Tecnología de Computadores

Trabajo Final de Laboratorio

UNIVERSIDAD NEBRIJA

Cristóbal García Fernández

UNIVERSIDAD NEBRIJA

1. Introducción

A lo largo de este primer cuatrimestre, conforme he ido avanzando en la signatura y en las practicas, he ido adquiriendo habilidades y experiencia programando en el lenguaje de diseño hardware *VHDL*. A modo de evaluación final, se me pide realizar un trabajo que pone a prueba mis habilidades.

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 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.

· La máquina siempre tiene latas disponibles, por lo que, para simplificar, no se considera la posibilidad de quedarse sin stock.

Como ampliación, se propone realizar 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 reinicia esta señal al introducir 3 nuevas latas en la máquina.

En este proyecto incluyo una descripción y los razonamientos que he ido aplicando para llegar al resultado junto con los códigos VHDL que he utilizado.

Para la realización del trabajo, me he ayudado de la suite de software *Vivado* para todo el ámbito de escribir los módulos, las simulaciones de prueba, etc.

2. Descripción del trabajo

Como he comentado antes, dada la descripción del trabajo que se me ha facilitado, he optado por comenzar haciendo un diagrama de estados que contemple todas las opciones del enunciado. De esta manera, se facilita considerablemente el trabajo ya que es más sencillo tener una visión general del problema.

En el enunciado no se especifica que la maquina tenga que ser de *Moore* o de *Mealy* por lo que, a la hora de plantear el diagrama, he considerado las dos opciones para ver en un futuro cual se ajusta mejor a las especificaciones.

Para mi diagrama final, es decir, el modelo con el que he trabajado, he optado por una máquina de Mealy. El resultado es el siguiente:

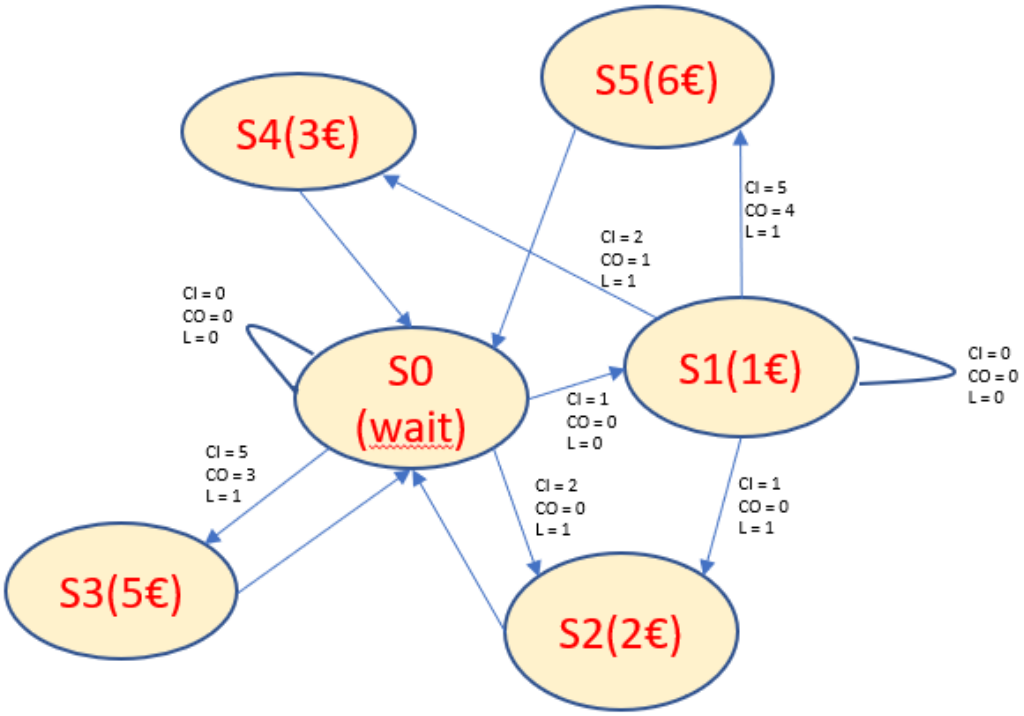


Imagen 1.

Tal y como se aprecia en la imagen 1, tenemos el estado inicial y de espera que es el estado 0. Partiendo de aquí, hay tres posibles opciones de monedas a introducir en la máquina. Si introducimos una moneda de 1€ estando en el estado 0, nos llevara al estado 1 (que es precisamente el estado 1€) sin devolver cambio y sin sacar una lata dado que la lata cuesta 2€. En cambio, si estando en el estado 0 introducimos una moneda de 2€, nos llevara al correspondiente estado 2, sacandonos una lata por la trampilla y sin devolvernos cambio. De la misma manera, si estando en el estado 0, introduciomos un billete de 5€, nos llevara al estado 3 (estado 5€) sacandonos una lata y un cambio de 3€ (5€ - 2€ que cuesta la lata).

Sin embargo, al encontrarnos en el estado de 1€, la maquina sigue a la espera de recibir dinero para poder vendernos la lata. Llegados a este puntos tenemos varias opciones que hemos de considerar. Podemos introducir otra moneda de 1€ (lo cual nos llevaria al estado 2€ devolviendonos asi una lata) o, podemos también introducir una moneda de 2€ (lo que nos llevaria al estado 3€ devolviendonos 1€ de cambio y la lata) o, como ultima opción, podriamos introducir un billete de 5€ lo cual conlleva que la maquina devuelve 4€ de cambio ((1€ + 5€) – 2€) y dispensa una lata.

Los estados 2, 3, 4 y 5, conducen directamente de vuelta al estado de espera dado que son aquellos estados en los que acceder a ellos, conlleva que el cliente ya ha recibido una lata y la maquina pasa de nuevo al estado de espera pendiente de recibir mas dinero. Esto ultimo ha sido una decisión personal de diseño. Como he mencionado antes, también probé distintas opciones. Una de ellas, un modelo de Moore. Para este caso, el diagrama de estados es prácticamente idéntico al que he utilizado yo, con la única diferencia de que las salidas COIN_OUT y LATA se contemplan en el propio estado. Es decir, que dependen unicamente del estado. Otro diseño de Mealy que contemplé fue uno en el que únicamente habia 2 estados.

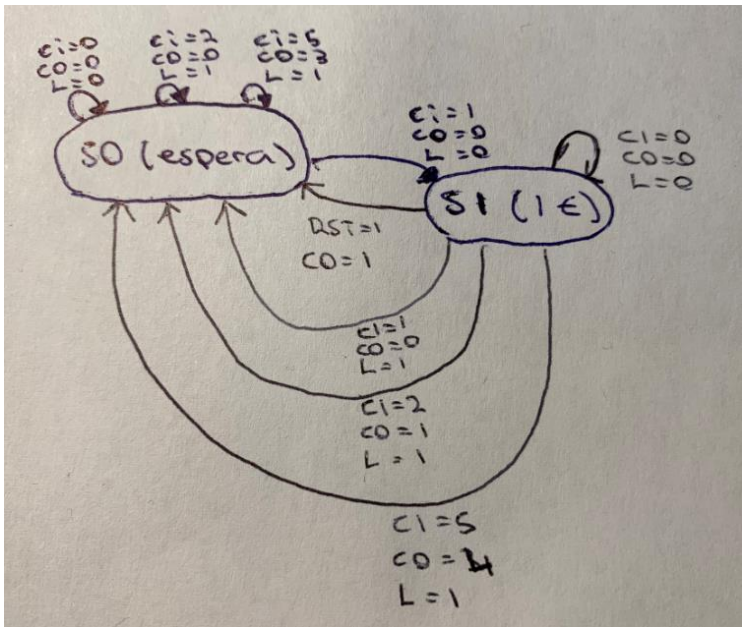


Imagen 2.

Este caso es equivalente ya que el estado de espera S0, proporciona una salida de Mealy para devolver una lata en los caso en los que se introducen 2 o 5€, y vuelve al estado de espera. Este sería el equivalente a lo que he mencionado antes de el hecho de que los estados 2, 3, 4 y 5 vuelvan por defecto al estado de espera.

Cabe destacar también la codificación que he usado para las entradas y las salidas.

· La entrada COIN_IN toma 4 distintos valores por lo que para su implementación, he optado por una entrada de 2 bits. Estos son sus correspondientes valores:

"00" = 0€

"01" = 1€

"10" = 2€

"11" = 5€

· La salida COIN_OUT toma tambien 4 distintos valores por lo que la he declarado en mi código como una salida de 2 bits. Estos son los valores que pude tomar:

"00" = 0€

"01" = 1€

"10" = 3€

"11" = 4€

· Por ultimo, la salida LATA, que cuando toma el valor '1', se activa la trampilla para que el usuario pueda recoger la lata.

3. Códigos VHDL

Una vez completo el proceso de diseño, es mas sencillo estructurar el programa que se realiza.

La entidad del código principal se asemeja mucho a aquella de una maquina de estados ordinaria. Entradas, salidas, entrada de reloj y reset.

```
entity FMS is
    Port (CLK, RST : in STD_LOGIC;
          COIN_IN : in STD_LOGIC_VECTOR(1 downto 0);
          LATA : out STD_LOGIC;
          COIN_OUT : out STD_LOGIC_VECTOR(1 downto 0)
    );
end FMS;
```

Imagen 3.

Tal y como se aprecia en la imagen 3, la maquina que he diseñado consta de 3 entradas; CLK, RST y COIN_IN. Siendo esta ultima la variable que transiciona entre estados ya que indica el tipo de moneda introducido.

Seguido de la entidad, se declara la arquitectura, en este caso de tipo *Behavioral* dado que alberga instrucciones de tipo condicional y distintos *Process*. Antes del *begin*, se declaran las señales *CS* y *NS* que son las que representan el estado actual y es estado siguiente respectivamente, y son del tipo que se ve declarado en la imagen 4. Es decir, S0, S1, S2, S3, S4 y S5. En esencia, son las señales que simulan las transiciones entre los estados.

```
architecture Behavioral of FMS is
    type state_type is (S0, S1, S2, S3, S4, S5); signal CS, NS : state_type;
```

Imagen 4.

Seguido de la declaración de señales, encontramos el *begin* de la arquitectura en el que he descrito el primer *Process*.

```
process (CLK) begin
    if(rising_edge(CLK)) then
        if(RST = '1') then
            CS <= S0;
        else
            CS <= NS;
        end if;
    end if;
end process;
```

Imagen 5.

El código que se muestra en la imagen 5, muestra el proceso secuencial síncrono que he decidido implementar. Este se encarga de reiniciar los estados al estado de espera S0 cuando la señal *RST* se pone a '1' en un flanco de subida del reloj o de actualizar el estado actual al estado siguiente. Todo esto, porque he decidido hacer una maquina de estados síncrona.

Una vez finalizado este *Process*, es hora de implementar el proceso combinacional para controlar los cambios de estado y la salida.

```
process (CS, COIN_IN) begin
    case CS is
        when S0 =>
            if(COIN_IN = "00") then
                COIN_OUT <= "00"; LATA <= '0'; NS <= S0;
            elsif(COIN_IN = "01") then
                COIN_OUT <= "00"; LATA <= '0'; NS <= S1;
            elsif(COIN_IN = "10") then
                COIN_OUT <= "00"; LATA <= '1'; NS <= S2;
            elsif(COIN_IN = "11") then
                COIN_OUT <= "10"; LATA <= '1'; NS <= S3;
            end if;
        when S1 =>
```

Imagen 6.

Tal y como se aprecia en la imagen 6, la lista de sensibilidad del proceso contiene la señal CS y la señal COIN_IN que es la que va a indicar a qué estado se debe transicionar. En este proceso, se contemplan las distintas posibilidades que he señalado antes para cada estado. Para los estados S2 - S5, se redirecciona directamente al estado S0 con la instrucción *NS <= S0;* tal y como se refleja en el diagrama de estados.

El correcto funcionamiento de la maquina de estados lo he ido comprobando a lo largo del transcurso del trabajo. He implementado un *testbench*, al cual le he ido modificando los valores de las entradas para comprobar el correcto funcionamiento. En este, se crea una entidad vacía, se define un componente y se crean unas señales para poder instanciarlo. (imagen 7)

```
architecture Behavioral of Test is
    component FMS is
        Port (CLK, RST : in STD_LOGIC;
              COIN_IN : in STD_LOGIC_VECTOR(1 downto 0);
              LATA : out STD_LOGIC;
              COIN_OUT : out STD_LOGIC_VECTOR(1 downto 0)
        );
    end component;
    signal CLK, RST, LATA : STD_LOGIC;
    signal COIN_IN, COIN_OUT : STD_LOGIC_VECTOR(1 downto 0);
begin
    DUT : FMS port map(CLK => CLK, RST => RST, COIN_IN => COIN_IN, COIN_OUT => COIN_OUT, LATA => LATA);

    process begin
```

Imagen 7.

Una vez hecho esto, se definen en varios procesos, el comportamiento que van a tener las distintas entradas. Como decía antes, este aspecto es el que he ido modificando para contemplar los distintos casos que se aprecian en el diagrama. He aquí un ejemplo.

```
  process begin
      COIN_IN <= "01"; wait for 10 ns;
      COIN_IN <= "10"; wait for 10 ns;
      COIN_IN <= "11"; wait for 10 ns;
      COIN_IN <= "10"; wait for 10 ns;
      COIN_IN <= "01"; wait;
  end process;

  process begin
      RST <= '1'; wait for 50 ns;
      RST <= '0'; wait for 50 ns;
  end process;
```

Imagen 8.

En la siguiente imagen se puede ver como la simulación del testbench muestra los valores correspondientes. COIN_IN = 01, COIN_OUT = 00 y LATA a 0 en ese preciso instante.

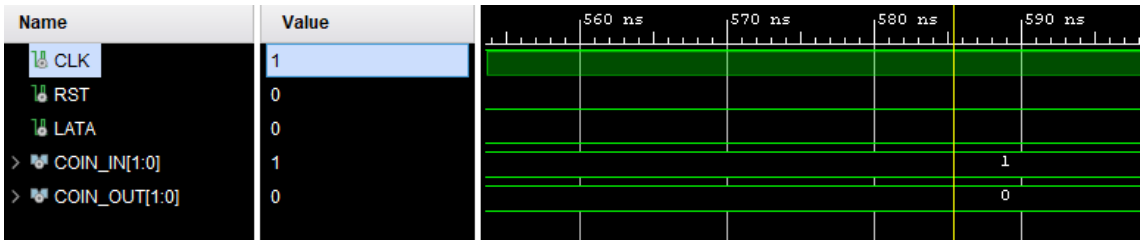


Imagen 9.

Tarea de ampliación

Para la tarea de ampliación se pedía implementar un sistema de cuenta que active una señal EMPTY cuando se haya agotado el número de latas de bebida disponibles en la máquina, suponiendo que se carga la maquina expendedora con 3 latas.

Para empezar, he creado una señal *EMPTY* de tipo *std_logic* y una señal *CUENTA* de tipo *Integer* con valor 3, la cual he usado como contador.

```
architecture Behavioral of FMS is
    type state_type is (S0, S1, S2, S3, S4, S5); signal CS, NS : state_type;
    signal CUENTA : integer := 3;
    signal EMPTY :  STD_LOGIC;
begin
```

Imagen 10.

En un proceso nuevo, controlo que la señal *EMPTY* se active cuando *CUENTA* sea < 1 ya que eso querrá decir que se ha agotado el stock.

```
process (CUENTA) begin
    if(CUENTA < 1) then
        EMPTY <= '1';
    end if;
end process;
```

Imagen 11.

De manera simultánea, en el proceso combinacional, para aquellos casos en los que se activa la señal *LATA*, es decir, aquellos casos en los que el usuario introduce la cantidad de dinero requerida y la maquina dispensa una lata, incluyo una línea de código en la que se le resta una unidad a la señal *CUENTA.* Vease en la imagen 12.

```
when S0 =>
    if(COIN_IN = "00") then
        COIN_OUT <= "00"; LATA <= '0'; NS <= S0;
    elsif(COIN_IN = "01") then
        COIN_OUT <= "00"; LATA <= '0'; NS <= S1;
    elsif(COIN_IN = "10") then
        COIN_OUT <= "00"; LATA <= '1'; NS <= S2; CUENTA <= CUENTA -1;
    elsif(COIN_IN = "11") then
        COIN_OUT <= "10"; LATA <= '1'; NS <= S3; CUENTA <= CUENTA -1;
    end if;
when S1 =>
    if(COIN_IN = "00") then
        COIN_OUT <= "00"; LATA <= '0'; NS <= S1;
    elsif(COIN_IN = "01") then
        COIN_OUT <= "00"; LATA <= '1'; NS <= S2; CUENTA <= CUENTA -1;
    elsif(COIN_IN = "10") then
        COIN_OUT <= "01"; LATA <= '1'; NS <= S4; CUENTA <= CUENTA -1;
    elsif(COIN_IN = "11") then
        COIN_OUT <= "11"; LATA <= '1'; NS <= S5; CUENTA <= CUENTA -1;
    end if;
when S2 =>
```

Imagen 12.

En la siguiente imagen se muestra, en una simulación, la correcta sinergia entre las nuevas señales *EMPTY* y *CUENTA* (imagen 13).

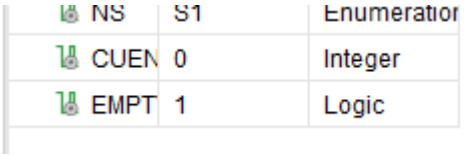


Imagen 13.

La imagen anterior es la simulación de uno de los *testbench* que he realizado (imagen 8) y que en el mismo simulo la introducción de 3 monedas de cantidad superior a 2€. En consecuencia, la maquina dispensa 3 correspondientes latas, la señal *CUENTA* se reduce a 0 y se activa la señal *EMPTY*.

En caso de querer contemplar el caso en el que el usuario pida una lata pero se hayan agotado, teniendo en cuenta mi implementación, condicionaría el proceso secuencial a la señal *CUENTA*. Es decir, dado que dicha señal indica el numero de latas restantes, pondría una condición *if* al principio del proceso para que deje continuar en caso de haber latas disponibles. De esta manera, si hubiese una lata restante (por ejemplo), entraría en la condición y se ejecutaría esa parte del código, la señal *CUENTA*, se actualizaría (*CUENTA* = 0), y ya no se cumpliría más la condición. Dicho de otra forma, la señal *EMPTY* indica si se puede o no sacar mas latas.

4. Evaluación de parámetros físicos del diseño

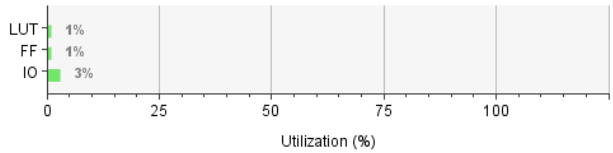
La evaluación de parámetros nos permite verificar que el diseño cumple con las especificaciones definidas. Una vez completa la síntesis, podemos pasar a la implementación y a la evaluación de parámetros.

Análisis de área

En este análisis, se contabiliza el numero de recursos de la *FPGA* utilizados por el diseño. Se determina el número de FF's, LUTs (look-up-tables) y block RAMs. El resultado es el siguiente:

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	Block RAM Tile (135)	Bonded IPADs (2)	BUFIO (24)
FMS	6	9	4	6	9	7	1

Resource	Utilization	Available	Utilization %
LUT	6	63400	0.01
FF	9	126800	0.01
IO	7	210	3.33



Ref Name	Used	Functional Category
LDCE	6	Flop & Latch
LUT2	5	LUT
IBUF	4	IO
OBUF	3	IO
FDRE	3	Flop & Latch
LUT5	2	LUT
LUT4	2	LUT
LUT3	2	LUT
BUFG	1	Clock

Imagen 14.

En las imágenes se aprecia el porcentaje de los recursos usados con respecto de los que hay disponibles.

Análisis de consumo y temperatura

Podemos obtener 3 valores de consumo con la herramienta de software de análisis. La siguiente imagen (imagen 14) contiene un resumen de todos los valores.

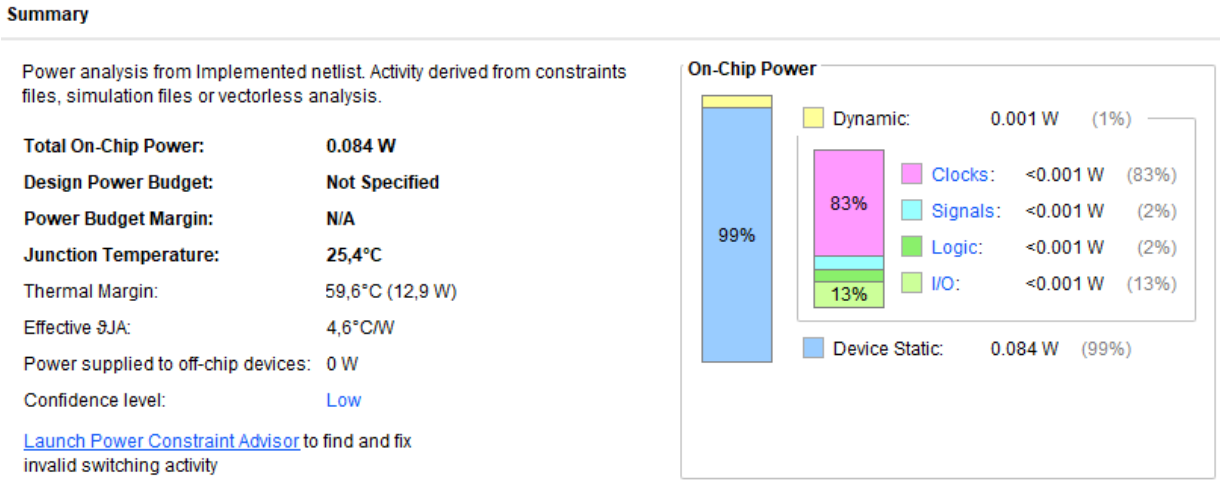


Imagen 15.

Si nos fijamos en la parte izquierda de la imagen 14, se muestra en la grafica el consumo estático del dispositivo, que hace referencia al consumo inherente de la FPGA debido a los transistores por el hecho de estar alimentados. Podemos observar también el consumo dinámico, que es el que depende de los valores de tensión, de la frecuencia y de los recursos activos en cada momento. Y por ultimo, el consumo total "on-chip" que es la suma de ambos.

En la imagen también encontramos parámetros relacionados con la temperatura. Por una parte tenemos la resistencia térmica efectiva cuyo valor es de 4.6ºC/W. Este valor es un coeficiente que define como de bien se disipa la potencia termica del chip de silicio de la FPGA hacia el ambiente. Este valor a su vez se utiliza para calcular la temperatura en la unión, cuyo valor en este caso es de 25,4ºC. Podriamos haber calculado esta temperatura haceiendo uso de la ecuacion:

Temperatura en la unión *= Temperatura ambiente +*
(Consumo total "on-chip" · Resistencia térmica efectiva)

Comprobamos que el resultado concuerda con el resultado de la implementación de Vivado.

Temperatura en la unión = 25.0ºC + (0.084W * 4.6ºC/W)

Temperatura en la unión = 25.3864ºC → 25.4ºC

5. Comentarios finales

Considero que este proyecto refleja bien mi progreso a lo largo del transcurso de la asignatura, y estoy muy satisfecho tanto con el resultado del trabajo final, como con mi progreso en general.

Quisiera agradecer también a mi profesor Luis Alberto Aranda por la resolución de pequeñas dudas y aclaraciones que han sido de utilidad para el proyecto dado que haberlo hecho de manera individual ha supuesto un reto en algunas ocasiones.