∞	2	1	1	coin_out_reg[0]/G	coin_out[0]	4.937	3.407	1.530	∞				0.000
Path 4	∞	3	3	8	coin_in[1]	nxstate_reg/L7/D	3.725	1.255	2.470	∞	input port clock			0.000
Path 5	∞	2	2	6	coin_in[0]	coin_out_reg[1]/D	3.256	1.110	2.147	∞	input port clock			0.000
Path 6	∞	2	2	6	coin_in[0]	coin_out_reg[0]/D	3.209	1.138	2.072	∞	input port clock			0.000
Path 7	∞	2	2	8	coin_in[1]	lata_reg/CLR	3.100	1.131	1.969	∞	input port clock			0.000
Path 8	∞	2	2	8	coin_in[1]	lata_reg/D	2.683	1.126	1.557	∞	input port clock			0.000
Path 9	∞	2	1	6	rst	state_reg/D	2.581	1.129	1.452	∞	input port clock			0.000

Hold

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 10	∞	2	2	8	state_reg/C	nxstate_reg/L7/D	0.377	0.186	0.191	-∞				0.000
Path 11	∞	2	1	1	nxstate_reg/L7/G	state_reg/D	0.429	0.203	0.226	-∞				0.000
Path 12	∞	2	2	8	state_reg/C	lata_reg/CLR	0.633	0.186	0.447	-∞				0.000
Path 13	∞	2	2	8	state_reg/C	lata_reg/D	0.710	0.186	0.524	-∞				0.000
Path 14	∞	2	2	8	state_reg/C	coin_out_reg[0]/D	0.758	0.185	0.573	-∞				0.000
Path 15	∞	2	2	8	state_reg/C	coin_out_reg[1]/D	0.787	0.186	0.601	-∞				0.000
Path 16	∞	2	1	1	coin_out_reg[0]/G	coin_out[0]	1.666	1.383	0.283	-∞				0.000
Path 17	∞	2	1	1	lata_reg/G	lata	1.693	1.322	0.372	-∞				0.000
Path 18	∞	2	1	1	coin_out_reg[1]/G	coin_out[1]	1.711	1.386	0.325	-∞				0.000

Timing Check	Count ^1	Worst Severity
no_clock	15	High
unconstrained_internal_endpoints	6	High
no_input_delay	3	High
no_output_delay	3	High

Trabajo Opcional

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

entity m_expV2 is

    port(
        coin_in : in std_logic_vector(1 downto 0);
        rst, clk, reponedor : in std_logic;
        coin_out : out std_logic_vector(2 downto 0);
        lata : out std_logic
    );

end m_expV2;
```

Para el trabajo opcional en la entidad solo se ha añadido una entrada de reponedor para saber cuándo el reponedor ha puesto nuevo stock de latas.

```
architecture Behavioral of m_expV2 is

type state_type is (S0, S1);
signal state, nxstate : state_type;
signal inventario : integer range 0 to 3 := 3;
signal empty : std_logic := '0';
signal aux : integer := 0;

begin

    process(clk, rst) begin
        if(rising_edge(clk)) then
            aux <= aux + 1;
            if(rst = '1') then
                state <= S0;
            else
                state <= nxstate;
            end if;
         end if;
    end process;
```

En las señales hemos añadido el contador del inventario (incialmente 3 para facilidad de testeo, pero es posible cambiarlo), la señal de empty que saltará cuando la máquina este vacía, y la señal aux que utilizaremos para activar el segundo process cuando no hay cambio en el valor de la entrada coin_in (es decir, que metes dos veces seguidas la misma moneda cuando el valor es “10” o “11”). El valor de aux se incrementa en el primer process para estimular el segundo.

```
process(state, coin_in, reponedor, aux) begin

    coin_out <= "000";
    lata <= '0';

    case state is

        when S0 =>

            if(reponedor = '1') then

                inventario <= 3;
                empty <= '0';

            elsif empty = '1' or inventario = 0 then

                if(coin_in = "10") then
                    coin_out <= "100";

                elsif coin_in = "11" then
                    coin_out <= "101";

                elsif coin_in = "01" then

                    coin_out <= "001";

                else

                    coin_out <= "000";

                end if;
```

Hemos cambiado la asignación de monedas devueltas ya que al devolver la propia moneda que entra cuando no hay latas se han aumentado el número de casos de devoluciones posibles.

Nuevo índice de devoluciones:
000 = 0 euros
001 = 1 euro
010 = 3 euros
011 = 4 euros
100 = 2 euros
101 = 5 euros

Cuando entramos en el estado S0 y el reponedor está en 1, el inventario se resetea y se pone la señal de empty a 0.

Cuando empty es igual a 1 o el inventario es 0 (deberían estar las 2 en este estado) devuelve la misma moneda que entra ya que no quedan más latas que dispensar.

```
                nxstate <= S0;

            else

                case coin_in is

                    when "01" =>
                        nxstate <= S1;

                    when "10" =>
                        lata <= '1';
                        inventario <= inventario - 1;

                        if inventario = 0 then --No quedan latas
                           empty <= '1';

                        end if;

                        nxstate <= S0;

                    when "11" =>

                        lata <= '1';
                        coin_out <= "010";
                        inventario <= inventario - 1;

                        if inventario = 0 then --No quedan latas
                           empty <= '1';
                        end if;

                        nxstate <= S0;
```

Cuando entramos en S0 y todavía quedan latas el sistema es el mismo que en la parte obligatoria, el único cambio es que restamos en 1 el contador de inventario de latas cuando se dispensa una lata y se activa la señal empty cuando no hay más latas.

```
                when "00" =>

                    nxstate <= S0;

                when others =>

                    nxstate <= S0;

            end case;

        end if;

    when S1 =>

        if (rst = '1') then -- Devuelve 1$ si se pulsa reset (la persona al final no quiere comprar la lata).
            coin_out <= "001";
            nxstate <= S0;

        elsif coin_in = "01" then
            lata <= '1';
            inventario <= inventario - 1;

            if inventario = 0 then --No quedan latas
               empty <= '1';
            end if;

            nxstate <= S0;
```

En el estado S1 son los mismos cambios que en S0.

```
                elsif coin_in = "10" then
                    lata <= '1';
                    coin_out <= "001";
                    inventario <= inventario - 1;

                    if inventario = 0 then --No quedan latas
                       empty <= '1';
                    end if;

                    nxstate <= S0;

                elsif coin_in = "11" then
                    lata <= '1';
                    coin_out <= "011";
                    inventario <= inventario - 1;

                    if inventario = 0 then --No quedan latas
                       empty <= '1';
                    end if;

                    nxstate <= S0;

                elsif coin_in = "00" then -- En el caso de que no se meta nada, te quedas en S1 esperando otra moneda
                    nxstate <= S1;

                end if;

        end case;
    end process;
nd Behavioral;
```

Simulación:

El único cambio en el test bench del trabajo opcional ha sido el añadir de la nueva entrada "reponedor" en el componente de la máquina expendedora. El resto del código es exactamente igual. Por supuesto, se han añadido nuevas pruebas con la implementación del sistema de stock y reponedor, que se podrán ver en el archivo adjunto.

Análisis:

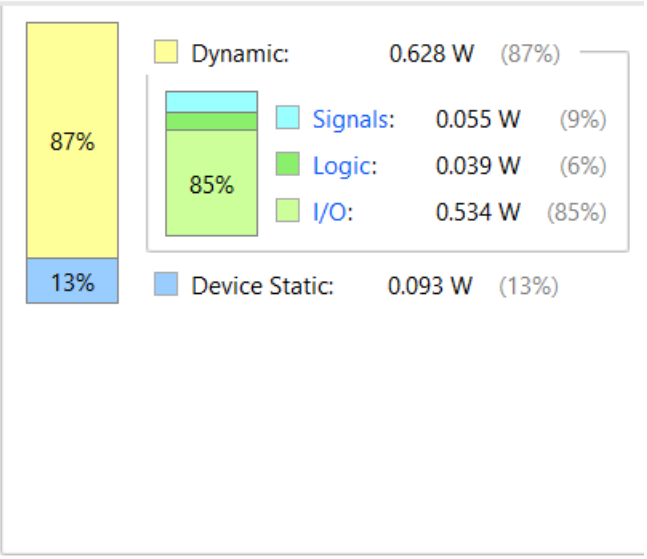
Consumo:

Summary

derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.721 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**28,3°C**
Thermal Margin:	56,7°C (12,3 W)
Effective ϑJA:	4,6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

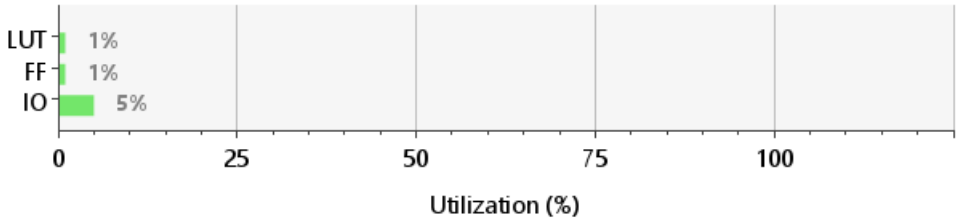
Launch Power Constraint Advisor to find and fix invalid switching activity



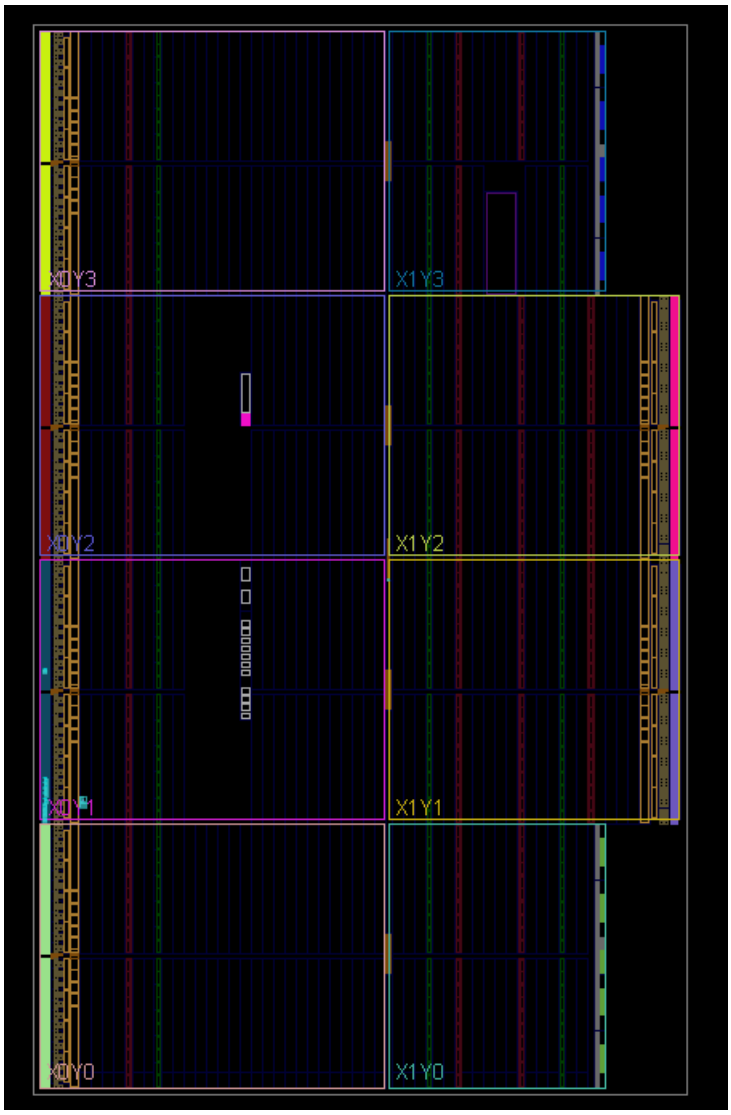
Área:

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
N m_expV2	13	5	4	13	10	1

Resource	Utilization	Available	Utilization %
LUT	13	63400	0.02
FF	5	126800	0.00
IO	10	210	4.76



Device:



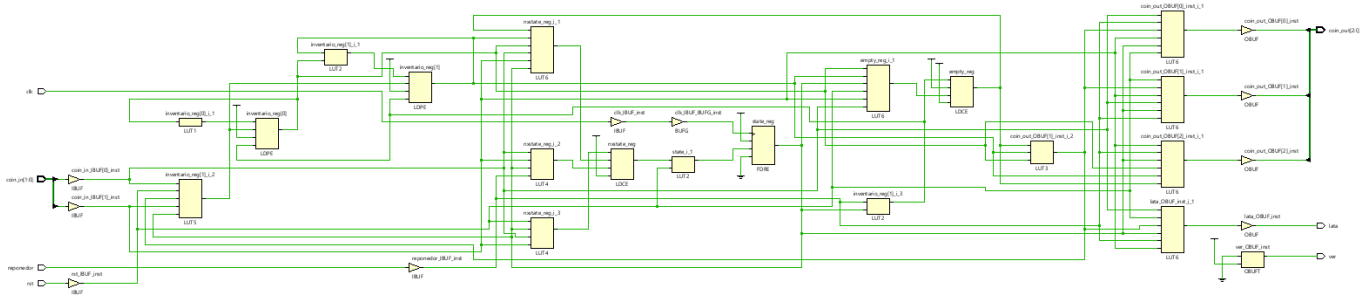
Tiempo:

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	3	2	9	coin_in[1]	coin_out[2]	7.382	3.777	3.605	∞	input port clock			0.000
Path 2	∞	3	2	8	coin_in[0]	lata	7.252	3.772	3.480	∞	input port clock			0.000
Path 3	∞	3	2	8	coin_in[0]	coin_out[1]	7.087	3.757	3.330	∞	input port clock			0.000
Path 4	∞	4	3	6	inventario_reg[0]/G	coin_out[0]	7.085	3.687	3.398	∞				0.000
Path 5	∞	2	2	9	coin_in[1]	nxstate_reg/CLR	3.249	1.131	2.117	∞	input port clock			0.000
Path 6	∞	2	2	6	reponedor	empty_reg/CLR	3.083	1.129	1.955	∞	input port clock			0.000
Path 7	∞	2	2	6	reponedor	inventario_reg[0]/PRE	3.078	1.129	1.949	∞	input port clock			0.000
Path 8	∞	2	2	6	reponedor	inventario_reg[1]/PRE	3.078	1.129	1.949	∞	input port clock			0.000
Path 9	∞	2	2	9	coin_in[1]	nxstate_reg/D	2.873	1.131	1.742	∞	input port clock			0.000
Path 10	∞	2	2	6	inventario_reg[0]/G	inventario_reg[1]/D	2.117	0.683	1.434	∞				0.000

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 11	∞	2	1	1	nxstate_reg/G	state_reg/D	0.419	0.203	0.216	-∞				0.000
Path 12	∞	2	2	10	state_reg/C	inventario_reg[0]/PRE	0.478	0.186	0.292	-∞				0.000
Path 13	∞	2	2	10	state_reg/C	inventario_reg[1]/PRE	0.478	0.186	0.292	-∞				0.000
Path 14	∞	2	2	10	state_reg/C	empty_reg/CLR	0.482	0.186	0.296	-∞				0.000
Path 15	∞	2	2	6	inventario_reg[0]/G	inventario_reg[0]/D	0.489	0.203	0.286	-∞				0.000
Path 16	∞	2	2	10	state_reg/C	nxstate_reg/D	0.518	0.186	0.332	-∞				0.000
Path 17	∞	2	2	10	state_reg/C	nxstate_reg/CLR	0.562	0.186	0.376	-∞				0.000
Path 18	∞	2	2	5	inventario_reg[1]/G	inventario_reg[1]/D	0.615	0.203	0.412	-∞				0.000
Path 19	∞	3	2	10	state_reg/C	coin_out[0]	1.862	1.352	0.510	-∞				0.000
Path 20	∞	3	2	6	inventario_reg[0]/G	coin_out[2]	1.908	1.365	0.543	-∞				0.000

Timing Check	Count	Worst Severity
no_clock	19	High
constant_clock	0	
pulse_width_clock	0	
unconstrained_internal_endpoints	8	High
no_input_delay	4	High
no_output_delay	4	High

Esquemático:



Conclusión

El trabajo realizado ha sido una máquina expendedora en VHDL en la que se pueden introducir distintos tipos de moneda para comprar una lata de refresco. Se han utilizado las entradas y salidas propuestas en el enunciado de la actividad y se ha hecho un reset síncrono.

En el test bench se comprueban las posibles combinaciones de entradas y las salidas que proporciona el código escrito y el análisis de consumo nos dice que tanto las temperaturas como el consumo no son demasiado altos.

Para la parte opcional hemos añadido el contador que se encarga de contabilizar las latas que quedan en stock, la señal de empty que nos indica que no quedan más latas que dispensar y la entrada reponedor para saber cuándo se reponen las latas.