# UNIVERSIDAD POLITÉCTICA DE MADRID

## ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y DISEÑO INDUSTRIAL

**TRABAJO DE VHDL:**

MÁQUINA DESPACHADORA DE REFRESCOS

**MATERIA:**

SISTEMAS ELÉCTRÓNICOS DIGITALES

**INTEGRANTES:**


- GUTIÉRREZ LÓPEZ JOSUE
- OCARANZA MORALES JENNIFER
- TELLEZ QUINTANA RICARDO


# Índice

# INTRODUCCIÓN

Durante este proyecto se va a diseñar una máquina expendedora de refrescos con las siguientes características:

Admite monedas de 10c, 20c, 50c y 1€. Sólo admite el importe exacto, de forma que si introducimos dinero de más da un error y "devuelve" todo el dinero. Cuando se llega al importado exacto del refresco (1€) se activará una señal para dar el producto. Como entradas tendrán señales indicadoras de la moneda, señales indicadoras de producto y como salidas la señal de error y la de producto.

Implementaciones:

Uno de los aspectos más destacables es la capacidad de operar en dos modos diferentes mediante un interruptor (sw_c). En el modo de cambio (sw_c = '1'), la máquina permite ingresar una cantidad superior al precio exacto del producto, calcula el cambio correspondiente, lo muestra en el display de 7 segmentos y posteriormente entrega el producto. Por otro lado, en el modo sin cambio (sw_c = '0'), la máquina no permite excedentes y solo acumula dinero hasta alcanzar el importe exacto, mostrando el monto acumulado en el display y emitiendo una señal de error si se intenta introducir más dinero del necesario. Esta funcionalidad proporciona flexibilidad y mejora la experiencia del usuario al adaptarse a distintas situaciones.

Además, la máquina permite seleccionar entre tres diferentes precios de refresco, lo que amplía las opciones disponibles para los usuarios. Cada precio está asociado a un producto específico, identificado mediante caracteres en el display de 7 segmentos: d para el producto 1, E para el producto 2 y F para el producto 3. Esta información se muestra junto con el precio correspondiente, permitiendo al usuario visualizar de manera clara el producto seleccionado y su costo.

El diseño incluye un controlador lógico encargado de gestionar todas las señales de entrada y salida del sistema. Este controlador valida las monedas ingresadas, determina si se debe devolver cambio y activa la señal para la entrega del producto una vez que se alcanza el importe necesario. En caso de errores, como introducir dinero de más en el modo sin cambio, el sistema genera una señal de error y devuelve automáticamente todo el dinero ingresado.

# DESARROLLO

Para el diseño de nuestra máquina expendedora de refrescos, se optó por implementar una máquina de estados finitos (FSM) debido a la naturaleza secuencial de las operaciones que debe realizar. La FSM permite modelar de manera eficiente los diferentes pasos necesarios para el correcto funcionamiento del sistema, garantizando una transición clara entre las etapas según las entradas y condiciones establecidas. En este proyecto, cada estado de la FSM representa una etapa clave del funcionamiento de la máquina.

## MÁQUINA DE ESTADOS PARA CONTROLADOR

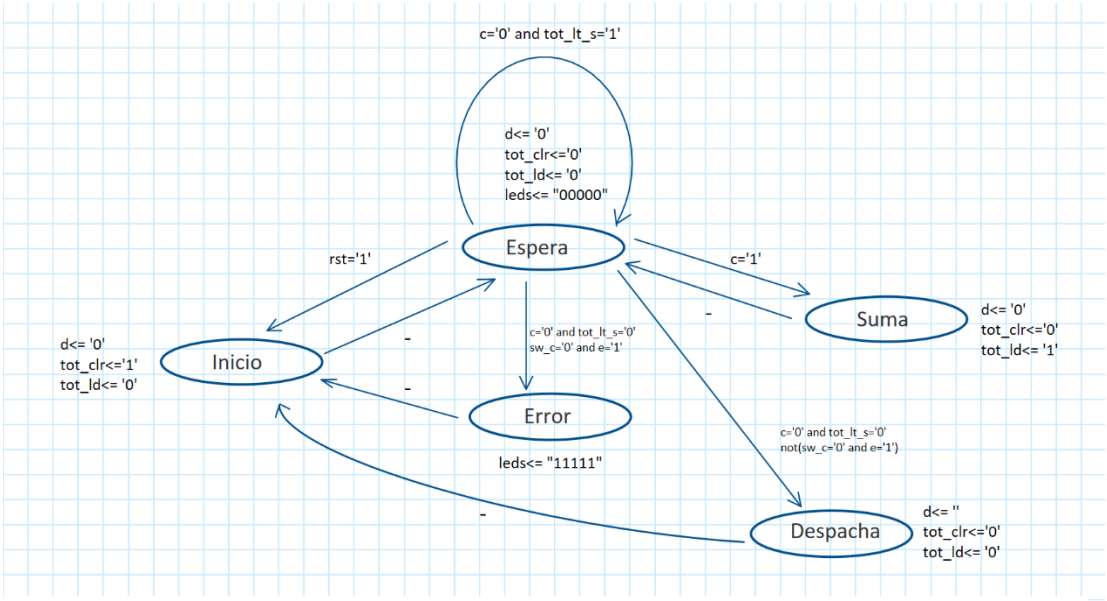


*Figura 1. Máquina de estados para el controlador de la Maquina de refrescos despachadora.*

El controlador está implementado con una máquina de estados finitos (FSM) con cinco estados: Inicio, Espera, Suma, Despacha, y Error.

**Estado Inicio:** Este es el estado en el que arranca la máquina.

- La señal tot_clr se activa para reiniciar el acumulado de dinero.
- La salida tot_ld se desactiva, no se cargan las monedas que tenga la máquina.
- La señal de entrega d también se desactiva (no despacha).

Transiciones:

- Después del estado Inicio, hay un salto incondicional al estado Espera.

**Estado Espera:** En este estado, la maquina permanece atento a las acciones del usuario.

- Las señales tot_clr y tot_ld permanecen desactivadas.
- Los leds muestran "00000", indicando que no hay errores.

Transiciones:

- Si rst = '1', el sistema regresa al estado Inicio.
- Si se detecta una moneda válida (c = '1'), la maquina pasa al estado Suma para registrar el monto ingresado.
- Si el importe acumulado no alcanza el precio del producto (c = '0' y tot_lt_s = '1'), la máquina permanece en el estado Espera.
- Si el importe acumulado no alcanza el precio del producto (c = '0' y tot_lt_s = '1') y si sw_c = '0' (sin cambio) y hay un error (e = '1'), el sistema transita al estado Error. En caso contrario, transita al estado Despacha.

**Estado Suma:** En este estado, la máquina carga el dinero ingresado.

- La señal tot_ld se activa para permitir la carga del acumulador de monedas.
- La señal d permanece desactivada (no despacha).

Transiciones:

- Una vez registrado el monto, la máquina regresa al estado Espera.

**Estado Despacha:** En este estado se entrega el producto.

- La señal d se activa para permitir la entrega del producto (despacha).
- Las señales tot_clr y tot_ld permanecen desactivadas.

Transiciones:

- Después de despachar, la máquina regresa al estado Inicio.

**Estado Error:** En este estado se activa la alarma de error, por ingresar más dinero del necesario en el modo sin cambio (sw_c = '0').

- Los leds muestran "11111" para indicar un error.

Transiciones:

- Después de indicar el error, la máquina regresa al estado Inicio.

## BLOQUE CONTROLADOR

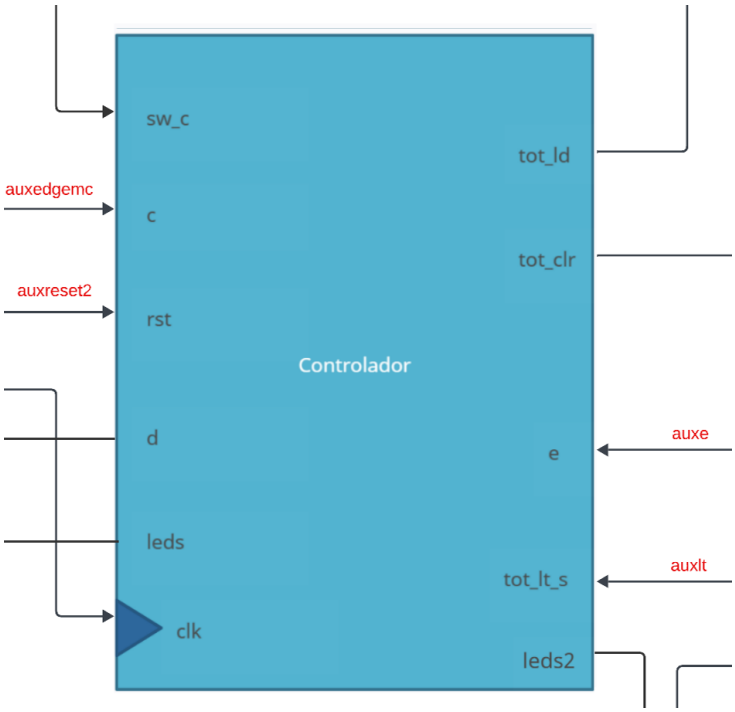


*Figura 2. Entradas y salidas para bloque Controlador.*

Como ya se describió anteriormente, Controlador es el núcleo de la maquina despachadora de refrescos.

# SIMULACIÓN CONTROLADOR

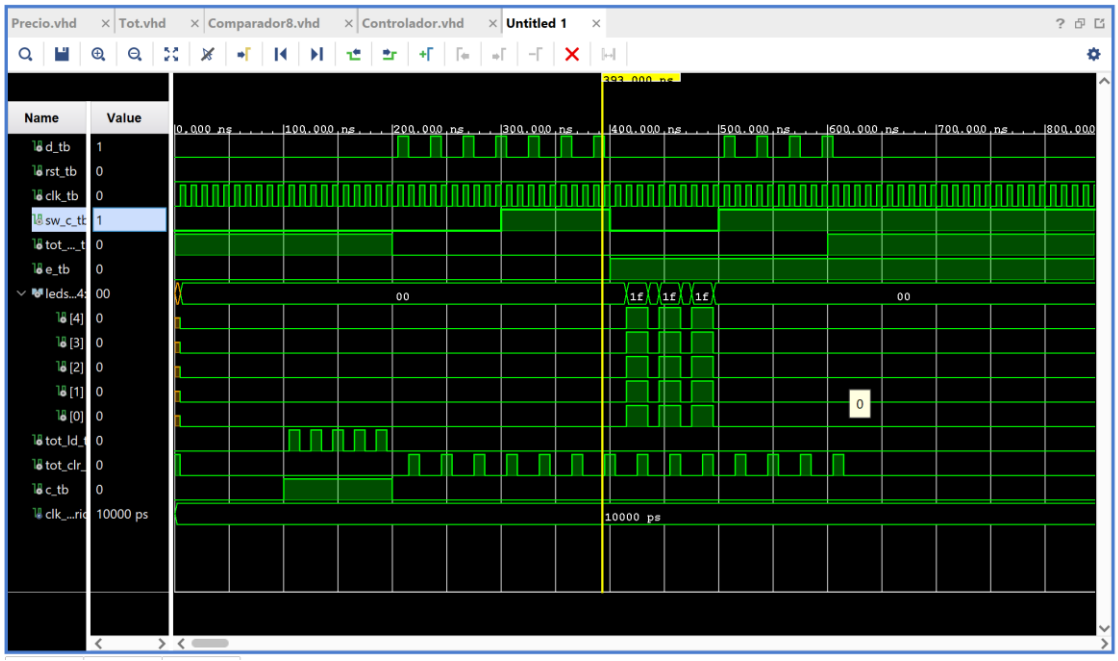


*Figura 3. Simulación para bloque Controlador.*

En la simulación se puede apreciar que la máquina responde adecuadamente a las condiciones y transiciones que se definieron en el diseño. Inicialmente, el sistema comienza en el estado de espera (salto incondicional de inicio a espera). Cuando se introduce una señal válida, como c = '1', la maquina pasa al estado de suma, donde registra correctamente la acumulación del dinero ingresado.

Dependiendo de la configuración del interruptor sw_c, el sistema puede pasar al estado de despacha, donde se activa la señal de entrega del producto, o al estado de error si se excede el importe en modo sin cambio. Durante la simulación, las señales de salida, como los leds reflejan los comportamientos esperados en cada estado, confirmando que el diseño cumple con los requisitos. Esto demuestra que la máquina está funcionando según lo planeado.

## BLOQUE SINCRONIZADOR

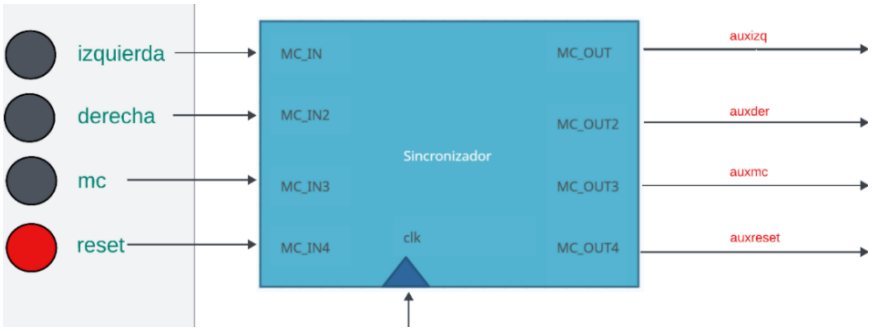


*Figura 4. Entradas y salidas para bloque Sincronizador.*

El bloque Sincronizador es esencial en el diseño, ya que se encarga de estabilizar las señales de entrada provenientes de los botones físicos para garantizar que sean interpretadas de manera correcta y sin errores por los demás bloques del sistema. Cada entrada (MC_IN, MC_IN2, MC_IN3, MC_IN4) representa una señal asociada a un botón, como izquierda, derecha, carga de moneda o reset. Estas señales, al ser generadas por pulsadores, pueden presentar rebotes eléctricos (bouncing), lo que provoca fluctuaciones indeseadas.

El sincronizador emplea registros de desplazamiento (shift registers) controlados por el reloj (clk) para filtrar estas fluctuaciones. Cuando se detecta un flanco ascendente del reloj, el sistema desplaza el valor de la señal de entrada y almacena su estado estable en un registro. Una vez que la señal ha pasado por este proceso, se genera una salida sincronizada (MC_OUT, MC_OUT2, MC_OUT3, MC_OUT4) que representa el estado limpio y estable de la señal original.

# SIMULACIÓN SINCRONIZADOR

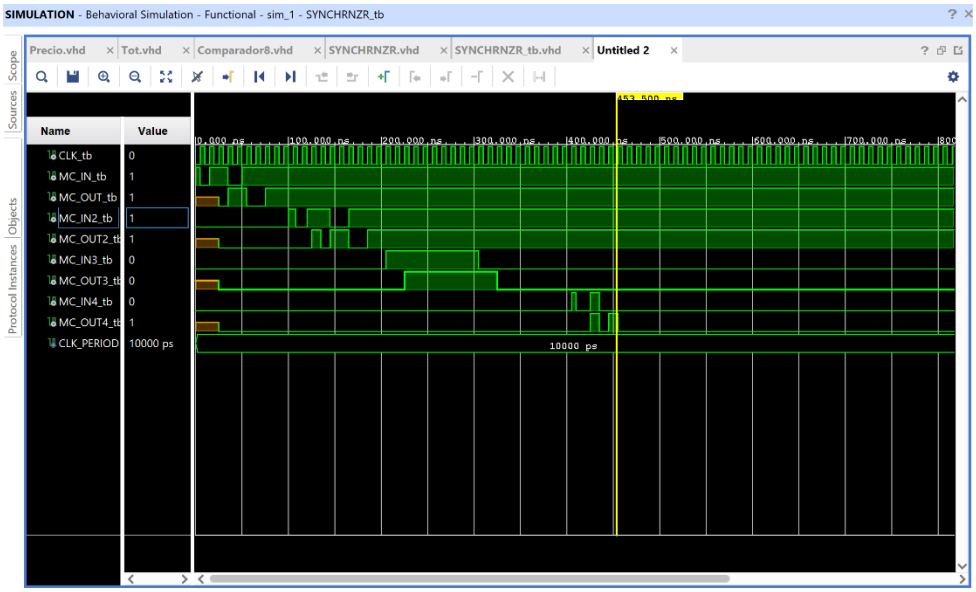


*Figura 5. Simulación para bloque Sincronizador.*

Durante la simulación, las entradas fluctúan entre valores altos y bajos en intervalos de tiempo cortos, simulando un comportamiento típico de rebote.

El sincronizador procesa estas señales utilizando registros de desplazamiento controlados por una señal de reloj (clk) . A medida que las entradas son capturadas, se observa cómo las salidas correspondientes (MC_OUT, MC_OUT2, MC_OUT3, MC_OUT4) permanecen estables y no reflejan las fluctuaciones rápidas, mostrando únicamente los valores válidos y estabilizados después de ser procesados por el sistema. Este comportamiento es especialmente evidente en las señales con patrones de rebote más marcados, como MC_IN y MC_IN2, donde las entradas cambian rápidamente, pero las salidas se actualizan de manera limpia y confiable solo cuando la señal es consistente durante varios ciclos de reloj.

## BLOQUE EDGEDETECTER

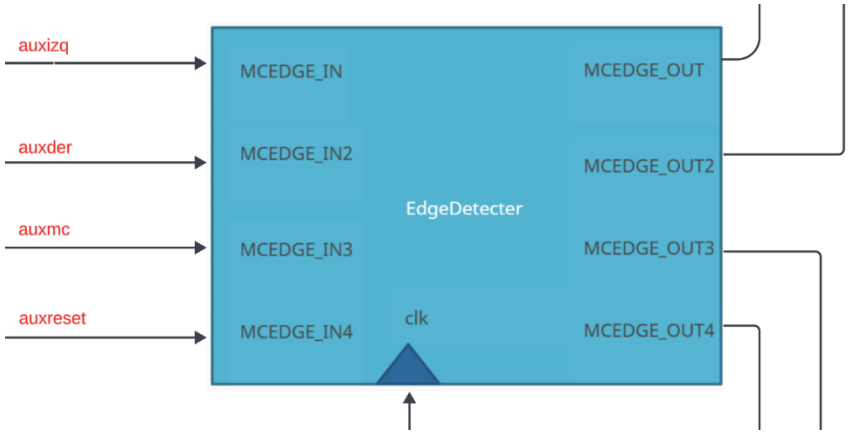


*Figura 6. Entradas y salidas para bloque EdgeDetecter.*

El bloque EdgeDetecter se encarga de detectar cuando una señal de entrada pasa de un valor bajo (0) a uno alto (1), lo que conocemos como un flanco ascendente. Cuando esto sucede, el bloque genera un pulso corto en la salida correspondiente. Para lograrlo, utiliza registros de desplazamiento que almacenan el estado actual y el estado anterior de cada señal, todo esto sincronizado con el reloj (clk). Esto permite recordar cómo estaba la señal antes y comparar si ocurrió ese cambio de 0 a 1.

El bloque asegura que las salidas solo se activen cuando se detecta este tipo de cambio específico. Básicamente, si el registro identifica un patrón claro de cambio (en este caso el paso de "100"), entonces activa la salida. Con esto se puede ignorar ruidos o cambios no relevantes en las señales, enfocándose únicamente en los eventos importantes.

# SIMULACIÓN EDGEDETECTER

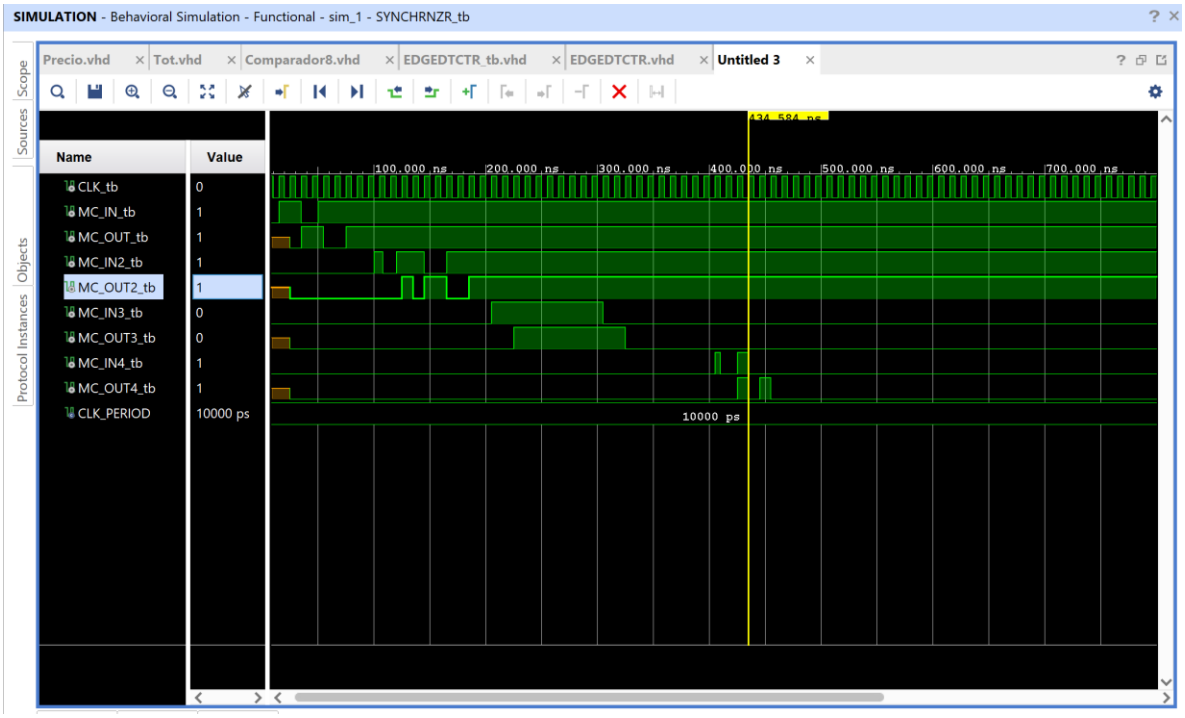


*Figura 7. Simulación para bloque EdgeDetecter.*

En la simulación del bloque EdgeDetecter, se ve claramente cómo el bloque detecta cuando una señal de entrada pasa de 0 a 1 y genera un pulso en la salida correspondiente. Por ejemplo, cuando MCEDGE_IN cambia de 0 a 1, la salida MCEDGE_OUT responde con un pulso corto, indicando que el flanco fue detectado correctamente. Esto mismo ocurre con las otras señales de entrada, MCEDGE_IN2, MCEDGE_IN3 y MCEDGE_IN4, donde las salidas (MCEDGE_OUT2, MCEDGE_OUT3, MCEDGE_OUT4) también generan pulsos al detectar un cambio de 0 a 1 en sus entradas.

El bloque ignora cualquier fluctuación o ruido en las señales y solo responde cuando el cambio es claro, lo que asegura que las salidas sean precisas y confiables. Este comportamiento es importante porque permite al sistema reaccionar únicamente a eventos relevantes, como cuando un usuario presiona un botón o se activa una señal específica. En resumen, la simulación confirma que el bloque funciona correctamente y cumple su propósito de identificar flancos ascendentes y generar salidas sincronizadas.

## BLOQUE PRECIO

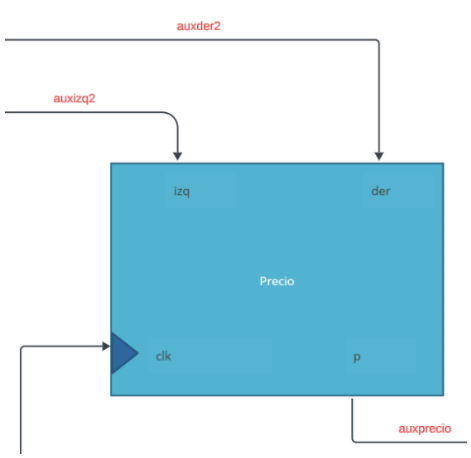


*Figura 8. Entradas y salidas para bloque Precio.*

El bloque Precio es el encargado de asignar el precio a los productos según las entradas izq (izquierda) y der (derecha). Funciona con tres estados: A, B y C, que representan tres precios diferentes. Dependiendo de qué botón se presione, el sistema cambia entre estos estados y ajusta la salida p, que muestra el precio actual en formato binario.

Por ejemplo, en el estado inicial (A), el precio es 1,00€ (100 en decimal). Si se presiona el botón de la derecha (der), el sistema cambia al estado B y el precio pasa a ser 5,30€. Si en cambio se presiona el botón de la izquierda (izq), cambia al estado C, donde el precio es 2,80€. A partir de ahí, los botones permiten moverse entre estos estados y precios de forma cíclica.

## MÁQUINA DE ESTADOS PARA PRECIO

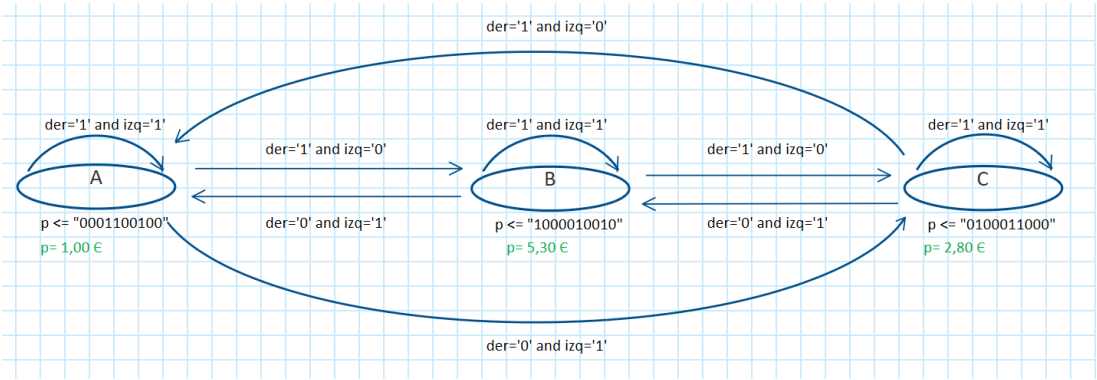


*Figura 9. Máquina de estados para la selección del precio de la Maquina de refrescos despachadora.*

# SIMULACIÓN PRECIO

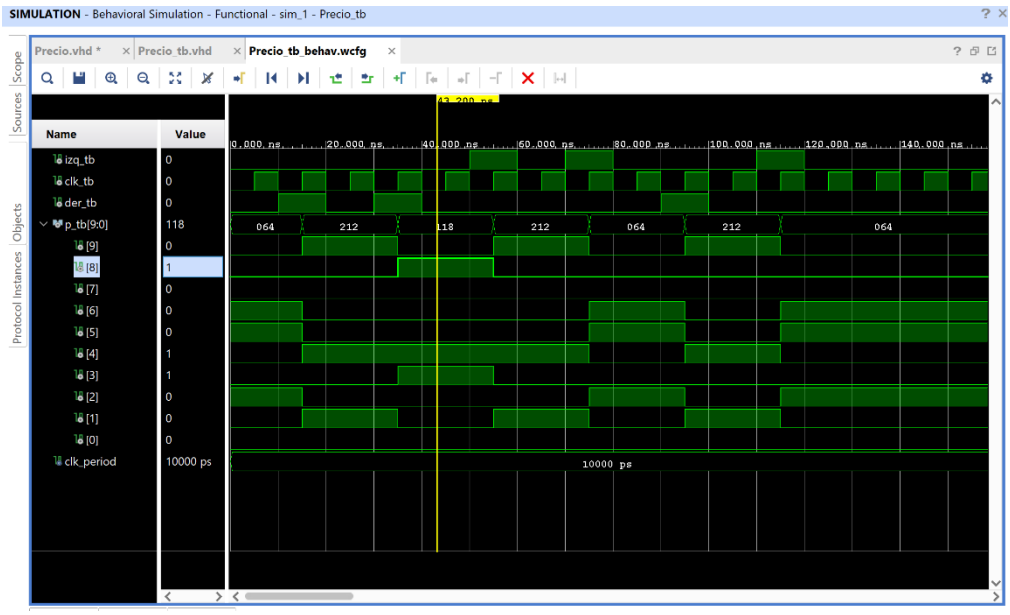


*Figura 10. Simulación para bloque Precio.*

En la simulación del bloque Precio, podemos observar cómo la máquina responde correctamente a los botones izq y der, que permiten cambiar entre los diferentes precios asociados a los estados A, B y C. Al iniciar, el sistema comienza en el estado A, donde el precio es 1,00€ (“0001100100”). Si se presiona el botón der, la máquina pasa al estado B, cambiando el precio a 5,30€ (“1000010010”). Si se presiona nuevamente der, la máquina avanza al estado C, donde el precio cambia a 2,80€ (“0100011000”).

Por otro lado, si queremos retroceder, presionamos el botón izq. Por ejemplo, si la máquina está en el estado C y se presiona izq, este regresa al estado B, restaurando el precio a 5,30€. Este comportamiento cíclico permite moverse entre los tres estados y precios según los botones presionados, haciendo que el usuario pueda seleccionar cualquier precio de forma sencilla.

## BLOQUE DIVISOR

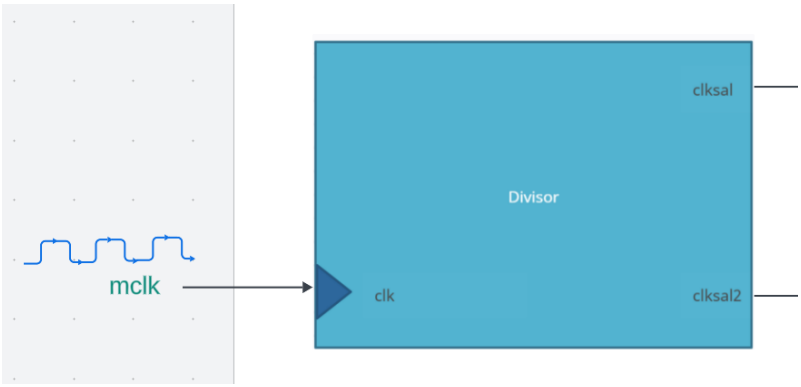


*Figura 11. Entradas y salidas para bloque Divisor.*

El bloque Divisor toma un reloj de entrada (clk) que funciona a una frecuencia alta y genera dos nuevas señales de reloj (clksal y clksal2) con frecuencias mucho más bajas. Esto se hace utilizando un contador de 24 bits que va sumando uno en cada pulso del reloj de entrada. A medida que el contador avanza, algunos de sus bits cambian más lento que otros, y esos bits se usan para crear las señales de salida.

Por ejemplo, la señal clksal2 utiliza el bit número 20 del contador, lo que significa que su frecuencia será $2^{20}$ veces más lenta que la frecuencia original. Como el master clock tiene una frecuencia de 100 MHz, clksal2 tendrá una frecuencia de alrededor de 95 Hz. Por otro lado, la señal clksal usa el bit 23 del contador, dividiendo aún más la frecuencia, resultando en aproximadamente 12 Hz para el mismo reloj de entrada.

## SIMULACIÓN DIVISOR

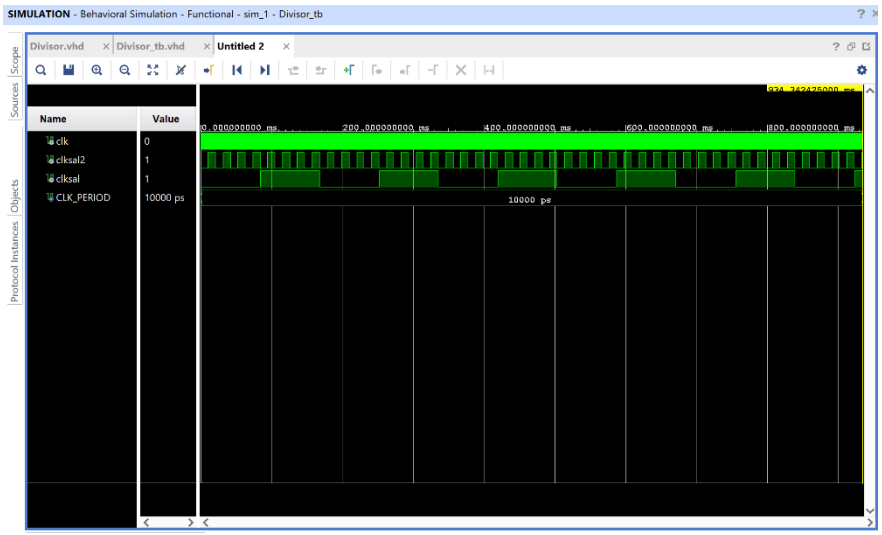


*Figura 12. Simulación para bloque Divisor.*

# BLOQUE TOT

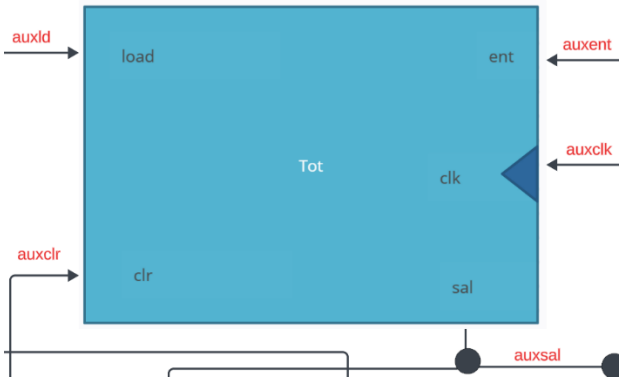


*Figura 13. Entradas y salidas para bloque Tot.*

En la máquina de refrescos, el bloque Controlador le dice a Tot cuándo cargar las monedas o dinero ingresado a la máquina (activando load) o cuándo reiniciar (devolver el dinero en la máquina de refrescos al usuario), esto se realiza activando clr. De esta forma, el controlador se asegura de que Tot tenga el valor correcto en el momento que el maquina lo necesita.

Por otro lado, el bloque Suma utiliza el dinero guardado en Tot (sal) como parte de los cálculos. Es decir, Tot le pasa el valor almacenado a Suma, logrando así que se ejecute la operación de suma con las monedas que el usuario ingrese (si es que se hace el ingreso nuevamente de monedas). Es decir, Tot actúa como un "almacén temporal" que se conecta al controlador para recibir órdenes y al bloque de suma para entregar el valor almacenado cuando se necesita.

# SIMULACIÓN TOT

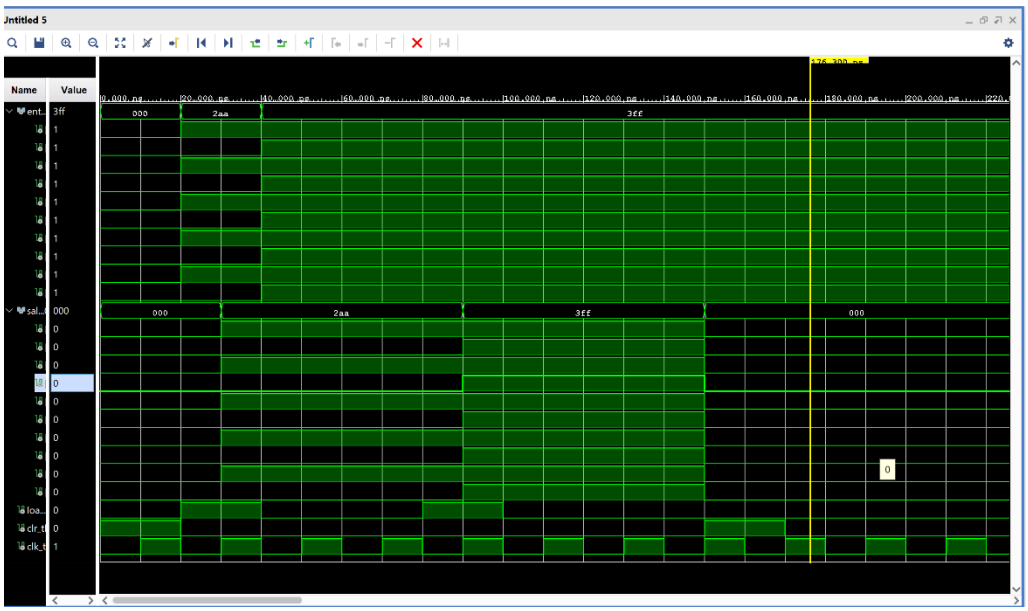


*Figura 14. Simulación para bloque Tot.*

En la simulación del bloque Tot, se puede ver cómo responde correctamente a las señales de control clr y load, y cómo almacena y actualiza el valor en la salida (sal).

Al inicio la salida tome el valor "000". Luego, se asigna un valor a la entrada ent ("1010101010") y se activa la señal load. En ese momento, el bloque carga este valor en la salida sal, reflejando el cambio de manera inmediata en el siguiente pulso del reloj. Esto confirma que el bloque responde correctamente cuando se le pide cargar un nuevo dato.

Más adelante, el valor de ent se cambia a "1111111111", pero como la señal load no está activa, el bloque no actualiza la salida sal y sigue mostrando el valor anterior. Finalmente, cuando se activa load nuevamente, el nuevo valor de ent se transfiere a la salida. En una última prueba, se activa otra vez clr y la salida vuelve a cero, mostrando que el reinicio funciona como se espera.

# BLOQUE SUMADOR

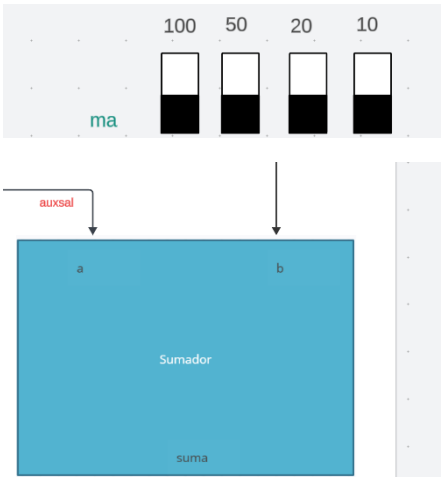


*Figura 15. Entradas y salidas para bloque Sumador.*

El bloque Sumador se encarga de realizar una operación de suma entre dos entradas: a, que corresponde a la salida sal del bloque Tot, y b, que representa el valor de la moneda ingresada en la máquina por el usuario. El resultado de esta suma se refleja en la salida suma, que es un valor acumulado de 10 bits.

La entrada b utiliza una tabla interna que convierte su valor de 4 bits en un número equivalente al valor de la moneda ingresada. Por ejemplo, si b es "0001", se interpreta como 10 ("0000001010"), y si b es "1000", se interpreta como 100 ("0001100100"). El bloque luego suma este valor al acumulado actual que proviene de a (es decir, de la salida sal del bloque Tot).

```
c <= "0000000000" when b="0000" else --0
     "0000001010" when b="0001" else --10
     "0000010100" when b="0010" else --20
     "0000011110" when b="0011" else --30
     "0000110010" when b="0100" else --50
     "0000111100" when b="0101" else --60
     "0001000110" when b="0110" else --70
     "0001010000" when b="0111" else --80
     "0001100100" when b="1000" else --100
     "0001101110" when b="1001" else --110
     "0001111000" when b="1010" else --120
     "0010000010" when b="1011" else --130
     "0010010110" when b="1100" else --150
     "0010100000" when b="1101" else --160
     "0010101010" when b="1110" else --170
     "0010110100" when b="1111" else --180
     "UUUUUUUUUU";
```

*Figura 16. Conversor de monedas para bloque Sumador.*

# SIMULACIÓN SUMA

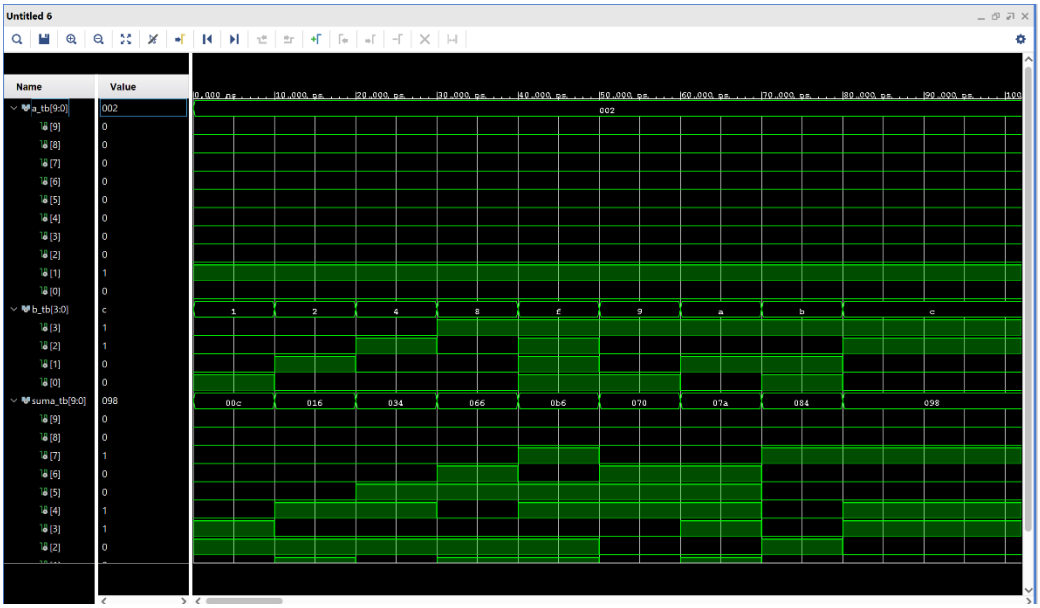


*Figura 17. Simulación para bloque Suma.*

En la simulación del bloque Suma, podemos observar cómo funciona correctamente al sumar los valores de las entradas a y b para generar el resultado en la salida suma. La entrada a (proveniente de la salida sal del bloque Tot) representa el valor acumulado hasta ese momento (en este caso, siempre es 2), mientras que la entrada b corresponde al valor de una moneda (o monedas) traducido internamente al valor que tendrá dentro de la máquina.

Por ejemplo, en el primer caso de prueba, a es 2 y b tiene el valor "0001", que equivale a 10. El bloque realiza la suma, generando un resultado de 12 en la salida suma. En los siguientes casos, b toma diferentes valores: "0010" (20), "0100" (50), y así sucesivamente, mientras que a permanece constante en 2. En cada caso, la salida suma refleja correctamente el resultado, como 22, ó 52 confirmando que el bloque está procesando las sumas de manera precisa.

El caso cuando b tiene el valor más alto es, "1111", que corresponde a 180. El bloque suma este valor con los 2 de a, generando una salida de 182.

# BLOQUE COMPARADOR

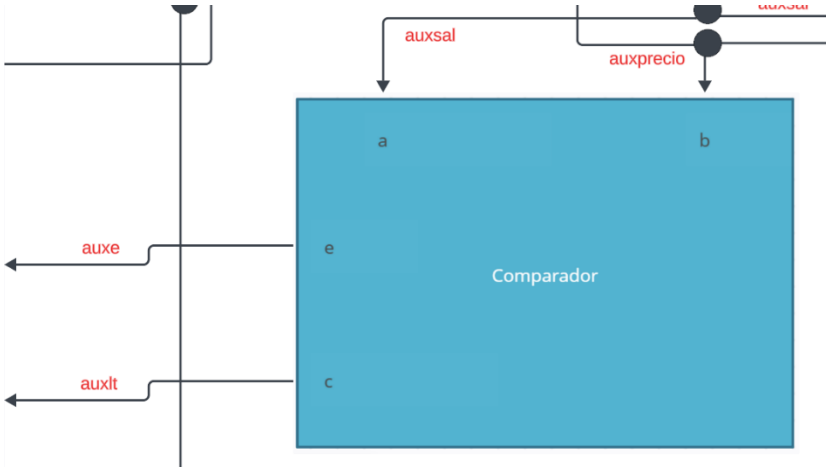


*Figura 18. Entradas y salidas para bloque Comparador.*

El bloque Comparador es fundamental en el sistema, ya que se encarga de comparar el valor acumulado de dinero (a, que proviene de la salida sal del bloque Tot) con el precio del producto (b, que es la salida p del bloque Precio). Según esta comparación, genera dos señales de salida: c y e, que indican si el dinero ingresado es suficiente o si hay algún error.

La salida c controla si se puede despachar el refresco. Si el valor de a (dinero ingresado) es menor que b (precio del producto), el sistema indica que no se puede entregar el refresco (c = '1'). Si el dinero ingresado es igual al precio (a = b), la señal c se pone en 0, permitiendo que se entregue el producto (también puede ser mayor la cantidad ingresada que el precio del refresco $a \geq b$). Por otro lado, si el dinero ingresado supera al precio, aunque el refresco se puede despachar (c = '0'), se activa la señal de error (e = '1'), indicando que hay un exceso de dinero, lo que puede interpretarse como un error o la necesidad de devolver cambio (sólo si sw_c='0') Pero esto lo decidirá el bloque Controlador.

# SIMULACIÓN COMPARADOR

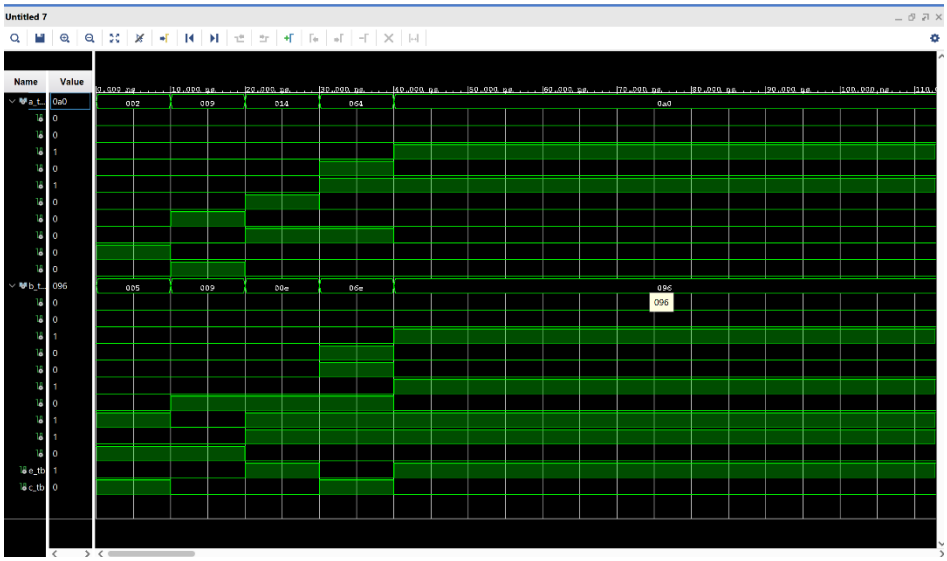


*Figura 19. Simulación para bloque Comparador.*

En la simulación del bloque Comparador, se puede observar cómo este bloque compara correctamente las dos entradas a (valor acumulado del bloque Tot) y b (precio del producto proveniente del bloque Precio), y genera las señales de salida c y e según la relación entre los valores.

En el primer caso de prueba, a es 2 y b es 5. Como a es menor que b, la señal c se pone en 1, indicando que no se puede despachar el producto porque el dinero ingresado no es suficiente. La señal e, que indica error por exceso de dinero, permanece en 0, ya que no se ha ingresado más dinero del necesario.

En el segundo caso, tanto a como b tienen el mismo valor (9). Aquí, la salida c cambia a 0, habilitando la entrega del producto, y e sigue en 0, ya que no hay exceso de dinero. Esto demuestra que el bloque responde adecuadamente cuando el monto acumulado coincide exactamente con el precio.

En el tercer caso, a (20) es mayor que b (14). En este escenario, la señal c permanece en 0, permitiendo que el producto sea despachado, pero ahora la señal e se activa (1), indicando que hay más dinero del necesario, lo que podría requerir un cambio o un error según sea el caso.

# BLOQUE CAMBIO

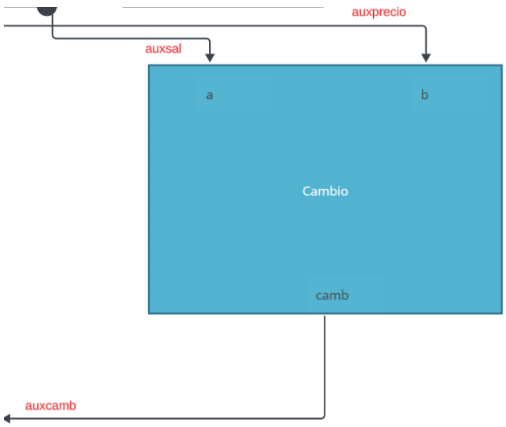


*Figura 20. Entradas y salidas para bloque Cambio.*

El bloque Cambio es responsable de calcular el monto que debe devolverse al usuario como cambio cuando el dinero ingresado (a, que proviene de la salida sal del bloque Tot) es mayor o igual al precio del producto (b, que proviene de la salida p del bloque Precio). El resultado de esta operación se envía a través de la señal Camb, que luego es utilizada por el bloque Display para decodificar y mostrar el valor del cambio en el display de la Nexys.

El bloque funciona de la siguiente manera: si el dinero ingresado (a) es menor que el precio (b), el bloque coloca el valor en Camb ("1111111111") que puede interpretarse como una señal para indicar que no se puede calcular el cambio porque el monto ingresado no es suficiente y en el bloque Display en lugar de mostrarse el cambio se mostrar el acumulado hasta el momento. Si el dinero ingresado es igual o mayor al precio, el bloque calcula la diferencia (a - b) y coloca el resultado en Camb. Este valor será más adelante procesado por el Display para que el usuario pueda ver el cambio que se le debe devolver.

# SIMULACIÓN CAMBIO

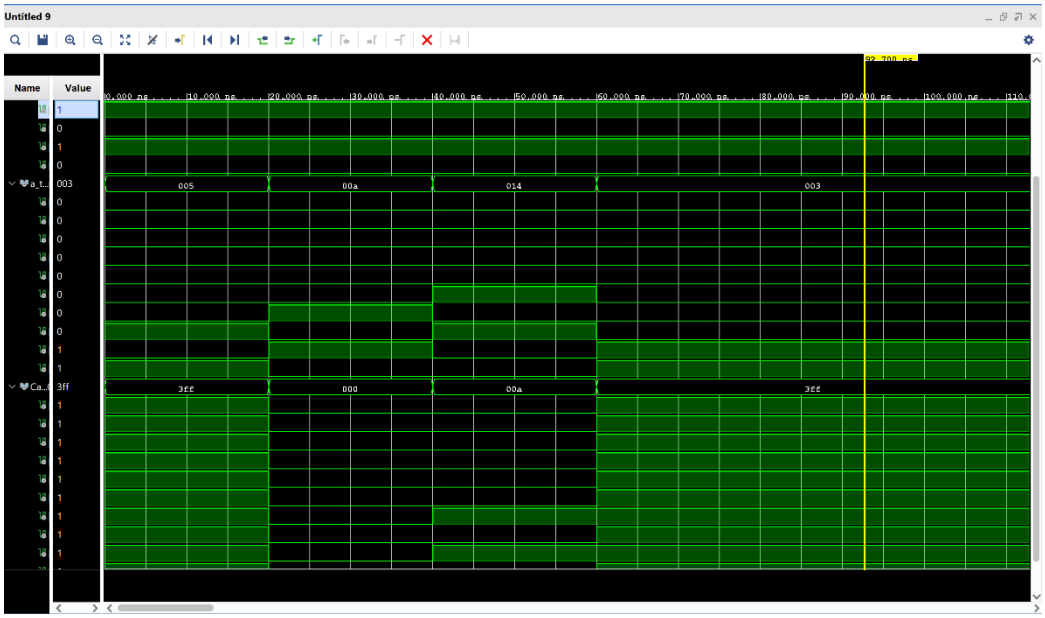


*Figura 21. Simulación para bloque Cambio.*

En la simulación del bloque Cambio, podemos observar cómo responde correctamente en diferentes escenarios al calcular el cambio entre las entradas a (dinero ingresado, proveniente del bloque Tot) y b (precio del producto, proveniente del bloque Precio). El resultado de este cálculo se refleja en la señal de salida Camb, que luego se envía al bloque Display para mostrar el cambio al usuario.

En el primer caso, a tiene un valor de 5 y b es 10, lo que significa que el dinero ingresado no es suficiente para cubrir el precio del producto. En este escenario, el bloque asigna el valor a Camb de “1111111111", no se puede calcular el cambio.

En el segundo caso, a y b son iguales (10). Aquí, el bloque realiza la operación de resta y calcula correctamente que no hay cambio que devolver, colocando un valor de "000" en la salida Camb.

En el tercer caso, a es mayor que b (20). El bloque calcula la diferencia (a - b) y coloca el resultado, 10 ("0000001010"), en la salida Camb, indicando que este es el cambio que debe devolverse al usuario.

## BLOQUE DISPLAY

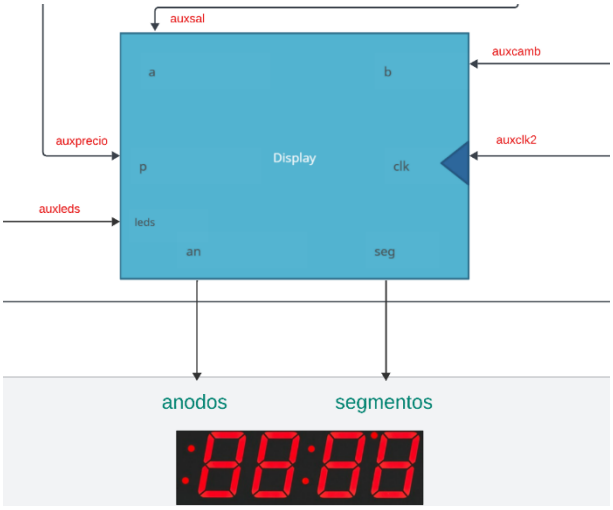


*Figura 22. Entradas y salidas para bloque Display.*

El bloque Display es el encargado de controlar y actualizar los valores que se muestran en el display de siete segmentos conectado a la Nexys. Este bloque toma como entradas el dinero acumulado (a del bloque Tot), el cambio calculado (b del bloque Cambio), y el precio del producto (p del bloque Precio). Con estas entradas, decide qué información se mostrará en los segmentos y dígitos del display.

El funcionamiento comienza con un contador interno sincronizado con el reloj (clk) que determina qué dígito del display está activo en cada momento. Esto permite manejar múltiples dígitos del display de forma secuencial, iluminando solo uno a la vez mediante las señales an, mientras las señales seg controlan los segmentos para representar los valores.

Para cada entrada, el bloque procesa sus valores decodificándolos en centenas, decenas y unidades. Por ejemplo, el dinero acumulado a se divide en centenas (c_a), decenas (d_a), y unidades (u_a), y lo mismo ocurre con el cambio (b). Dependiendo del valor de b, el bloque también muestra una letra especial como A (modo acumulado) o C (modo cambio), y en el caso de p, se muestra el precio junto con una letra identificativa del refresco (d, E, o F) según el valor seleccionado.

El bloque utiliza un decodificador para convertir los valores numéricos y las letras en patrones que puedan ser entendidos por los segmentos del display. Por ejemplo, si el valor decimal de un dígito es 2, el decodificador genera el patrón que enciende los segmentos correspondientes a "2" en el display ("0010010").

```
process(dato)
begin
    case dato is
        when "0000000" => seg <= "0000001"; -- 0
        when "0000001" => seg <= "1001111"; -- 1
        when "0000010" => seg <= "0010010"; -- 2
        when "0000011" => seg <= "0000110"; -- 3
        when "0000100" => seg <= "1001100"; -- 4
        when "0000101" => seg <= "0100100"; -- 5
        when "0000110" => seg <= "0100000"; -- 6
        when "0000111" => seg <= "0001111"; -- 7
        when "0001000" => seg <= "0000000"; -- 8
        when "0001001" => seg <= "0000100"; -- 9
        when "0001010" => seg <= "0001000"; -- A
        when "0001100" => seg <= "1110010"; -- C
        when "0001101" => seg <= "1000010"; -- d
        when "0001110" => seg <= "0110000"; -- E
        when "0001111" => seg <= "0111000"; -- F
        when others => seg <= "1111111"; -- Apagado
    end case;
end process;
```

*Figura 23. Decodificador para Display de 7 segmentos para bloque Display.*

```
-- Procesamiento de a en señales intermedias u_a, d_a, c_a
process (a)
    variable a_decimal : integer;
begin
    a_decimal := to_integer(unsigned(a)); -- Convertir a a decimal
    u_a <= std_logic_vector(to_unsigned(a_decimal mod 10, 7)); -- Unidades de a
    d_a <= std_logic_vector(to_unsigned((a_decimal / 10) mod 10, 7)); -- Decenas de a
    c_a <= std_logic_vector(to_unsigned((a_decimal / 100) mod 10, 7)); -- Centenas de a
end process;

-- Procesamiento de b en señales intermedias u_b, d_b, c_b
process (b)
    variable b_decimal : integer;
begin
    b_decimal := to_integer(unsigned(b)); -- Convertir b a decimal
    u_b <= std_logic_vector(to_unsigned(b_decimal mod 10, 7)); -- Unidades de b
    d_b <= std_logic_vector(to_unsigned((b_decimal / 10) mod 10, 7)); -- Decenas de b
    c_b <= std_logic_vector(to_unsigned((b_decimal / 100) mod 10, 7)); -- Centenas de b
end process;
```

*Figura 24. Procesamiento del valor acumulado de dinero (a) o del cambio del usuario (camb) para el bloque Display.*

# SIMULACIÓN DISPLAY

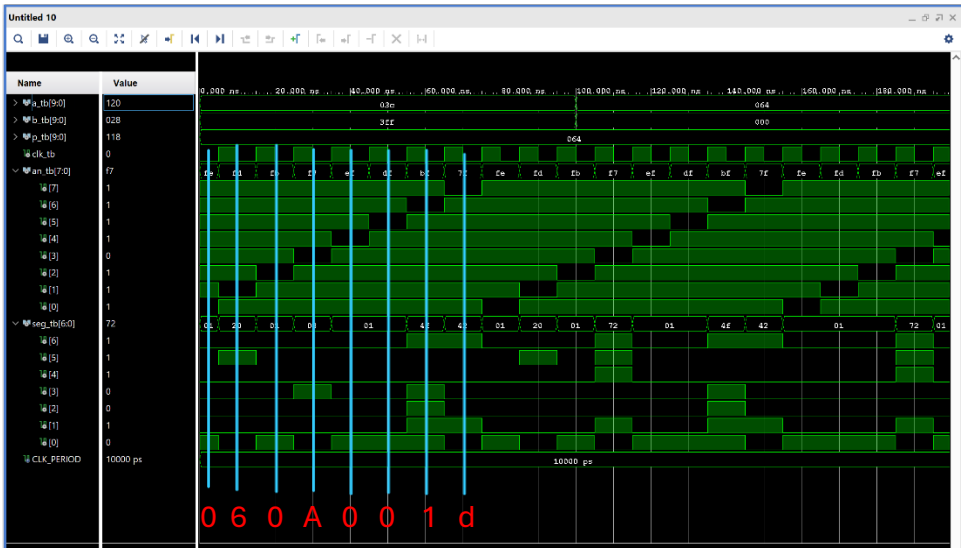


*Figura 23. Simulación para bloque Display.*

En la simulación del bloque Display, el Caso 1 destaca porque muestra el comportamiento del sistema cuando el dinero ingresado es insuficiente para cubrir el precio del producto. En este caso, el precio del producto p está configurado en 1,00€, mientras que el dinero acumulado a es 0,60€. Como resultado, el bloque Display toma el valor de b de "1111111111", dinero insuficiente, y lo utiliza para mostrar un mensaje de dinero acumulado en el display. Esto se traduce visualmente en la representación de los segmentos del display, mostrando un patrón que alerta al usuario sobre la situación.

El bloque Display alterna entre las diferentes posiciones del display para mostrar el dinero acumulado (0,60€) acompañado de la letra 'A' ("0001000"), el precio del producto (1,00€) acompañado de la letra que corresponde a ese precio (d-"1000010").

En los demás casos, el comportamiento cambia según las entradas. Por ejemplo, en el Caso 2, donde el dinero acumulado es igual al precio (a = p = 1,00€), el bloque muestra que no hay cambio (b= 0,00€) y actualiza los segmentos para reflejar esta información. En el Caso 3, cuando hay dinero excedente (a > p), el bloque calcula y muestra el cambio correspondiente (b= 0,20€). Finalmente, en el Caso 4, con un precio diferente (2,80€) y dinero ingresado mayor (3,20€), se muestra el cambio resultante (0,40€) junto con los otros valores.

# TOP

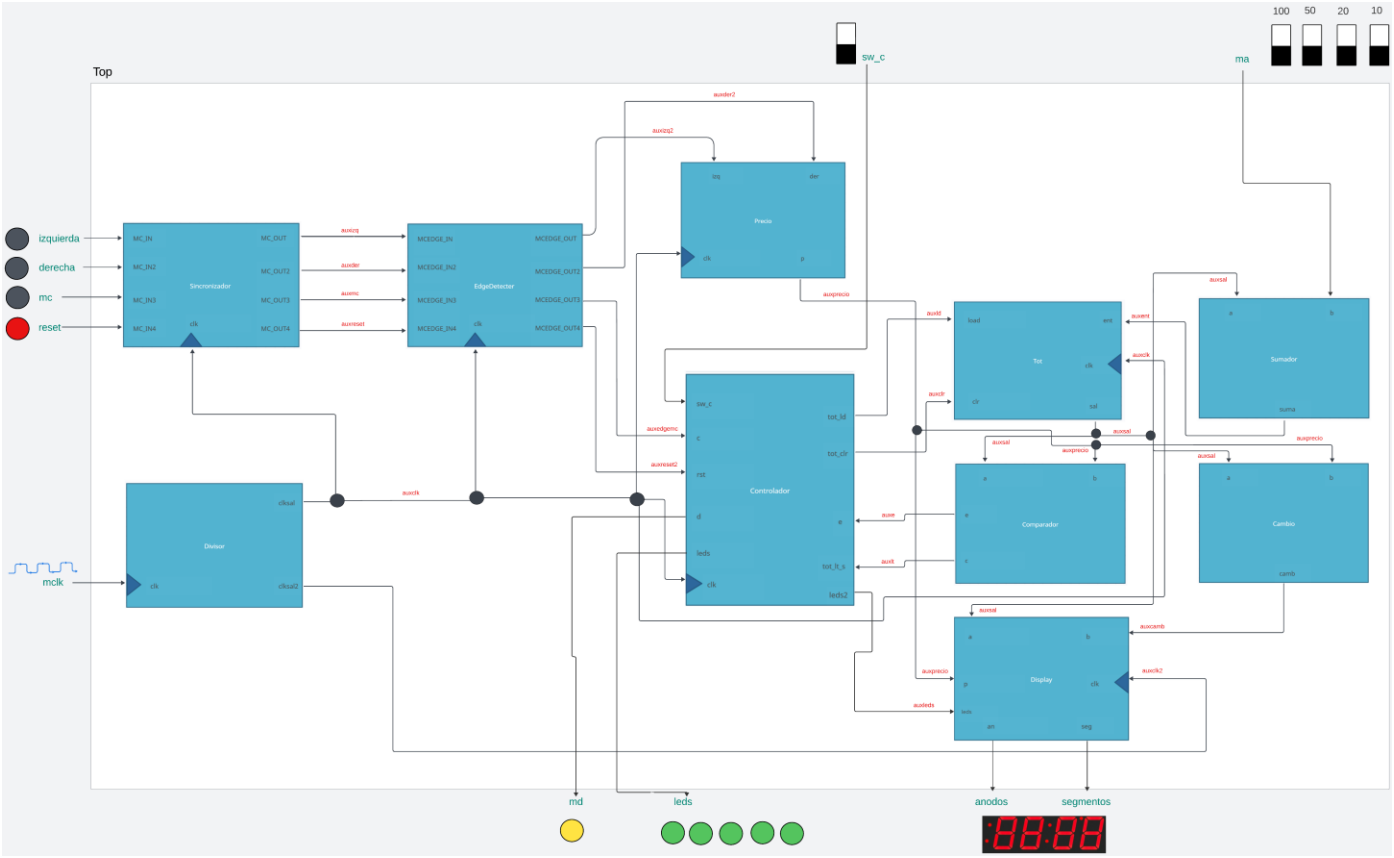


*Figura 24. Diagrama de bloques para la Maquina Despachadora de Refrescos.*

El Top, es el núcleo donde se instancian y conectan todos los bloques que componen el sistema de la máquina expendedora. Aquí se instancian todos los bloques: Controlador, Divisor, Tot, Sumador, Comparador, Cambio, Display, Precio, EDGEDTCTR y SYNCHRNZR, cada uno desempeñando roles específicos.

Los bloques se conectan entre sí mediante señales intermedias como auxld, auxclr, auxclk, auxprecio, entre otras, que facilitan la comunicación y el flujo de datos entre los componentes.

## Entradas externas:

Push buttons

- izquierda: Botón para cambiar el precio del refresco hacia abajo.
- derecha: Botón para cambiar el precio del refresco hacia arriba.
- mc: Botón que simula la introducción de monedas.
- reset: Botón para reiniciar todo el sistema.

Slide switches

- sw_c: Interruptor que determina si la máquina debe dar cambio.
- ma: Entrada que representa el valor de las monedas introducidas, usando 4 bits para representar denominaciones como 10, 20, 50 y 100.

Reloj

- mclk: Señal de reloj principal que proviene de la tarjeta Nexys, utilizada para sincronizar todas las operaciones internas.

## Salidas externas:

Displays de la Nexys:

- anodos: Salida de 8 bits que controla cuál de los displays de 7 segmentos está activo en un momento dado.
- segmentos: Salida de 7 bits que determina qué segmentos individuales se iluminan para mostrar números o letras en el display.

Leds:

- leds: Salida de 5 bits que indica error al ingresar dinero de más.
- md: Salida que indica si se despachó un refresco.

## SIMULACIÓN TOP

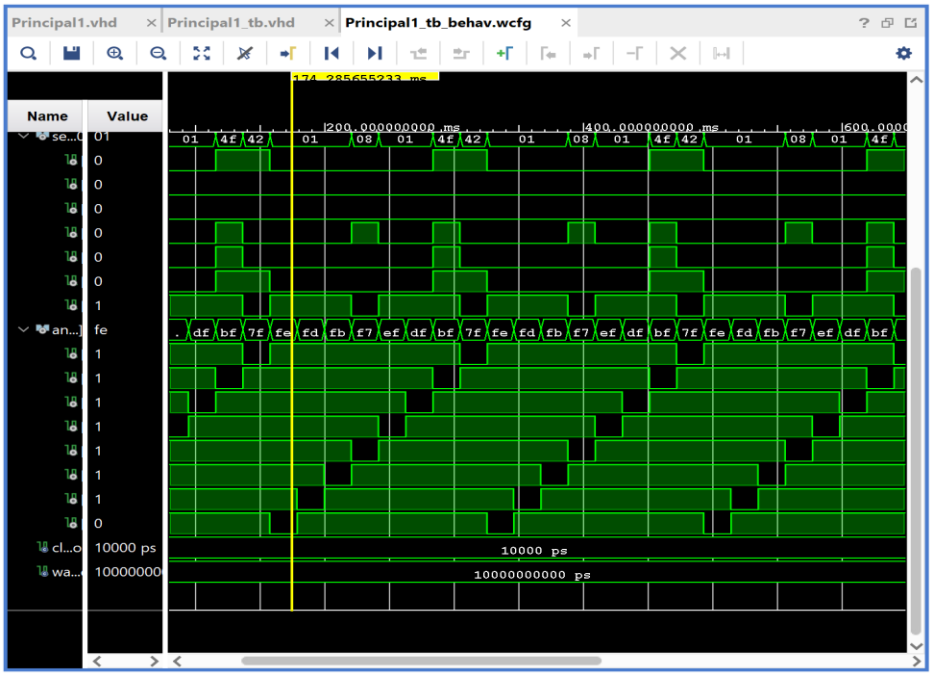


*Figura 25. Simulación para Top.*

En la simulación se observa cómo se integran los diferentes bloques para simular el comportamiento completo de la máquina expendedora de refrescos.

En el caso 1, el interruptor de cambio sw_c está activado, el usuario introduce monedas que superan el precio del producto (1,00€). Aquí, el sistema calcula correctamente el cambio, que es reflejado en el display como el valor restante. Por ejemplo, si el usuario ingresa 1,80€, el cambio calculado y mostrado es 0,80€, mientras que el display también alterna entre el precio del producto (1,00€) mostrando la letra “p” y el cambio (0,80€), mostrando la letra "c".

En el caso 2, sw_c está desactivado, por lo que solo permite acumular exactamente el precio del producto. Si el usuario intenta ingresar más dinero del necesario, la máquina genera un error, indicado por los leds encendidos ("11111") y el display mostrando el mensaje de error. Como el precio del producto es 1,00€ y el usuario ingresa 1,80€, el sistema no permite la operación y devuelve el dinero acumulado.

En el caso 3, podemos notar que el display refleja los cambios en el precio del refresco junto con el identificador del producto correspondiente ("d", "E" o "F").

# RESULTADOS

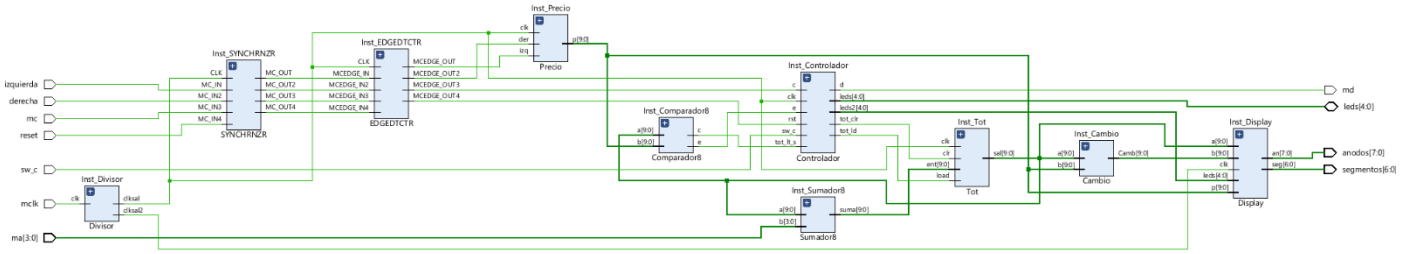


*Figura 26. Diagrama de bloques para la máquina de estados generado por Vivado.*

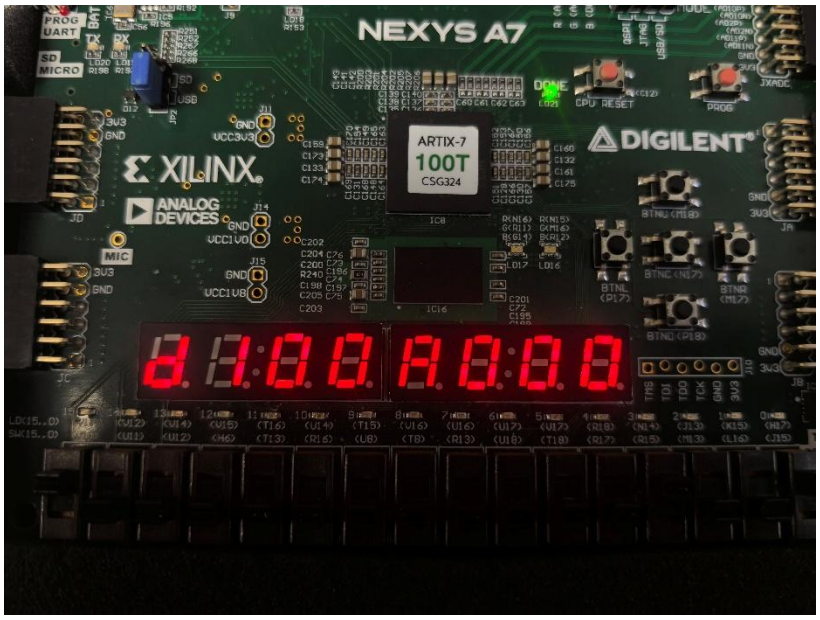


*Figura 27. Presentación del refresco "d " y su precio (1,00€), además del estado inicial del dinero acumulado en la máquina (0).*

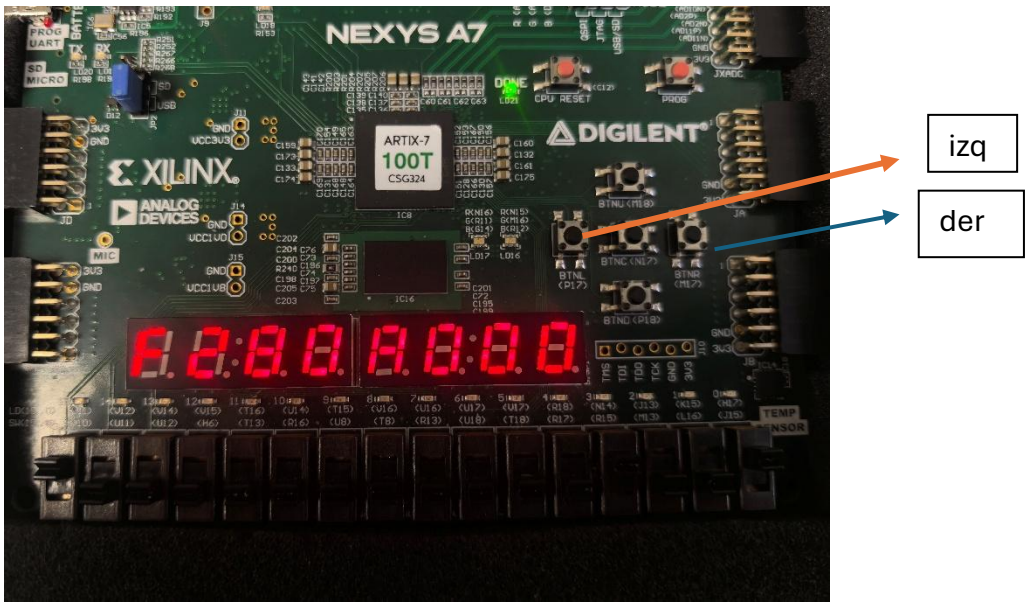


*Figura 28. Presentación del refresco "F " y su precio (2,80€), además del estado inicial del dinero acumulado en la máquina (0).*

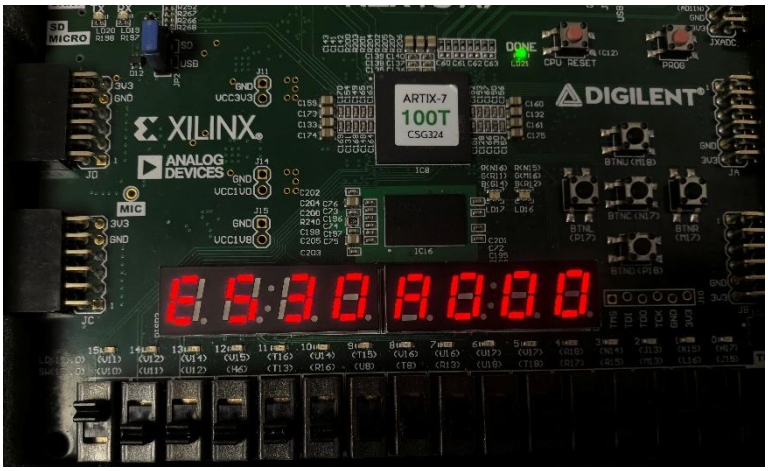

*Figura 29. Presentación del refresco "E " y su precio (5,30€), además del estado inicial del dinero acumulado en la máquina (0).*

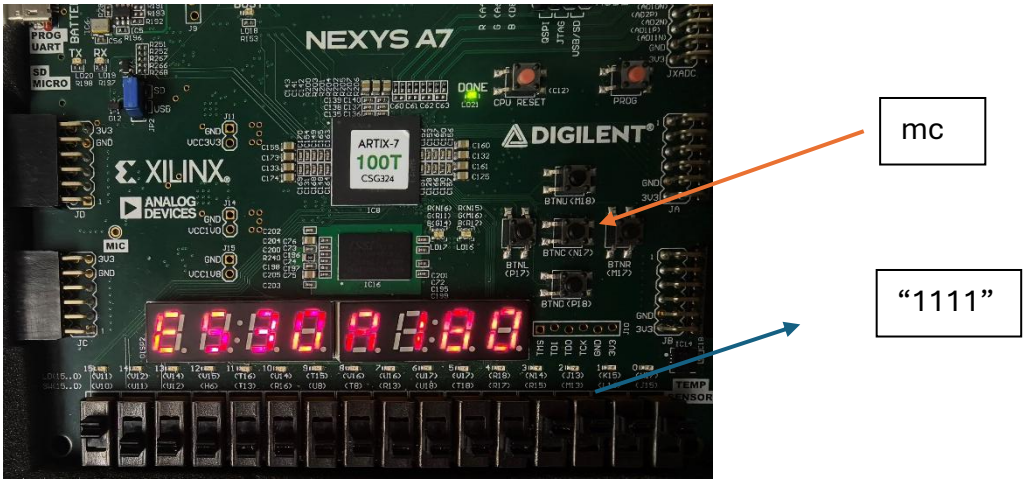


*Figura 30. Se ingresan 1,80€ por medio de los slide switch's (ma) y con el botón de carga (mc)*

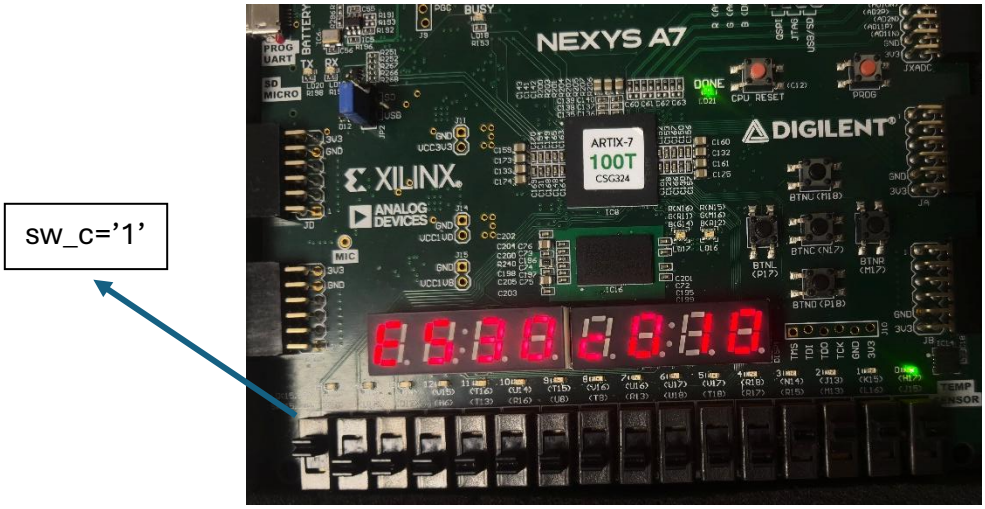


*Figura 31. Se ingresan 1,80€ 3 veces a la máquina, superando el precio de 5,30 €, con lo que obtenemos 0,10 € de cambio(sw_c='1'). Podemos notar el led md que indica que la máquina despachó producto.*

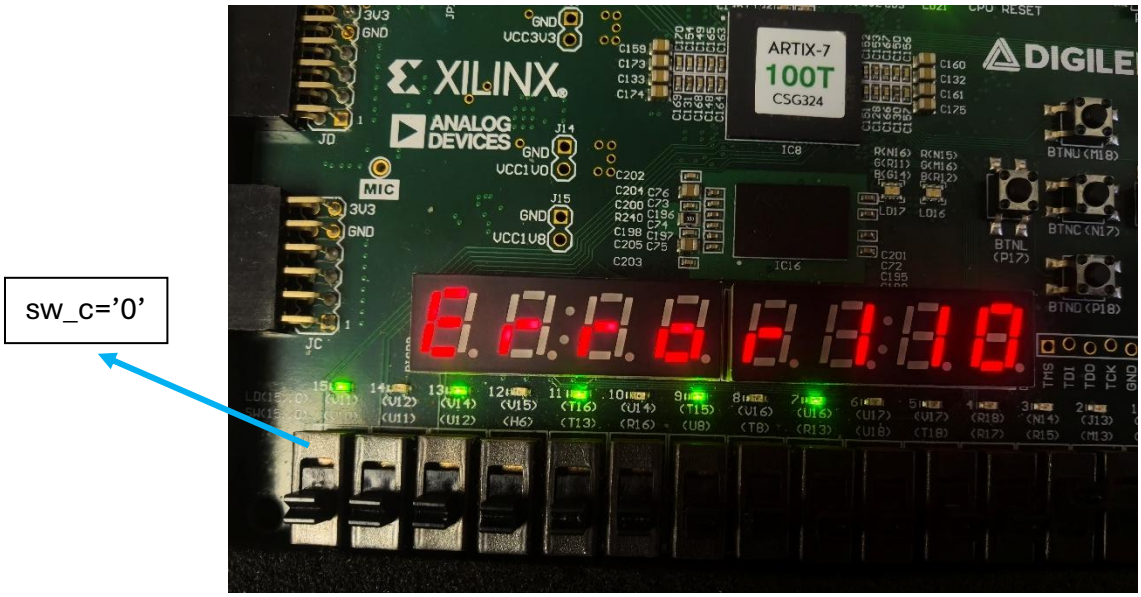


*Figura 32. Se ingresa 1,10€ a la máquina, superando el precio del refresco (1,00 €), con lo que obtenemos un error (mostrado tanto en el display como en los leds) esto debido a que la máquina no da cambio (sw_c=’0’).*

# MEJORAS

En esta fase del proyecto, se decidió incorporar un módulo adicional (entidad SPI_Slave) con el fin de habilitar la comunicación serial de tipo SPI con un microcontrolador, en particular, el STM32F411E-DISCO.

## BLOQUE SPI_SLAVE

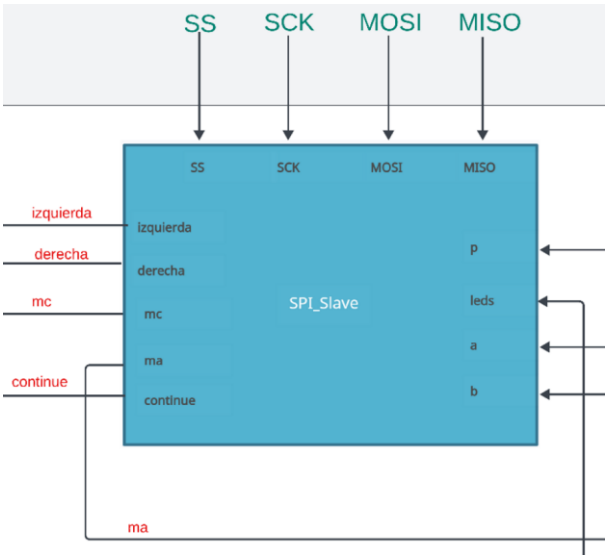


*Figura 33. Entradas y salidas para bloque SPI_Slave.*

El bloque SPI_Slave es responsable de implementar la comunicación serial SPI en la máquina, actuando como el esclavo, y con un tipo de comunicación SPI. Básicamente permite recibir y transmitir datos entre el microcontrolador STM32F411E-DISCO y otros bloques dentro de la máquina de refrescos.

Recepción de datos: Recibe un byte enviado desde el microcontrolador STM32 a través de la línea MOSI. Los datos se reciben bit a bit y se almacenan en registros internos hasta que se completa la recepción de un byte completo y se lo asigna a nuestras señales auxiliares, mismas que se conectan a los demás bloques del sistema.

Fue importante mandar el registro de los bits recibidos solo cuando se hubiera terminado la recepción, ya que de esa manera evitamos errores de funcionamiento con la máquina.

```
izquierda <= byte_received(7);
derecha <= byte_received(6);
mc <= byte_received(5);
ma <= byte_received(4 downto 1);
continue <= byte_received(0);
```

*Figura 34. Recepción de un byte por medio del bloque SPI_Slave.*

Transmisión de datos: Transmite 3 bytes hacia el microcontrolador a través de la línea MISO. Los datos por transmitir se definen mediante las señales p, leds, a, y b, que representan lo siguiente:

p: Representan los dos últimos bits más significativos del valor del precio “p” del Bloque Precio. Ya que nos dimos cuenta de que estos bits cambian entre cada uno de los precios, por lo que no necesitábamos saturar de información al microcontrolador, además de esta forma se reduce el número de bytes a enviar.

a: Se enviarán los 10 bits que representa el dinero total acumulado por la maquina despachadora de refrescos.

b: Se enviarán los 10 bits que representa el cambio en dinero cuando “a” alcanza el precio del refresco, o “1111111111” cuando no sea así.

leds: Se enviará un único bit de los 5, ya que de la misma forma que “p” disminuimos el numero de bytes a enviar.

```
send_bytes1 <= a(9 downto 2);
send_bytes2 <= a(1 downto 0) & b(9 downto 4);
send_bytes3 <= b(3 downto 0) & p(1 downto 0) & leds(4) & '0';
```

*Figura 35. Envío de bytes por medio del bloque SPI_Slave.*

Control de la comunicación: El bloque controla la sincronización de la transmisión y recepción mediante la señal de reloj SCK (250KHz) y la señal de selección de chip SS. La comunicación se lleva a cabo solo cuando SS está activa (SS=0), lo que permite la correcta gestión de los datos.

# SIMULACIÓN SPI_SLAVE

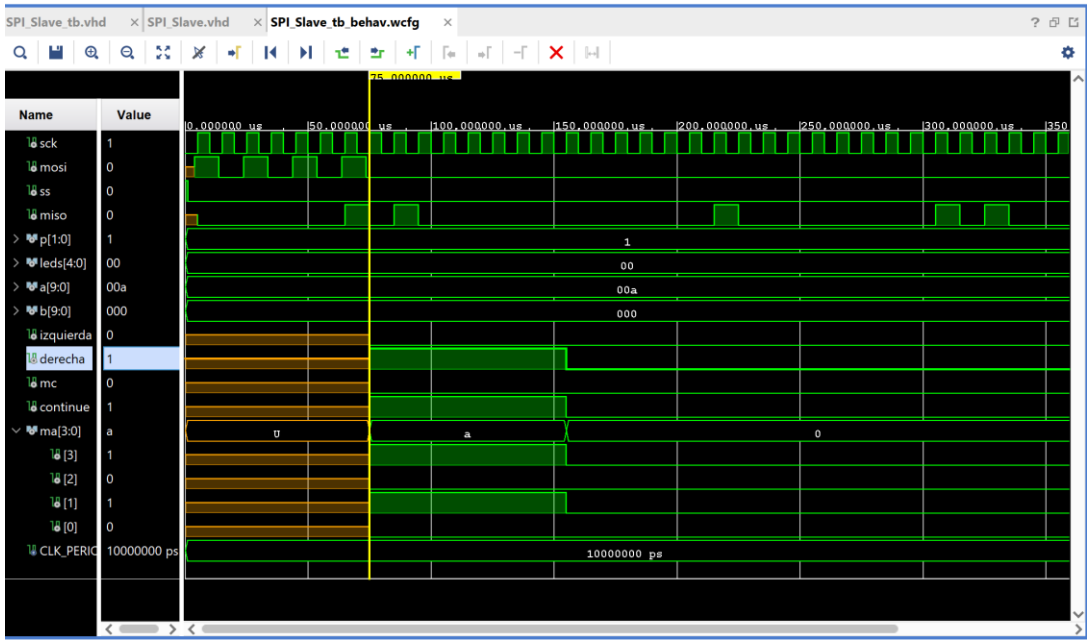


*Figura 36. Simulación para SPI_Slave.*

De la simulación podemos observar que después de los primeros ciclos de reloj (SCK) provenientes del microcontrolador, obtenemos una lectura de un byte (“10101010”) por medio de la línea MOSI, tal y como indicamos que se hiciera en el testbench. Lo que nos permite saber que nuestro código permite la recepción y correcta asignación de valores a nuestras salidas.

```
process
begin
    ss <= '1';
    wait for CLK_PERIOD / 8;
    ss <= '0';
    wait for CLK_PERIOD / 4;
    mosi <= '1';
    wait for CLK_PERIOD ;
    mosi <= '0';
    wait for CLK_PERIOD ;
    mosi <= '1';
    wait for CLK_PERIOD ;
    mosi <= '0';
    wait for CLK_PERIOD ;
    mosi <= '1';
    wait for CLK_PERIOD ;
    mosi <= '0';
    wait for CLK_PERIOD ;
    mosi <= '1';
    wait for CLK_PERIOD ;
    mosi <= '0';
    wait for 300us;
    assert false
    report "[PASSED]: simulation finished"
    severity failure;
end process;
```

*Figura 37. Valores (“10101010”) “enviados por el Master” y recibidos por el slave (FPGA).*

Desde que comenzó la señal de SCK el programa fue capaz de mandar los valores de “a”, “b”, “p” y “leds” al Master por medio de la línea MISO, lo que comprueba que nuestra máquina de refrescos es capaz de mandar bytes al microcontrolador.

```
a <="0000001010";
b <="0000000000";
p <="01";
leds <="00000";
```

*Figura 38. Valores de a, b, p y leds “recibidos por el Master” y enviados por el slave (FPGA).*

## TOP FINAL

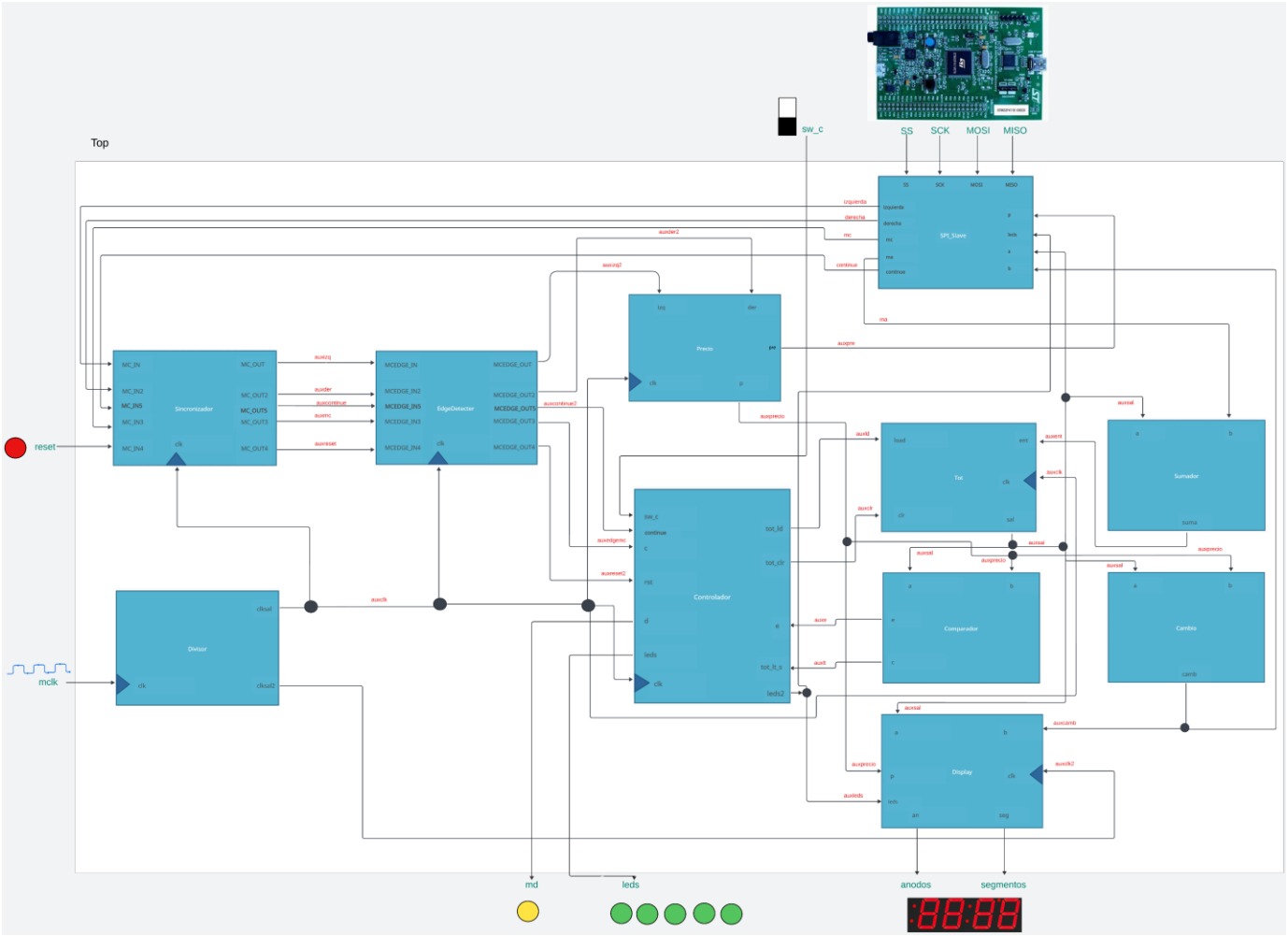


*Figura 39. Diagrama de bloques mejorado para la Maquina Despachadora de Refrescos.*