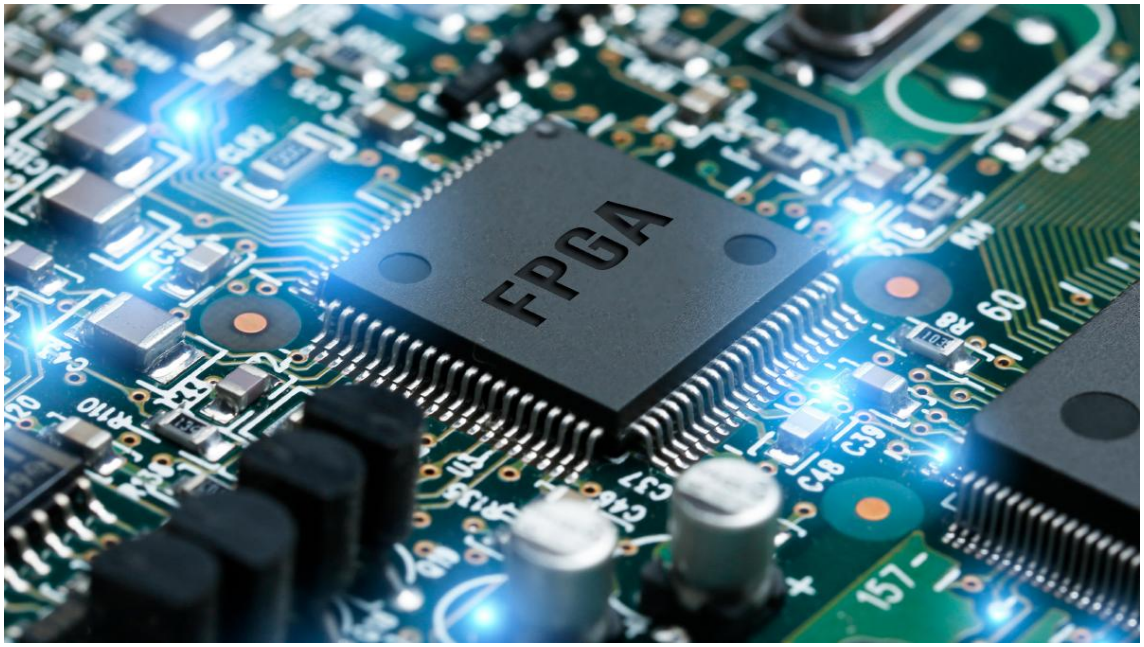
TRABAJO VHDL



GRUPO 24

Mario Fernández Liétor, 55839

Sergio Ballesteros Palomo, 55736

Alejandro Bonilla Hervás, 55756

ÍNDICE

1. INTRODUCCIÓN Y OBJETIVOS 3
1.1. Enunciado 3
1.2. Objetivos 3
2. DESCRIPCIÓN DEL PROYECTO 3
2.1. Funcionamiento 3
2.2. Entradas y salidas 4
2.2.1 Entradas 4
1. CLK (Reloj) 4
2. RESET (Reinicio) 4
3. TIPO_MONEDA (Vector de Entradas) 4
4. PAGO (Confirmación de Pago) 4
5. TIPO_REFRESCO (Selección del Producto) 4
2.2.2 Salidas 5
1. PRODUCTO_OK (Entrega del Producto) 5
2. ESTADOS (Vector de Estados) 5
3. DIGCTRL (Control de Displays de 7 Segmentos) 5
4. SEGMENTOS 5
2.3. Diagrama de estados 6
3. DESARROLLO DE ENTIDADES Y SIMULACIÓN 8
3.1. Esquema de identidades 8
3.2. SYNCHRONIZER 11
3.3. EDGE_DETECTOR 13
3.4. COUNTER 14
3.5. PRESCALER 18
3.6. DISPLAY_CONTROL 19
3.7. DECODER 22
3.8. FSM 24
3.9. MAQ_EXP (TOP) 27
4. GITHUB 27

1. INTRODUCCIÓN Y OBJETIVOS

1.1. Enunciado

Para esta fase del trabajo, implementaremos en la placa NEXYS 4 DDR/NEXYS A7 la solución al problema de la máquina expendedora, elegido de entre las propuestas disponibles en Moodle. El enunciado que define los requisitos del sistema es el siguiente:

> *"Diseñe una máquina expendedora de refrescos. 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 importe exacto del refresco (1€) se activará una señal para dar el producto. Como entradas tendrá señales indicadoras de la moneda, señales indicadoras de producto y como salidas la señal de error y la de producto"*

1.2. Objetivos

El principal propósito del trabajo es el diseño de la máquina expendedora como se propone en el enunciado. Además de esto, se han pensado más funcionalidades que se implementarán en el diseño inicial, como son:

- Añadir un refresco adicional a la máquina expendedora.
- Personalizar el precio de los refrescos sin realizar grandes modificaciones en el código.
- Utilizar el display de 7 segmentos visto en las prácticas de laboratorio.
- Botón de reset con el que se vuelva al estado inicial.
- Devolución de cambio, en el caso de sobrepasar el pago.

2. DESCRIPCIÓN DEL PROYECTO

2.1. Funcionamiento

Para el diseño y estructura del proyecto, se utilizarán como referencia los códigos de las prácticas de la asignatura, que servirán de base para desarrollar el resto del código. Por lo tanto, los aspectos relacionados con el diseño de máquinas de estado, la sincronización, la detección de flancos y el manejo de los displays seguirán una estructura similar a la presentada en estas prácticas.

El funcionamiento de la máquina expendedora comienza en un estado inicial en el que se debe seleccionar el tipo de refresco, ya sea el tipo 1 o el tipo 2. Antes de introducir monedas, se debe confirmar la intención de proceder con el pago. A continuación, se

introducen las monedas mientras el display indica la cantidad restante por depositar. Esta funcionalidad está a cargo tanto del contador como del controlador del display.

Si se introduce la cantidad exacta de dinero, aparecerá un mensaje indicando que el refresco seleccionado ha sido dispensado (OUT). Si la cantidad ingresada es superior al precio del producto, se mostrará el cambio devuelto y las monedas serán devueltas. Finalmente, se regresa al estado inicial.

Tenemos habilitada la opción de volver al estado inicial, en caso de equivocación del consumidor, al presionar el botón de RESET.

2.2. Entradas y salidas

2.2.1 Entradas

1. *CLK (Reloj)*
    - Señal de reloj principal de la placa con una frecuencia de 100 MHz.
    - Todas las señales del sistema, excepto la señal RESET, están sincronizadas con este reloj.

2. *RESET (Reinicio)*
    - Señal de reinicio del sistema que retorna la máquina a su estado inicial.
    - Es una señal asíncrona respecto al reloj principal.

3. *TIPO_MONEDA (Vector de Entradas)*
    - Vector de 4 bits que indica la moneda introducida en la máquina. Cada combinación corresponde a un valor específico:
        - "0001": Moneda de 10 céntimos (10c).
        - "0010": Moneda de 20 céntimos (20c).
        - "0100": Moneda de 50 céntimos (50c).
        - "1000": Moneda de 1 euro (1€).

4. *PAGO (Confirmación de Pago)*
    - Señal de control que confirma la selección del producto y da inicio al proceso de pago.
    - Una vez activada, se verifica si el importe introducido es suficiente y se procede a la entrega del refresco.

5. *TIPO_REFRESCO (Selección del Producto)*
    - Vector de 2 bits que permite elegir el refresco deseado. Las opciones disponibles son:
        - "01": Selección del Refresco 1. Su precio será de 1,00€
        - "10": Selección del Refresco 2. Su precio será de 1,30€
    - Cualquier otra combinación de bits se considera no válida.

2.2.2 Salidas

1. *PRODUCTO_OK (Entrega del Producto)*
    - Señal conectada a otro LED que confirma la entrega del refresco seleccionado.
    - Esta salida se activa cuando se ha introducido el importe exacto para el refresco.

2. *ESTADOS (Vector de Estados)*
    - Vector de señales que representa el estado actual de la máquina.
    - Su función es puramente informativa y permite el seguimiento del flujo de ejecución del sistema.

3. *DIGCTRL (Control de Displays de 7 Segmentos)*
    - DIGCTRL: Señal de control que activa el display específico que debe mostrar la información proporcionada.

4. *SEGMENTOS*
    - SEGMENTOS: Salida que contiene la información del número o letra que se mostrará en los displays de 7 segmentos.

Ambas señales (DIGCTRL y SEGMENTOS) operan a una frecuencia de reloj reducida respecto al CLK principal, gracias al prescaler incorporado en el diseño.

Notas Adicionales:

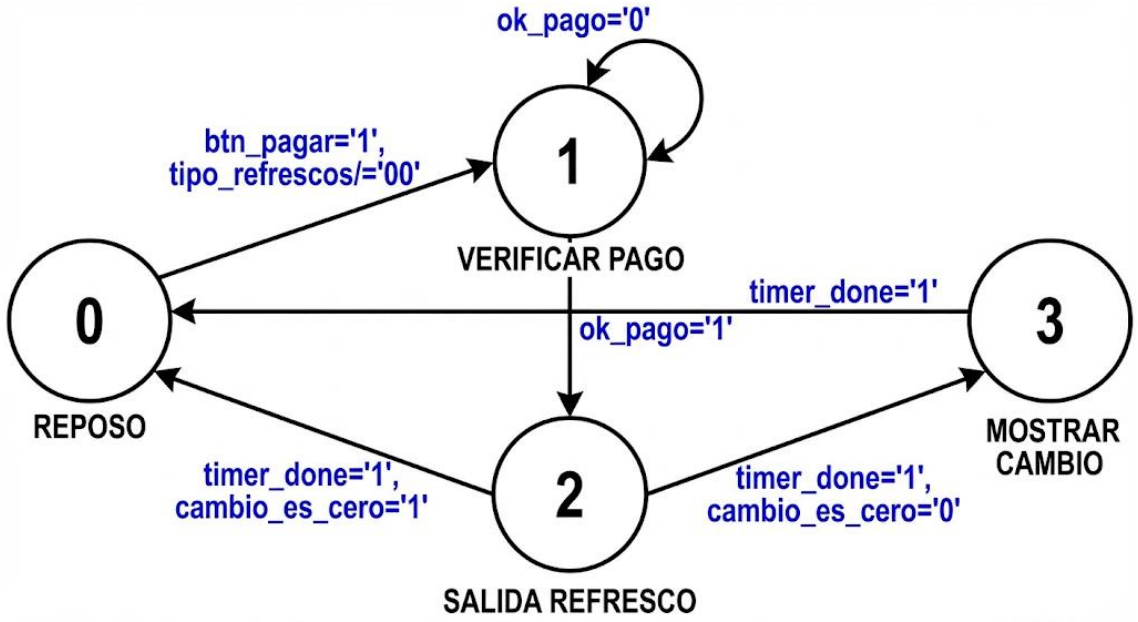
- El sistema está diseñado para operar de manera sincronizada, excepto por el RESET, que funciona de manera asíncrona.

- El control de pagos y selección de productos se basa en la correcta combinación de las señales TIPO_MONEDA, PAGO y TIPO_REFRESCO.

- Las salidas proporcionan retroalimentación visual a través de LEDs y displays de 7 segmentos, facilitando la interacción con el usuario.

2.3. Diagrama de estados

El diagrama de estados lo podemos ver en la siguiente hoja. En este sistema tenemos 4 estados principales: Reposo, Verificar Pago, Salida de Refresco y Mostrar Cambio.



Primero, en el estado de **Reposo** (ST_IDLE), la máquina está inactiva a la espera de que el usuario seleccione un producto y pulse el botón de pagar. En el display se muestra cuál es el refresco que está seleccionado actualmente. Este es el estado inicial y también es al que siempre regresaremos si se pulsa el botón RESET, sin importar en qué etapa nos encontremos.

Después, al entrar en la etapa de **Verificar Pago** (ST_CHECK_FUNDS), el sistema espera a que el módulo contador confirme que el dinero introducido es suficiente. Aquí, el contador va sumando el importe de las monedas y lo compara con el precio. Dependiendo de eso, el contador manda la señal de control.

- "ok_pago" se activa únicamente cuando el importe introducido es igual o superior al precio del refresco.

Al no haber etapa de error, la máquina simplemente permanecerá en este estado esperando más monedas hasta que se alcance el precio. La señal que se envía determinará el paso al siguiente estado.

- Cuando la señal es "ok_pago", pasaremos automáticamente a la etapa de Salida de Refresco.

En la etapa de **Salida de Refresco** (ST_DISPENSING), se activa el mecanismo para entregar el producto (simulado con un LED) y se inicia un temporizador. Al finalizar este tiempo, el sistema toma una decisión basada en si sobra dinero o no.

- Si “cambio_es_cero” es '1' (Pago exacto): La máquina vuelve directamente a Reposo.
- Si “cambio_es_cero” es '0' (Hay cambio): La máquina pasa a la etapa de Mostrar Cambio.

En la etapa de **Mostrar Cambio** (ST_SHOW_CHANGE), el display indica al usuario la cantidad que se le devuelve. Tras un tiempo de espera, la máquina se reinicia automáticamente.

Por último, siempre existe la opción de volver al estado de **Reposo** de dos maneras:

- Automáticamente, al terminar de servir el refresco (si el pago fue exacto) o tras mostrar el cambio (si sobró dinero).
- Pulsando el botón RESET, que siempre devuelve la máquina al estado inicial y reinicia las variables.

Además de todo esto, en cada estado, el display proporciona información importante:

- En Reposo se muestra el número del refresco seleccionado.
- En Verificar Pago se indica cuánto dinero falta por introducir para alcanzar el precio.
- En Salida de Refresco: Informa visualmente de que el producto se está sirviendo.
- En Mostrar Cambio: Indica la cantidad exacta de dinero devuelto al usuario.

3. DESARROLLO DE ENTIDADES Y SIMULACIÓN

3.1. Esquema de identidades

Una vez establecidos los objetivos y especificaciones que debe cumplir el proyecto, procederemos a planificar la estructura que adoptará nuestro programa para llevar a cabo adecuadamente cada una de sus funcionalidades.

El diseño del programa se desarrollará de manera que incorpore diversas entidades que interactuarán de forma coordinada, intercambiando datos a través de variables auxiliares que enlazarán sus entradas y salidas. Asimismo, cada entidad será responsable de ejecutar un conjunto de tareas distintivas que detallaremos a continuación.

En primer lugar, encontramos las entidades SYNCHRNZR y EDGE_DETECTOR, que, tal como hemos observado en las prácticas, se ocupan respectivamente de sincronizar las entradas con el reloj y de identificar los cambios de estado o flancos. Su utilización resulta fundamental para minimizar posibles errores durante el proceso de lectura.

Por otro lado, contamos con tres entidades principales que estructuran el programa: COUNTER, DISPLAY_CONTROL y FSM. La primera de ellas se encarga de contabilizar las monedas introducidas y de verificar que el importe depositado coincide con el solicitado, y en caso contrario, devolver el cambio. Mediante la entidad DISPLAY_CONTROL gestionaremos la visualización en los displays de la placa, ajustándola dinámicamente conforme el usuario avanza en la selección y el pago del producto. Finalmente, la entidad FSM actúa como el núcleo de coordinación del sistema, dirigiendo el funcionamiento de la máquina basándose en el diagrama de estados previamente diseñado.

El DISPLAY_CONTROL está vinculado a otra entidad, denominada DECODER, cuya función es activar los segmentos necesarios para representar, en los displays, los números o letras transmitidos desde la unidad de control.

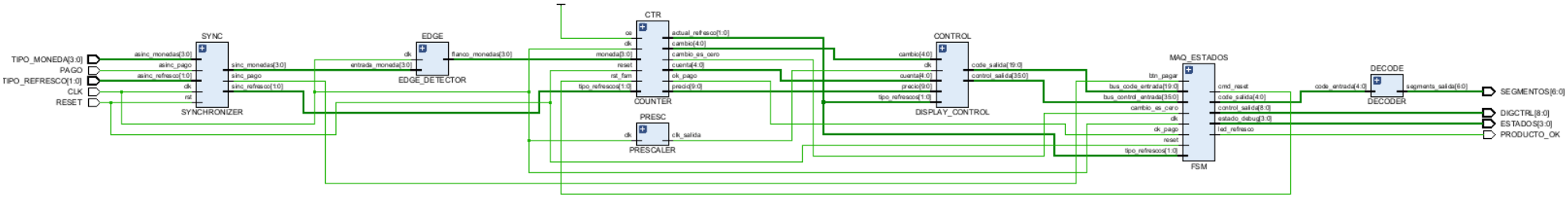
El PRESCALER actúa como un divisor de frecuencia que reduce la alta velocidad del reloj del sistema para generar una señal de control más lenta y adecuada para el refresco de los displays. Su función es alternar el encendido de los dígitos a la velocidad exacta para que, gracias a la persistencia de la visión, el ojo humano perciba los números como una imagen fija y estable, evitando parpadeos.

Además, disponemos de una entidad superior llamada MAQUINA_EXP, cuya función principal es invocar a las distintas entidades con sus correspondientes parámetros y definir las variables auxiliares necesarias para su interconexión.

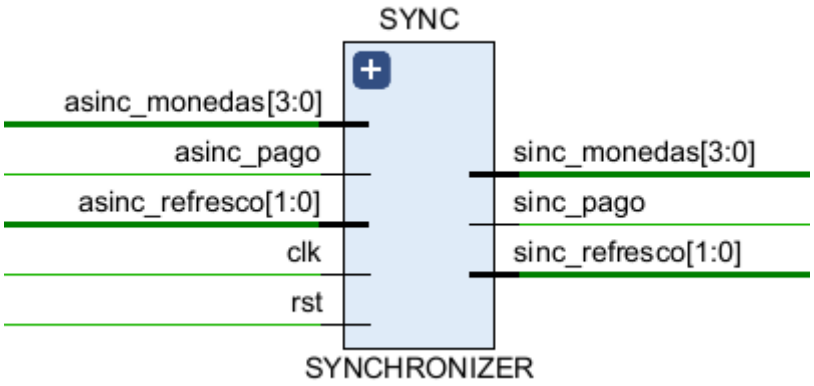
Una vez comprendido el rol de cada una de las entidades, pasaremos a analizar el flujo de ejecución del programa, utilizando como referencia tanto el diagrama de estados como el de entidades.

El sistema de la máquina expendedora comienza en un estado de reposo. Se activa al detectar, por medio de la FSM, que la entrada de PAGO ha sido activada y se ha seleccionado una de las dos opciones de refresco disponibles. Esto nos lleva al COUNTER, que registrará las monedas insertadas. Simultáneamente, gracias a DISPLAY_CONTROL, los displays mostrarán el producto seleccionado, su precio, la cantidad restante para completar el pago y el cambio. El contador comunicará su estado a la FSM mediante la señal OK_PAGO.

En la siguiente página se puede observar el diagrama de entidades.


TIPO_MONEDA[3:0]
PAGO
TIPO_REFRESCO[1:0]
CLK
RESET
SYNC
asinc_monedas[3:0]
asinc_pago
asinc_refresco[1:0]
clk
rst
sinc_monedas[3:0]
sinc_pago
sinc_refresco[1:0]
SYNCHRONIZER
EDGE
clk
entrada_moneda[3:0]
flanco_monedas[3:0]
EDGE_DETECTOR
CTR
oe
clk
moneda[3:0]
reset
rst_fsm
tipo_refrescos[1:0]
actual_refresco[1:0]
cambio[4:0]
cambio_es_cero
cuenta[4:0]
ok_pago
precio[9:0]
COUNTER
PRESC
clk
clk_salida
PRESCALER
CONTROL
cambio[4:0]
clk
cuenta[4:0]
precio[9:0]
tipo_refrescos[1:0]
code_salida[19:0]
control_salida[35:0]
DISPLAY_CONTROL
MAQ_ESTADOS
btn_pagar
bus_code_entrada[19:0]
bus_control_entrada[35:0]
cambio_es_cero
clk
ok_pago
reset
tipo_refrescos[1:0]
cmd_reset
code_salida[4:0]
control_salida[8:0]
estado_debug[3:0]
led_refresco
FSM
DECODE
code_entrada[4:0]
segments_salida[6:0]
DECODER
SEGMENTOS[6:0]
DIGCTRL[8:0]
ESTADOS[3:0]
PRODUCTO_OK

3.2. SYNCHRONIZER



Cuando las señales asíncronas se emplean directamente en la lógica sincronizada con el reloj del sistema, pueden producirse fallos en la lógica digital, lo que resalta la relevancia de esta entidad.

El SYNCHNZR desempeña un papel crucial para garantizar el funcionamiento fiable y consistente de la máquina expendedora, proporcionando al usuario una experiencia segura y uniforme. Su tarea principal consiste en recibir señales externas que no están sincronizadas con el reloj del sistema y alinearlas correctamente. Este proceso minimiza el riesgo de condiciones metaestables, en las que una señal podría quedar en un estado indefinido entre '0' y '1', lo que podría ocasionar un comportamiento impredecible del sistema.

En la definición de esta entidad se incluyen dos parámetros genéricos: NUM_REFRESCOS y NUM_MONEDAS, que corresponden a la cantidad de tipos de refrescos ofrecidos y a las distintas denominaciones de monedas aceptadas, como se explicó anteriormente.

Las entradas de la entidad reflejan las decisiones del usuario: las monedas introducidas están representadas por el vector asinc_monedas [3:0]; el bit asinc_pago adopta un nivel alto cuando el usuario presiona el botón de pagar, y la selección del tipo de refresco se transmite mediante la señal asinc_refresco [1:0].

En la arquitectura de la entidad, se declaran tres señales en un único proceso, El funcionamiento se basa en una cadena de dos registros (Flip-Flops) disparados por el flanco de subida del reloj:

1. Primera Etapa (ff1): Captura el valor de las señales asíncronas de entrada (asinc_monedas, asinc_pago, etc.). Esta etapa asume el riesgo inicial de inestabilidad.
2. Segunda Etapa (ff2): Un ciclo de reloj después, captura el valor estabilizado de la primera etapa.

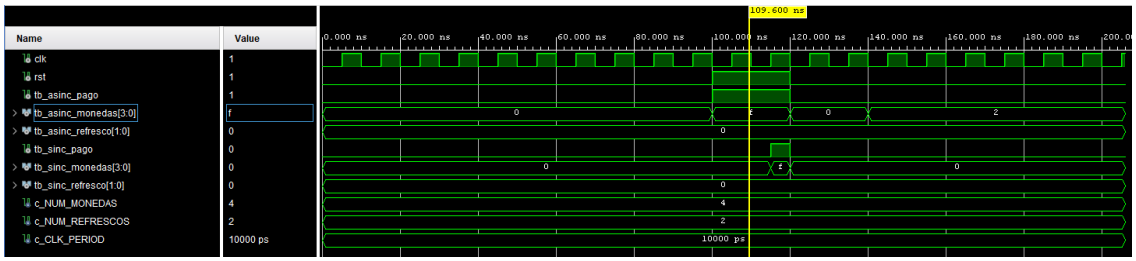
El resultado final (ff2_monedas, ff2_pago, etc.) son señales limpias y perfectamente sincronizadas con el reloj global, garantizando que los módulos posteriores (como el detector de flancos y la FSM) operen con datos estables y fiables.

```
p_synchronization: process(clk, rst)
begin
    if rst = '0' then
        ff1_monedas <= (others => '0');
        ff2_monedas <= (others => '0');
        ff1_pago <= '0';
        ff2_pago <= '0';
        ff1_refrescos <= (others => '0');
        ff2_refrescos <= (others => '0');

    elsif rising_edge(clk) then
        ff1_monedas <= asinc_monedas;
        ff1_pago <= asinc_pago;
        ff1_refrescos <= asinc_refresco;

        ff2_monedas <= ff1_monedas;
        ff2_pago <= ff1_pago;
        ff2_refrescos <= ff1_refrescos;
    end if;
end process;
```

Finalmente, las señales estabilizadas en los registros secundarios se transfieren a las salidas de la entidad. Estas señales están sincronizadas y ahora pueden ser utilizadas de manera segura por el resto de la lógica de la máquina expendedora.

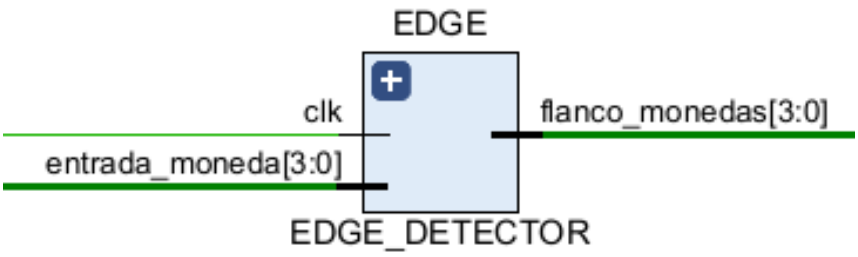
```
sinc_monedas <= ff2_monedas;
sinc_pago <= ff2_pago;
sinc_refresco <= ff2_refrescos;
```

TESTBENCH:



En la imagen, se muestra un diagrama de ondas que resulta de la simulación del Testbench. Este comportamiento es exactamente lo que esperarías de un sincronizador en un entorno digital, lo que demuestra que la entidad SYNCHRNZR está operando correctamente dentro del contexto de la simulación proporcionada.

3.3. EDGE_DETECTOR



La función principal del EDGE_DETECTOR es identificar transiciones en los estados de las señales digitales del sistema. En este caso, se encarga de detectar cambios en las variables de entrada cuando ocurre un flanco ascendente o descendente en cualquiera de las señales. Sin esta detección precisa de bordes, las señales podrían interpretarse de manera incorrecta o pasar desapercibidas, lo que podría ocasionar un funcionamiento irregular o errores en el sistema.

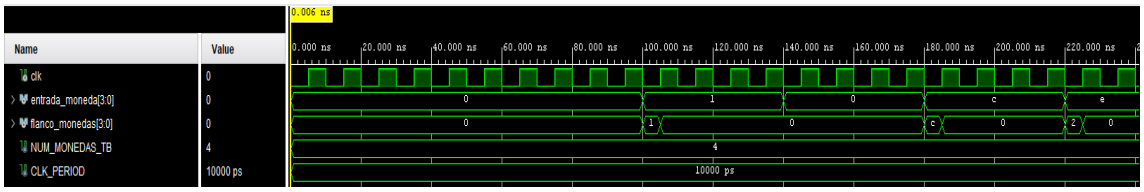
El EDGE_DETECTOR garantiza que cada moneda sea contabilizada una única vez, independientemente de cuánto tiempo la señal permanezca activa después de la inserción. Esto evita errores en el registro de créditos, asegurando que la máquina expendedora funcione de manera precisa y justa.

```
process(clk)
begin
    if rising_edge (clk) then
        moneda_anterior <= entrada_moneda;
    end if;
end process;
```

Este proceso tiene la función de almacenar el estado previo de la señal de entrada para crear una memoria temporal. Al ejecutarse en cada flanco de subida del reloj (rising_edge(clk)), guarda el valor actual de entrada_moneda en la señal auxiliar moneda_anterior, generando efectivamente un retraso de un ciclo de reloj. Esta copia retardada es vital para el funcionamiento del detector, ya que permite comparar simultáneamente el valor presente con el valor pasado inmediato para identificar cuándo ocurre exactamente una transición de '0' a '1' (flanco de subida).

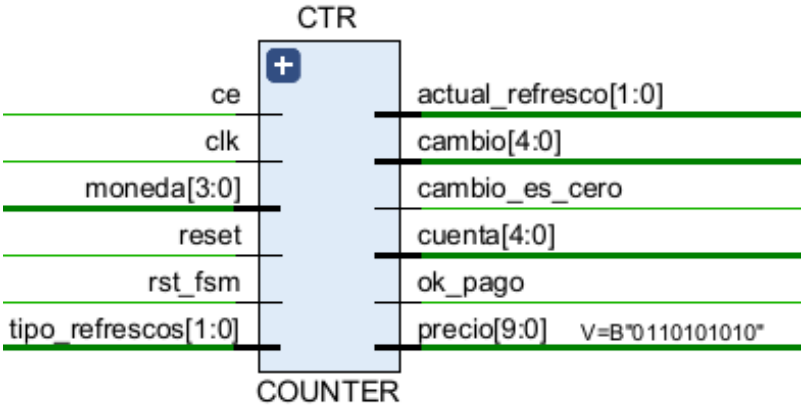
Esta entidad es un componente indispensable en la interfaz de usuario de la máquina expendedora, permitiendo una interacción precisa y fiable entre el usuario y la máquina. Su implementación garantiza que las señales de entrada sean procesadas correctamente, lo cual es esencial para el funcionamiento adecuado y la confiabilidad del sistema en general.

TESTBENCH:



Se muestra el diagrama de ondas del Testbench para la entidad EDGE_DETECTOR. La función es identificar los cambios en las señales digitales, típicamente de '0' a '1' (flanco ascendente) o de '1' a '0' (flanco descendente). La salida del EDGE_DETECTOR muestra reacciones a los cambios en la entrada.

3.4. COUNTER



La entidad COUNTER es responsable de gestionar y verificar que se ha ingresado la cantidad de dinero necesaria para adquirir el refresco seleccionado. La máquina cuenta con dos opciones de refresco, cada una con un precio diferente: el primero tiene un precio de 1€ y el segundo, 1,30€.

Para implementar esta entidad, disponemos de la entrada tipo_refrescos, que es un vector de 2 elementos y representa las dos opciones de refresco. Además, mediante la entrada moneda (un vector de 4 elementos), se introduce el dinero en la máquina. Cada bit del vector moneda representa un tipo de moneda: "0001" para 10 céntimos, "0010" para 20 céntimos, "0100" para 50 céntimos y "1000" para 1€. También cuenta con una entrada de habilitación ce, equivalente a la acción de "pagar" en el diagrama de bloques, y un reset.

En cuanto a las salidas, tenemos ok_pago, que se activa cuando la cantidad ingresada coincide con el precio del refresco seleccionado. Si se excede el monto necesario, se activa la salida cambio. Por su parte, la salida cuenta almacena la cantidad total de dinero introducida por el usuario, actualizándose conforme se suman las monedas. Adicionalmente, utilizamos las variables actual_refresco y precio para registrar el tipo de refresco elegido y su precio correspondiente, y así pasar esta información a la FSM.

Usaremos las distintas señales para completar todas las funciones del contador.

```
type ARRAY_PRECIOS_TYPE is ARRAY (0 to NUM_REFRESCOS - 1) of unsigned(TAM_CUENTA - 1 downto 0);
constant LISTA_PRECIOS_VAL: ARRAY_PRECIOS_TYPE := (to_unsigned(10, TAM_CUENTA), to_unsigned(13, TAM_CUENTA));

signal r_cuenta: unsigned(TAM_CUENTA - 1 downto 0) := (others => '0');
signal r_error: std_logic := '0';
signal r_ok_pago: std_logic := '0';
signal r_cambio: unsigned(TAM_CUENTA - 1 downto 0) := (others => '0');

signal r_refresco_sel: std_logic_vector(NUM_REFRESCOS - 1 downto 0) := (others => '0');
signal moneda_prev: std_logic_vector(NUM_MONEDAS - 1 downto 0) := (others => '0');
signal precio_objetivo: unsigned(TAM_CUENTA - 1 downto 0);
```

Para facilitar la implementación, definimos un array llamado ARRAY_PRECIOS_TYPE, que contiene los precios de los refrescos. Los valores se asignan a la señal LISTA_PRECIOS_VAL, donde LISTA_PRECIOS_VAL(0) equivale al valor binario "01010" (10 decimal, que representa 1,00€), y también LISTA_PRECIOS_VAL(1) corresponde al valor binario "01101" (13 decimal, que representa 1,30€). Asimismo, utilizamos la señal r_cuenta, que acumula el total de dinero ingresado en la máquina.

A continuación, se muestra el proceso que sigue el contador:

```
begin
    process(clk, reset)
    begin
        if reset = '0' then
            r_cuenta <= (others => '0');
            r_refresco_sel <= (others => '0');
            r_error <= '0';
            r_ok_pago <= '0';
            r_cambio <= (others => '0');
            moneda_prev <= (others => '0');

        elsif rising_edge(clk) then
            if ce = '1' then

                moneda_prev <= moneda;

                if rst_fsm = '1' then
                    r_cuenta <= (others => '0');
                    r_refresco_sel <= (others => '0');
                    r_error <= '0';
                    r_ok_pago <= '0';
                    r_cambio <= (others => '0');

                else
                    r_refresco_sel <= tipo_refrescos;
```

```
                if (moneda(0) = '1' and moneda_prev(0) = '0') then
                    if r_cuenta <= 30 then r_cuenta <= r_cuenta + 1; end if;
                end if;
                if (moneda(1) = '1' and moneda_prev(1) = '0') then
                    if r_cuenta <= 29 then r_cuenta <= r_cuenta + 2; end if;
                end if;
                if (moneda(2) = '1' and moneda_prev(2) = '0') then
                    if r_cuenta <= 26 then r_cuenta <= r_cuenta + 5; end if;
                end if;
                if (moneda(3) = '1' and moneda_prev(3) = '0') then
                    if r_cuenta <= 21 then r_cuenta <= r_cuenta + 10; end if;
                end if;

                if precio_objetivo > 0 then
                    if r_cuenta >= precio_objetivo then
                        r_ok_pago <= '1';
                        r_error <= '0';
                        r_cambio <= r_cuenta - precio_objetivo;
                    else
                        r_ok_pago <= '0';
                        r_error <= '0';
                        r_cambio <= (others => '0');
                    end if;
                else
                    r_ok_pago <= '0';
                    r_error <= '0';
                end if;
            end if;
        end if;
    end if;
end process;
```

Cuando la entrada de habilitación ce está desactivada, la señal r_cuenta se inicializa en "0000". Al activarse ce, ya no se puede modificar la selección del refresco, y la máquina comienza a aceptar monedas. La señal r_cuenta acumula en binario el valor de las monedas introducidas y, al completar el proceso, verifica si la cantidad coincide con el precio del refresco seleccionado.

```
process(r_refresco_sel)
begin
    precio_objetivo <= (others => '0');
    if r_refresco_sel = "01" then precio_objetivo <= LISTA_PRECIOS_VAL(0);
    elsif r_refresco_sel = "10" then precio_objetivo <= LISTA_PRECIOS_VAL(1);
    end if;
end process;
```

Si el monto acumulado en r_cuenta es mayor o igual al precio del refresco (determinado cuando r_refresco_sel es "01" o "10"), se activa la salida r_ok_pago para indicar que la transacción es válida. En caso de que el importe supere el precio, el sistema calcula automáticamente la diferencia y la almacena en la señal r_cambio para su posterior devolución, en lugar de activar la señal de error. Si la cantidad acumulada no es suficiente, la máquina mantiene r_ok_pago en '0' y sigue esperando más monedas. Este

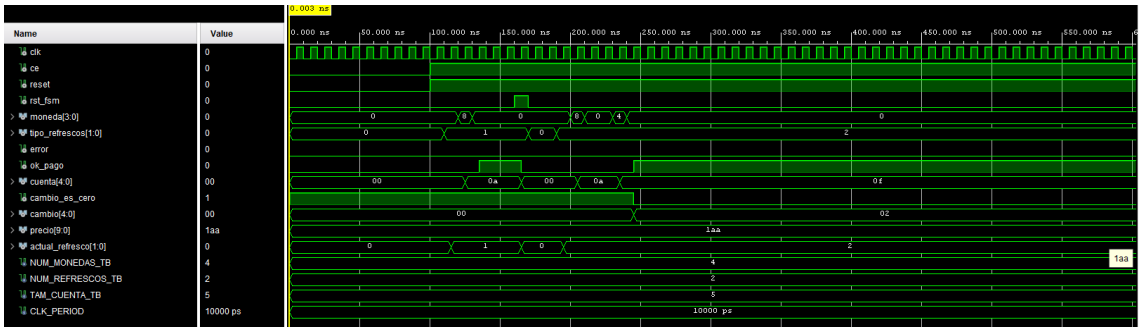
procedimiento se realiza comparando dinámicamente el valor de r_cuenta con la señal precio_objetivo, la cual obtiene su valor de la constante LISTA_PRECIOS_VAL.

Finalmente, las señales se conectan a las salidas de la entidad para compartir la información con los otros bloques del sistema.

```
error <= r_error;
ok_pago <= r_ok_pago;
cuenta <= std_logic_vector(r_cuenta);
actual_refresco <= r_refresco_sel;
precio <= std_logic_vector(LISTA_PRECIOS_VAL(1)) & std_logic_vector(LISTA_PRECIOS_VAL(0));
cambio <= std_logic_vector(r_cambio);

cambio_es_cero <= '1' when (r_cambio = 0) else '0';
```

Esta entidad es fundamental para gestionar el funcionamiento de la máquina expendedora, garantizando que los usuarios puedan comprar refrescos según los precios establecidos, mientras se emiten indicadores claros de error o éxito. Además, la entidad está diseñada de manera flexible, permitiendo añadir nuevas opciones de refrescos y precios con solo ampliar la lista, sin necesidad de modificar la estructura base del código.

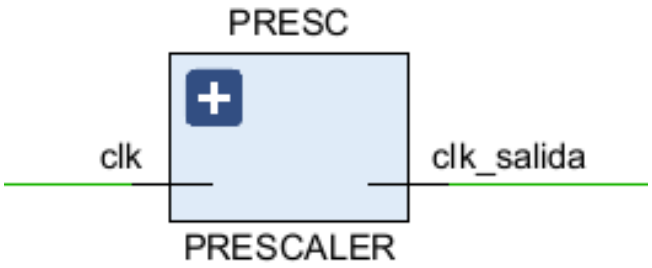
TESTBENCH:



Al ejecutar el testbench, se verificó el correcto funcionamiento de la entidad COUNTER. Cuando la entrada ce está activada y tras liberar el reset, el sistema queda listo para procesar pagos. En el primer caso (ver imagen), seleccionamos el refresco con precio de 1,00€ (valor interno 10). Al introducir las monedas necesarias hasta que la variable cuenta alcanza exactamente este valor, la salida ok_pago se activa, manteniendo cambio en 0.

Posteriormente, se activa rst_fsm para reiniciar la cuenta y se selecciona el segundo refresco (1,30€, valor interno 13). En esta segunda prueba, simulamos un pago superior al precio (acumulando un valor de 15, equivalente a 1,50€). Al superar el precio objetivo, la salida ok_pago se activa nuevamente, pero la diferencia clave es que el sistema calcula la devolución: la señal cambio toma el valor de 2 (20 céntimos) y la bandera cambio_es_cero pasa a nivel bajo, confirmando que la lógica de cambio funciona correctamente.

3.5. PRESCALER



Un componente del sistema, el display, no puede operar a la frecuencia del reloj principal, por lo que es necesario ajustar esta frecuencia. El objetivo de esta entidad es recibir una señal de reloj de entrada con alta frecuencia y reducirla a un nivel que sea adecuado para los componentes del sistema que necesitan trabajar con una frecuencia más baja para funcionar eficientemente.

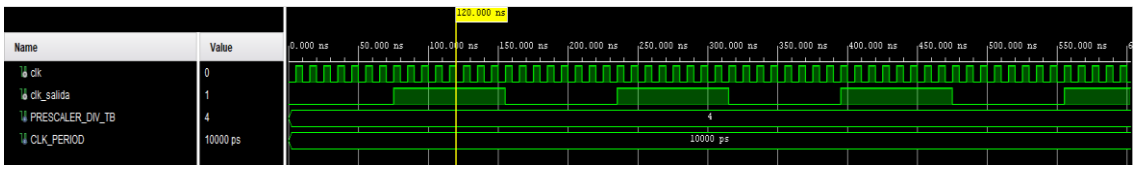
El PRESCALER funciona como un contador que incrementa su valor con cada pulso del reloj de entrada. Al alcanzar un valor predeterminado, el contador se reinicia y genera un pulso en la señal de salida del reloj. Por ejemplo, si se configura para un valor de 100, el PRESCALER producirá un pulso de salida por cada 100 pulsos de entrada, reduciendo así la frecuencia del reloj de entrada a una centésima parte.

El procedimiento para dividir la frecuencia de la señal de reloj se detalla a continuación:

```
process (clk)
begin
    if rising_edge (clk) then
        s_counter <= s_counter + 1;
    end if;
end process;
clk_salida <= std_logic(s_counter(s_counter'high));
```

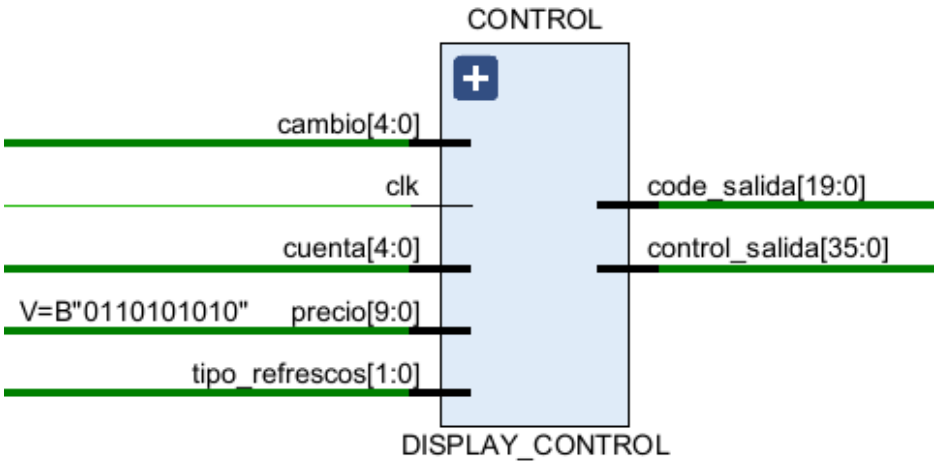
Este proceso implementa la división de frecuencia mediante un contador binario de carrera libre. En cada flanco de subida del reloj principal, se incrementa el valor de la señal s_counter. La reducción de frecuencia se logra asignando a la salida clk_salida únicamente el bit más significativo (MSB) del contador (s_counter'high). Dado que la frecuencia de conmutación se reduce a la mitad con cada bit sucesivo, seleccionar el bit más alto genera una señal de reloj estable y lo suficientemente lenta para controlar correctamente el refresco de los displays.

TESTBENCH:



La señal de reloj de salida cambia de estado con menor frecuencia que la señal de reloj de entrada. Esto corresponde al comportamiento esperado de un prescaler, cuyo propósito es disminuir la frecuencia de la señal de reloj original.

3.6. DISPLAY_CONTROL



Esta entidad es el núcleo de la interfaz visual del proyecto, encargada de gestionar qué información se muestra en los displays de 7 segmentos en función del estado actual de la máquina. Recibe como entradas el reloj de refresco (proveniente del PRESCALER), la cuenta actual acumulada, el tipo_refrescos seleccionado, el precio y el cambio calculado por el contador. Genera dos vectores de salida: code_salida (caracteres a decodificar) y control_salida (selección de ánodos), que guardan la información de todos los estados simultáneamente. Posteriormente, la FSM seleccionará qué bloque de información se envía físicamente a los displays.

A nivel de implementación, se utilizan dos arrays internos: s_control y s_code. Cada uno contiene 4 vectores correspondientes a los 4 estados de la máquina (ST_IDLE, ST_CHECK_FUNDS, ST_DISPENSING, ST_SHOW_CHANGE). Esto se ilustra en la siguiente imagen:

```
type T_CONTROL_ARRAY is array (0 to NUM_ESTADOS - 1) of std_logic_vector(NUM_DISPLAYS - 1 downto 0);
type T_CODE_ARRAY is array(0 to NUM_ESTADOS - 1) of std_logic_vector(TAM_CODE - 1 downto 0);

signal s_control : T_CONTROL_ARRAY := (others => (others => '1'));
signal s_code    : T_CODE_ARRAY := (others => (others => '0'));
```

El proceso principal se ejecuta en cada flanco de subida del reloj, dentro de este proceso, se actualiza cíclicamente el dígito activo para cada uno de los 4 estados de forma independiente. A continuación, se detalla la lógica para cada caso.

Cada elemento de la señal de control opera de forma independiente y gestiona un estado específico. En cada sentencia case, se actualiza el siguiente dígito junto con su código correspondiente. Por ejemplo, en el estado de pago, uno de los dígitos que requiere cálculo es el primer decimal de la cantidad restante por pagar (1.x0 €). Por ejemplo, si el precio del refresco es 1.50€ y se ingresan 40 céntimos, el display debe mostrar un 1 (indica que faltan 1.10€ por pagar). Si se ingresan 20 céntimos adicionales, el valor cambiará a 9 (indicando que faltan 0.90€). El cálculo de este dígito se detalla a continuación:

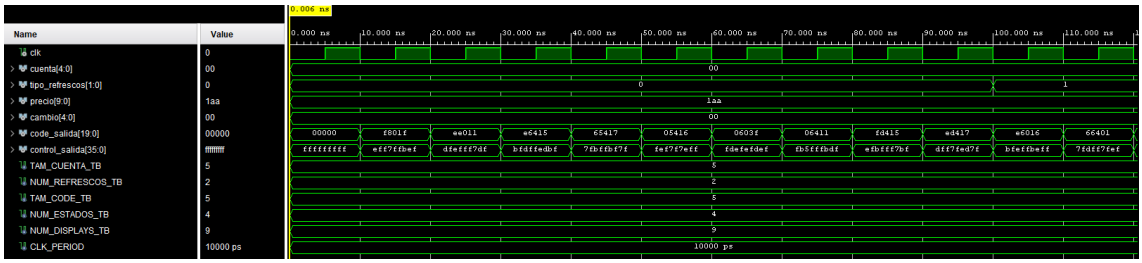
```
        when "111111011" =>
            s_control(1) <= "111110110";
            if tipo_refrescos = "01" then
                if precio(TAM_CUENTA - 1 downto 0) - cuenta >= "01010" then
                    s_code(1) <= CHAR_1;
                else s_code(1) <= CHAR_0;
                end if;
            elsif tipo_refrescos = "10" then
                if precio((TAM_CUENTA*2) - 1 downto TAM_CUENTA) - cuenta >= "01010" then
                    s_code(1) <= CHAR_1;
                else s_code(1) <= CHAR_0;
                end if;
            end if;

        when "111110110" =>
            s_control(1) <= "111011111";
            s_code(1) <= CHAR_0;

        when "111011111" =>
            s_control(1) <= "110111111";
            if tipo_refrescos = "01" then
                s_code(1) <= precio(TAM_CUENTA - 1 downto 0) - "01010";
            elsif tipo_refrescos = "10" then
                s_code(1) <= precio((TAM_CUENTA*2) - 1 downto TAM_CUENTA)- "01010";
            end if;

        when "110111111" =>
            s_control(1) <= "101111110";
            s_code(1) <= CHAR_1;

        when others =>
        s_control(1) <= "111111101";
        s_code(1) <= CHAR_0;
    end case;
```

Dado que los precios seleccionados están por debajo de los 2€, cuando es necesario determinar si el dinero restante está por debajo de 1€, o si han cambiado las unidades de euro, verificamos si el precio del refresco menos el dinero ingresado es mayor o igual a 10 (equivalente a 1€). Cada refresco se evalúa en función de su precio correspondiente. En realidad, los demás estados, al ser más descriptivos y contener letras, son más sencillos de implementar. Por ejemplo, en el estado donde el refresco se entrega exitosamente, solo mostramos "OUT n", donde n es el número del refresco comprado.

Finalmente, las salidas de la entidad control_salida y code_salida son el resultado de concatenar los elementos de los arrays de control y código.

```
control_salida <= s_control(3) & s_control(2) & s_control(1) & s_control(0);
code_salida    <= s_code(3)    & s_code(2)    & s_code(1)    & s_code(0);
```

TESTBENCH:

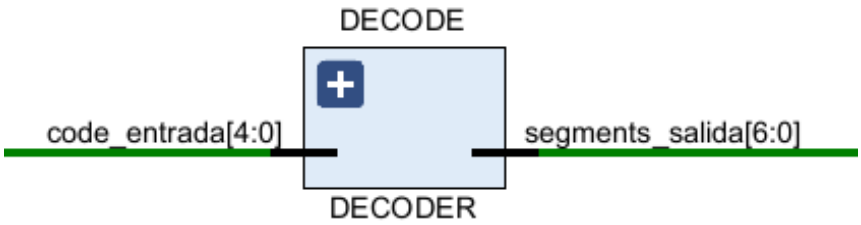


La simulación de esta entidad resulta bastante compleja debido a la cantidad de flancos de reloj necesarios para actualizar cada dígito. Además, se generan vectores de salida muy extensos. Dado que la entidad genera las señales para los 4 estados a la vez, en el cronograma se puede observar cómo rotan los bits de control (ánodos) y cambian los códigos de caracteres en cada ciclo de reloj clk.

Por ejemplo, para el Bloque 3 de ver el cambio, si la entrada cambio es "00010" (20 céntimos), el testbench muestra cómo los códigos de salida para ese bloque alternan entre los caracteres de texto "CHG" y los números '0' (unidades) y '2' (decimales), validando que la lógica de visualización del cambio funciona correctamente.

Otro ejemplo, en el primer estado se encarga de mostrar en pantalla "prod 1" o "prod 2" dependiendo del refresco seleccionado. En este caso, el refresco permanece como 1 y, en cada ciclo de reloj, se actualiza el dígito activo en ese momento. Observamos que en el quinto dígito se codifica un 1. En los demás dígitos, se representan las letras ‘d’, ‘o’, ‘r’ y ‘p’. En el decoder se puede comprobar el código que corresponde a cada letra.

3.7. DECODER



La entidad DECODER tiene como propósito interpretar señales codificadas y transformarlas en señales o acciones específicas. Este decodificador "traduce" un conjunto de entradas en un conjunto de salidas.

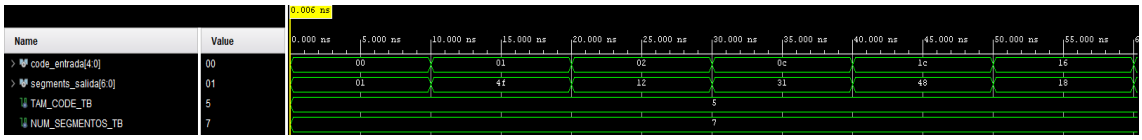
La entidad DECODER utiliza dos parámetros genéricos: TAM_CODE, que representa la longitud del vector de entrada (cantidad de bits en la entrada), y NUM_SEGMENTOS, que indica el tamaño del vector de salida (cantidad de segmentos a controlar). Code_entrada es la entrada del decodificador, mientras que segments_salida son las salidas encargadas de activar los segmentos del display.

```
case code_entrada is

    when "00000" => segments_salida <= "0000001";
    when "00001" => segments_salida <= "1001111";
    when "00010" => segments_salida <= "0010010";
    when "00011" => segments_salida <= "0000110";
    when "00100" => segments_salida <= "1001100";
    when "00101" => segments_salida <= "0100100";
    when "00110" => segments_salida <= "0100000";
    when "00111" => segments_salida <= "0001111";
    when "01000" => segments_salida <= "0000000";
    when "01001" => segments_salida <= "0000100";

    when "01100" => segments_salida <= "0110001";

    when "11100" => segments_salida <= "1001000";
    when "11101" => segments_salida <= "0100000";

    when "10000" => segments_salida <= "0001000";
    when "10001" => segments_salida <= "1000010";
    when "10010" => segments_salida <= "0110000";
    when "10011" => segments_salida <= "0111000";
    when "10100" => segments_salida <= "1110001";
    when "10101" => segments_salida <= "1100010";
    when "10110" => segments_salida <= "0011000";
    when "10111" => segments_salida <= "1111010";
    when "11000" => segments_salida <= "1110000";
    when "11001" => segments_salida <= "1100011";

    when others  => segments_salida <= "1111110";
end case;
```

En este proceso se implementa un decodificador combinacional que actúa como interfaz entre la lógica interna y el usuario. Su función es traducir los códigos binarios de 5 bits (code_entrada), generados por el controlador de displays, a los patrones de 7 bits necesarios (segments_salida) para iluminar los segmentos correspondientes en el display físico. El diseño utiliza una estructura case que funciona como una tabla de consulta (LUT), mapeando tanto dígitos numéricos como caracteres alfabéticos personalizados, y está configurado para operar con lógica de ánodo común (activa a nivel bajo), donde un '0' lógico enciende el segmento correspondiente.

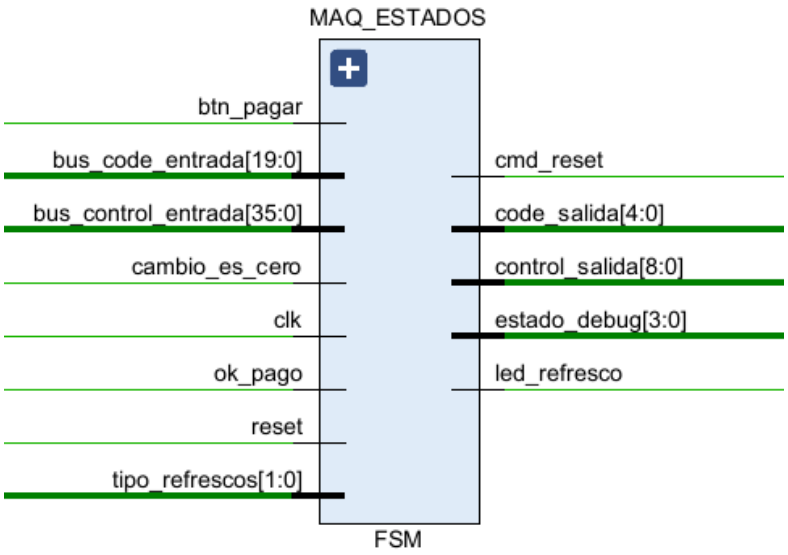
El decodificador está diseñado para manejar tanto números como algunas letras, lo que permite mostrar distintos caracteres en el display de siete segmentos.

TESTBENCH:



Finalmente, se verifica que cada combinación de entrada en code_entrada genera la salida esperada en segments_salida. Esto asegura que cada código se está interpretando correctamente, convirtiéndose en el conjunto adecuado de señales que activarán los segmentos del display para mostrar el número o carácter deseado.

3.8. FSM



La entidad FSM, como indica su nombre, representa la máquina de estados que define el control del sistema. Desde aquí se regula el cambio entre los cuatro estados definidos, en función de las entradas y salidas que los conectan, tal como se establece en el diagrama de estados inicial.

Esta entidad trabaja de manera conjunta con el contador y el display_control, ya que sus entradas provienen de las salidas de estas entidades. Las salidas del contador utilizadas son: cambio_es_cero (cuando se ingresa la cantidad exacta), ok_pago y actual_refresco (refresco seleccionado).

En primer lugar, se definen los cuatro estados de la máquina y se inicializa la variable current_state, que pertenece al tipo t_state comenzando siempre desde el estado de reposo (ST_IDLE).

```
type t_state is (ST_IDLE, ST_CHECK_FUNDS, ST_DISPENSING, ST_SHOW_CHANGE);
signal current_state, next_state : t_state;
```

A partir de aquí, podemos dividir la entidad en dos procesos principales: p_state_reg y p_comb.

```
p_state_reg: process(reset, clk)
begin
    if reset = '0' then
        current_state <= ST_IDLE;
        timer_counter <= 0;
    elsif rising_edge(clk) then
        current_state <= next_state;

        if current_state /= next_state then
            timer_counter <= 0;
        elsif (current_state = ST_DISPENSING or current_state = ST_SHOW_CHANGE) then
            if timer_counter < TIEMPO_ESPERA then
                timer_counter <= timer_counter + 1;
            end if;
        else
            timer_counter <= 0;
        end if;
    end if;
end process;
```

El primer proceso, p_state_register, tiene un funcionamiento simple: se implementa la lógica secuencial de la máquina de estados. Su función principal es actualizar el registro de estado (current_state) en cada ciclo de reloj, transfiriendo el valor calculado por la lógica combinacional. Además, integra un temporizador interno que se activa exclusivamente durante los estados de ST_DISPENSING y ST_SHOW_CHANGE.

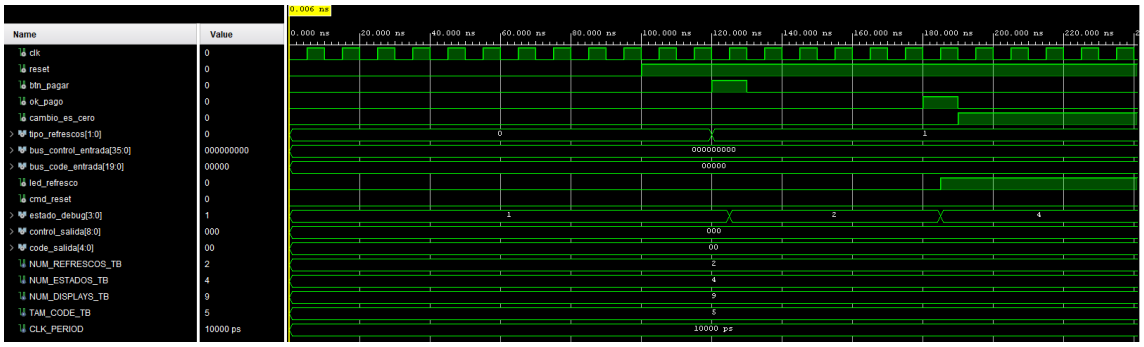
Este temporizador incluye una lógica de reinicio automático al detectar transiciones de estado, asegurando que la cuenta del tiempo de espera comience siempre desde cero al ingresar en dichas etapas, garantizando así la precisión en los tiempos de dispensado y visualización del cambio.

```
p_comb: process(current_state, btn_pagar, ok_pago, cambio_es_cero, tipo_refrescos, bus_control_entrada, bus_code_entrada, timer_done)
    variable v_mux_sel : integer range 0 to 3;
begin
    next_state <= current_state;
    led_refresco <= '0';
    cmd_reset <= '0';
    estado_debug <= "0000";
    v_mux_sel := 0;

    case current_state is
        when ST_IDLE =>
            estado_debug <= "0001";
            v_mux_sel := IDX_IDLE;
            if btn_pagar = '1' and tipo_refrescos /= "00" then
                next_state <= ST_CHECK_FUNDS;
            end if;

        when ST_CHECK_FUNDS =>
            estado_debug <= "0010";
            v_mux_sel := IDX_CHECK;
            if ok_pago = '1' then
                next_state <= ST_DISPENSING;

            end if;

        when ST_DISPENSING =>
            estado_debug <= "0100";
            led_refresco <= '1';
            v_mux_sel := IDX_DISPENSE;
```

```
            if timer_done = '1' then
                if cambio_es_cero = '1' then
                    cmd_reset <= '1';
                    next_state <= ST_IDLE;
                else
                    next_state <= ST_SHOW_CHANGE;
                end if;
            end if;

        when ST_SHOW_CHANGE =>
            estado_debug <= "1000";
            led_refresco <= '0';
            v_mux_sel := IDX_CHANGE;

            if timer_done = '1' then
                cmd_reset <= '1';
                next_state <= ST_IDLE;
            end if;
    end case;

    control_salida <= bus_control_entrada( (NUM_DISPLAYS * (v_mux_sel + 1)) - 1 downto (NUM_DISPLAYS * v_mux_sel) );
    code_salida    <= bus_code_entrada(    (TAM_CODE     * (v_mux_sel + 1)) - 1 downto (TAM_CODE     * v_mux_sel) );
end process;
end Behavioral;
```

Este proceso implementa la lógica combinacional de salida y transición de la FSM. Su función es evaluar las entradas y el estado actual para determinar, de manera asíncrona, el siguiente estado (next_state) al que debe evolucionar el sistema. Además, controla las señales de salida directas, como el comando de reset del sistema. Finalmente, actúa como un multiplexor de datos para los displays, seleccionando qué bloque de información (bus_control_entrada y bus_code_entrada) se visualiza en los 7 segmentos (precio, mensaje de salida o cambio) en función de la etapa activa en cada momento.

TESTBENCH:



La simulación inicia en el estado de **Reposo**. Al activar la señal btn_pagar y seleccionar un tipo de refresco válido, la máquina avanza al estado de **Verificar Pago**. El sistema permanece en espera hasta que se activa la señal de entrada ok_pago (indicando que el dinero es suficiente), lo que provoca la transición inmediata al estado de **Dispensación**. En esta etapa, se verifica el encendido de la señal led_refresco. Al finalizar el temporizador interno (timer_done = '1'), se comprueba la lógica de bifurcación:

1. Caso de Pago Exacto: Si la señal cambio_es_cero está activa ('1'), la máquina regresa automáticamente al estado de Reposo y activa cmd_reset para limpiar contadores.
2. Caso con Devolución**:** Si, por el contrario, simulamos que sobra dinero (cambio_es_cero = '0'), la FSM transita al estado de **Mostrar Cambio**. Tras

esperar el tiempo definido por el temporizador en este nuevo estado, el sistema vuelve finalmente al Reposo, completando el ciclo.

3.9. MAQ_EXP (TOP)

La entidad TOP representa el nivel más alto en la jerarquía del diseño del sistema. Su función principal es integrar y coordinar todas las subentidades y módulos que forman parte del diseño.

El objetivo de esta entidad es actuar como el punto central del sistema. En ella se interconectan todas las señales de entrada y salida, además de instanciarse los distintos componentes del diseño. TOP se encarga de gestionar cómo interactúan las señales entre los diversos módulos y asegura que el flujo de datos sea el adecuado para que la máquina expendedora funcione correctamente.

TOP proporciona una perspectiva clara y estructurada del proyecto. Funciona como el elemento lógico que une todos los módulos individuales, asegurando que el sistema opere de manera integrada y cohesiva.

4. GITHUB

https://github.com/Sergius842/Trabajo-SED