TRABAJO SED
VHDL

GRUPO Nº. 15

MÁQUINA EXPENDEDORA

INTEGRANTES

FRANCISCO MARTÍN CUSCURITA 55972

JIMENA LÓPEZ MALDONADO 55951

JOAN BELLIDO INES 55745

ÍNDICE

1.- INTRODUCCIÓN Y OBJETIVOS

1.1.- Enunciado

En esta sección del proyecto de la asignatura se llevará a cabo la implementación de un diseño en la placa de laboratorio NEXYS 4 DDR que resuelva uno de los enunciados propuestos en Moodle para los estudiantes. En nuestro caso, se ha elegido el problema de la máquina expendedora. El enunciado de la propuesta de diseño 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 para:
- Botón de reset con el que se vuelva al estado inicial.

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 se describe de la siguiente manera: se 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. Si la cantidad es incorrecta, se mostrará un mensaje de error y las monedas serán devueltas. Finalmente, se regresa al estado inicial

al presionar el botón de reinicio o, si la operación se ha completado correctamente, al establecer el valor de pagar en ‘0’ nuevamente.

2.2.- Entradas y salidas

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. **MONEDAS (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. **PAGAR (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.

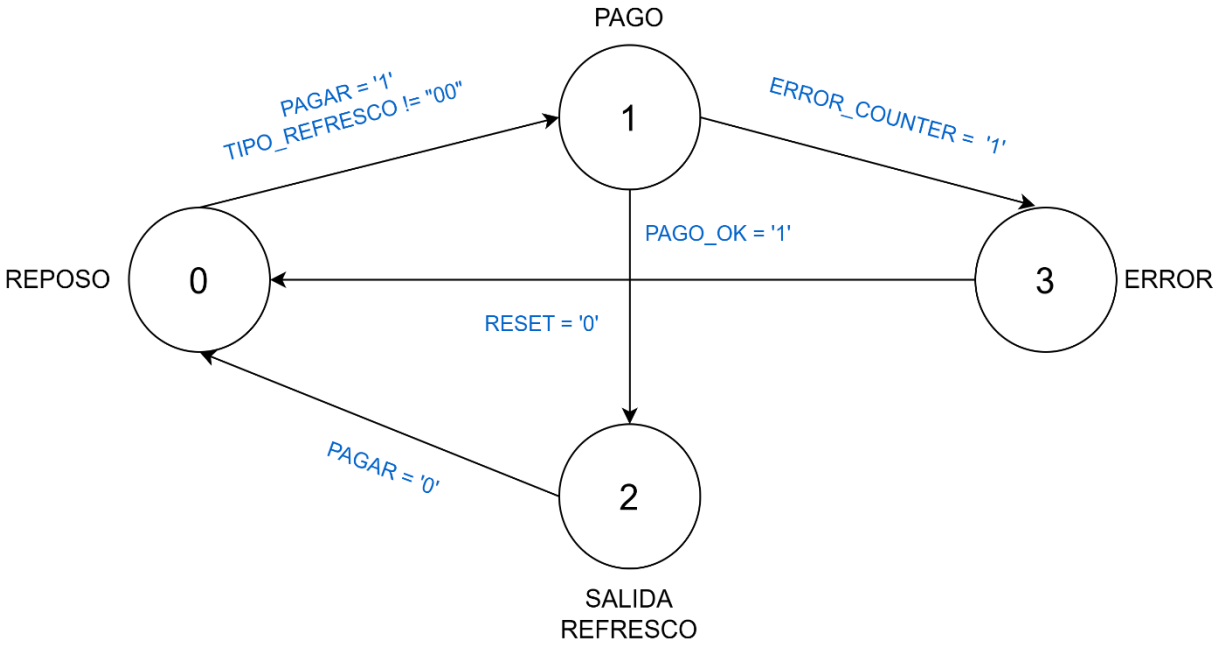
Salidas

1. **ERROR (Indicador de Error)**
    - Señal conectada a **un LED de la placa**.
    - Se activa si el importe introducido **excede el precio del refresco seleccionado**, indicando un error en la operación.
2. **REFRESCO_OUT (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.
3. **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.
4. **SEGMENTOS y DIGCTRL (Control de Displays de 7 Segmentos)**
    - **SEGMENTOS**: Salida que contiene la **información del número o letra** que se mostrará en los displays de 7 segmentos.
    - **DIGCTRL**: Señal de control que **activa el display específico** que debe mostrar la información proporcionada.
    - Ambas señales 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 MONEDAS, PAGAR 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



La máquina de estados tiene cuatro estados principales: Reposo, Etapa de Pago, Etapa de Error y Etapa de Salida de Refresco.

Primero, en el **estado de Reposo**, la máquina está inactiva, pero en el **display** se muestra cuál es el refresco que está seleccionado. 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 Pago**, el usuario tiene que introducir las monedas para pagar el refresco. Aquí, un **contador** va sumando el importe que se va introduciendo y calcula si es correcto o no. Dependiendo de eso, el contador manda una señal:

- **PAGO_OK** si el importe introducido es exacto.
- **ERROR_COUNTER** si se pasa del importe permitido o hay algún error.

La señal que se envía determinará el **próximo estado**:

- Si la señal es **PAGO_OK**, pasaremos a la **Etapa de Salida de Refresco**.
- Si la señal es **ERROR_COUNTER**, pasaremos a la **Etapa de Error**.

En la **Etapa de Error**, el **display** avisará al usuario de que ha habido un error, mientras que, en la **Etapa de Salida de Refresco**, el **display** mostrará que el refresco ha sido entregado correctamente.

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

1. Automáticamente, dependiendo del estado en el que nos encontremos.
2. Pulsando el botón **RESET**, que siempre devuelve la máquina al estado inicial.

Además, en cada estado, el **display** proporciona información importante:

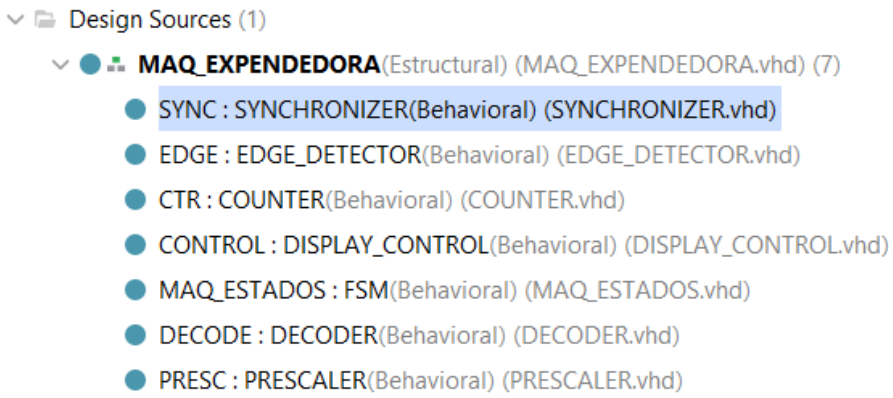
- En **Reposo**, muestra el refresco seleccionado.
- En **Pago**, indica el precio total y la cantidad de dinero que falta por introducir.
- En **Salida de Refresco**, informa que el refresco ha salido.
- En **Error**, avisa de que ha ocurrido un problema.

3.- DESARROLLO DE ENTIDADES Y SIMULACIÓN

2.1.- 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. 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.

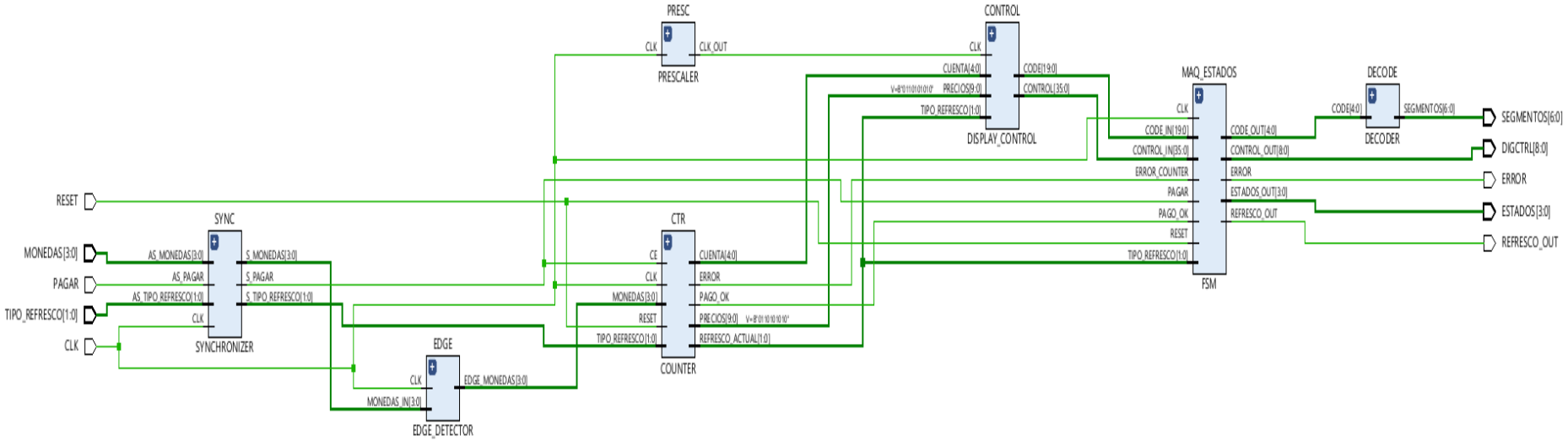
El **DISPLAY_CONTROL** está vinculado a otra entidad, denominada **DECODER**, cuya función es activar los segmentos necesarios para representar, en los displays, los números o letras transmitidos desde la unidad de control.

Además, disponemos de una entidad superior llamada **MAQ_EXP**, que no aparece reflejada en el diagrama, pero cuya función principal es invocar a las distintas entidades con sus correspondientes parámetros y definir las variables auxiliares necesarias para su interconexión.

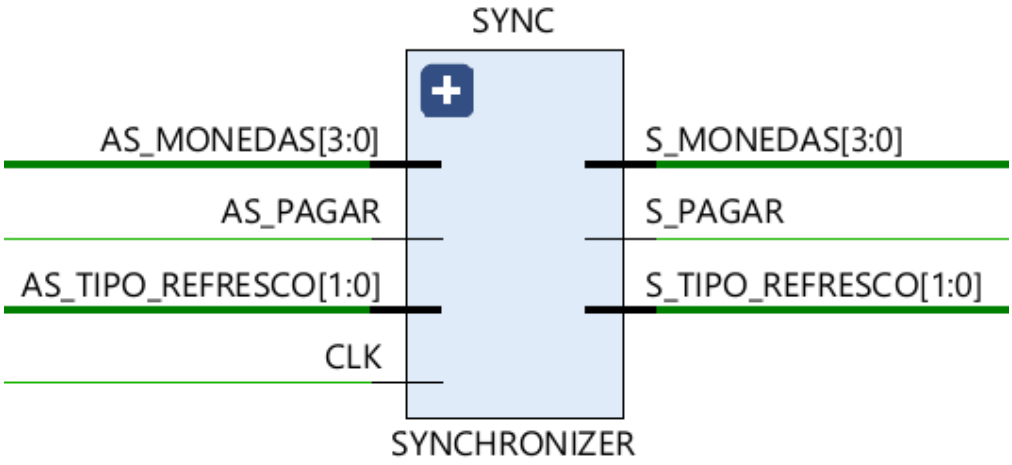
Una vez comprendido el rol de cada una de las entidades, pasaremos a analizar el flujo de ejecución del programa, utilizando como referencia tanto el diagrama de estados como el de entidades.

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 **PAGAR** 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 y la cantidad restante para completar el pago. El contador comunicará su estado a la **FSM** mediante las señales **PAGO_CORRECTO** y **ERROR**, proporcionando a esta última la información necesaria para determinar si el producto está listo para ser dispensado.

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


RESET
MONEDAS[3:0]
PAGAR
TIPO_REFRESCO[1:0]
CLK
SYNC
AS_MONEDAS[3:0]
AS_PAGAR
AS_TIPO_REFRESCO[1:0]
CLK
S_MONEDAS[3:0]
S_PAGAR
S_TIPO_REFRESCO[1:0]
SYNCHRONIZER
EDGE
CLK
MONEDAS_IN[3:0]
EDGE_MONEDAS[3:0]
EDGE_DETECTOR
PRESC
CLK
CLK_OUT
PRESCALER
CTR
CE
CLK
MONEDAS[3:0]
RESET
TIPO_REFRESCO[1:0]
CUENTA[4:0]
ERROR
PAGO_OK
PRECIOS[9:0]
REFRESCO_ACTUAL[1:0]
COUNTER
CONTROL
CLK
CUENTA[4:0]
PRECIOS[9:0]
TIPO_REFRESCO[1:0]
CODE[19:0]
CONTROL[35:0]
DISPLAY_CONTROL
MAQ_ESTADOS
CLK
CODE_IN[19:0]
CONTROL_IN[35:0]
ERROR_COUNTER
PAGAR
PAGO_OK
RESET
TIPO_REFRESCO[1:0]
CODE_OUT[4:0]
CONTROL_OUT[8:0]
ERROR
ESTADOS_OUT[3:0]
REFRESCO_OUT
FSM
DECODE
CODE[4:0]
SEGMENTOS[6:0]
DECODER
SEGMENTOS[6:0]
DIGCTRL[8:0]
ERROR
ESTADOS[3:0]
REFRESCO_OUT

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 **AS_MONEDAS [3:0]**; el bit **AS_PAGAR** adopta un nivel alto cuando el usuario presiona el botón de pagar, y la selección del tipo de refresco se transmite mediante la señal **S_TIPO_REFRESCO**.

En la arquitectura de la entidad, se declaran tres señales que, en un primer proceso, capturan los valores de sus respectivas entradas en cada flanco positivo del reloj. Posteriormente, un segundo proceso transfiere estas señales a un segundo registro en el siguiente borde ascendente del reloj. Este enfoque contribuye a reducir aún más el riesgo de condiciones metaestables, ya que permite que cualquier señal que haya quedado en un estado indefinido durante la primera captura sea estabilizada antes de llegar a la lógica principal del sistema.

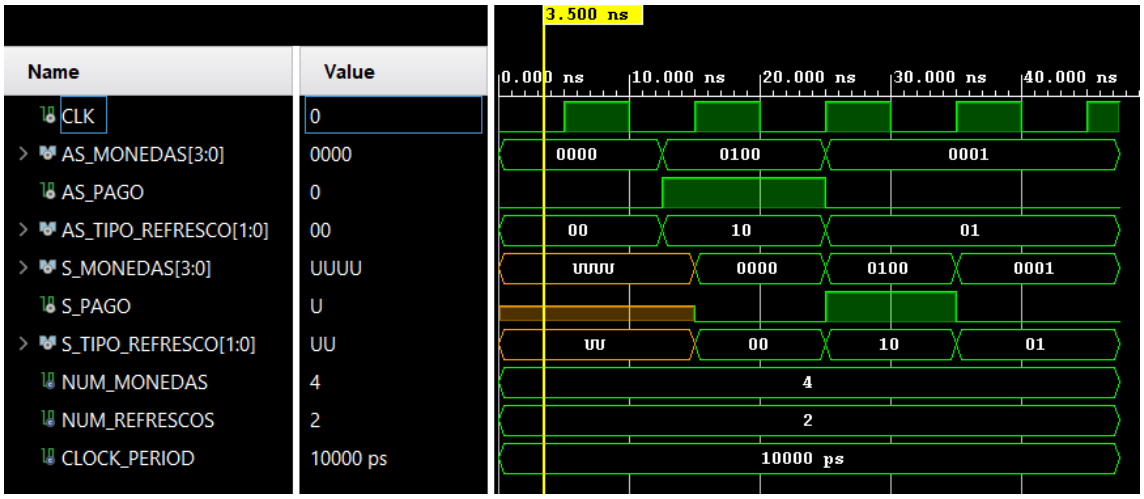
```
REGISTRO_1:process(CLK)
begin
    if rising_edge(CLK) then
        REG_1_MONEDAS <= AS_MONEDAS;
        REG_1_PAGO <= AS_PAGO;
        REG_1_TIPO_REFRESCO <= AS_TIPO_REFRESCO;
    end if;
end process;

REGISTRO_2:process(CLK)
begin
    if rising_edge(CLK) then
        REG_2_MONEDAS <= REG_1_MONEDAS;
        REG_2_PAGO <= REG_1_PAGO;
        REG_2_TIPO_REFRESCO <= REG_1_TIPO_REFRESCO;
    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.

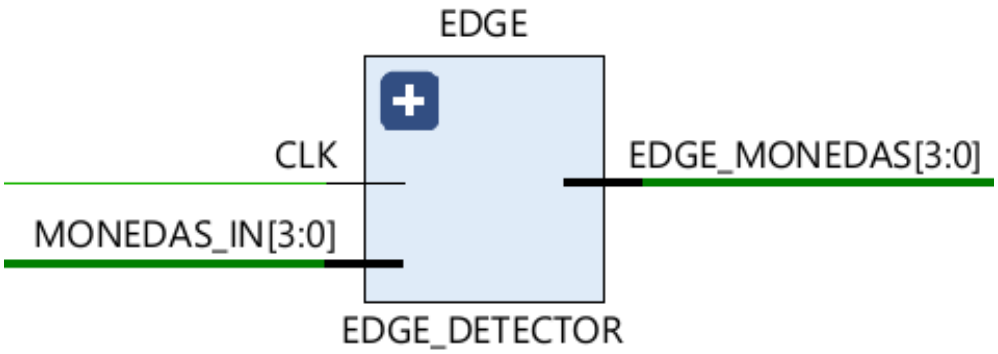
```
--Salidas sincronizadas
S_MONEDAS <= REG_2_MONEDAS;
S_PAGO <= REG_2_PAGO;
S_TIPO_REFRESCO <= REG_2_TIPO_REFRESCO;
```

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.

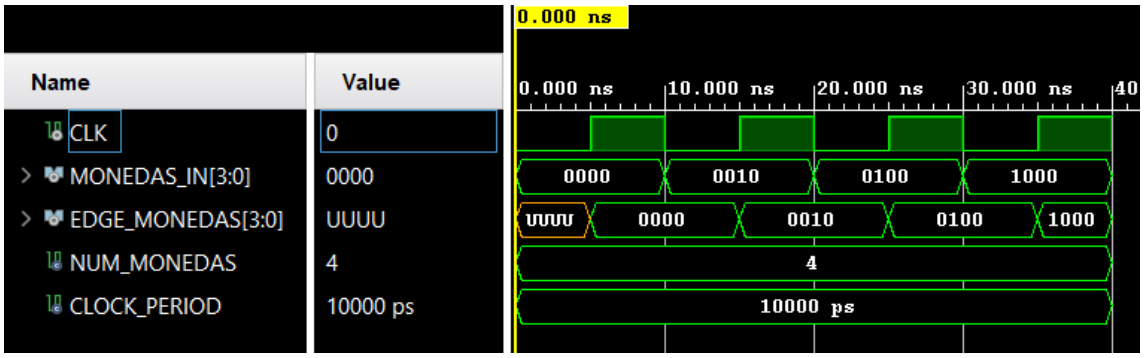
El **EDGE_DETECTOR** garantiza que cada moneda sea contabilizada una única vez, independientemente de cuánto tiempo la señal permanezca activa después de la inserción. Esto evita errores en el registro de créditos, asegurando que la máquina expendedora funcione de manera precisa y justa.

```
process(CLK)
begin
    if rising_edge (CLK) then
      if MONEDAS_IN /= MONEDAS_ANTERIORES then
          EDGE_MONEDAS <= MONEDAS_IN;
      else
          EDGE_MONEDAS <= (others => '0');
      end if;
      MONEDAS_ANTERIORES <= MONEDAS_IN;
    end if;
end process;
```

Dentro de este proceso, que se activa en cada borde ascendente de la señal de reloj **CLK**, se verifica si el estado actual de las entradas **MONEDAS_IN** es diferente al estado previamente almacenado en **EDGE_MONEDAS**. Si hay una diferencia, esto indica que se ha detectado un cambio de estado o un flanco en alguna de las entradas de monedas.

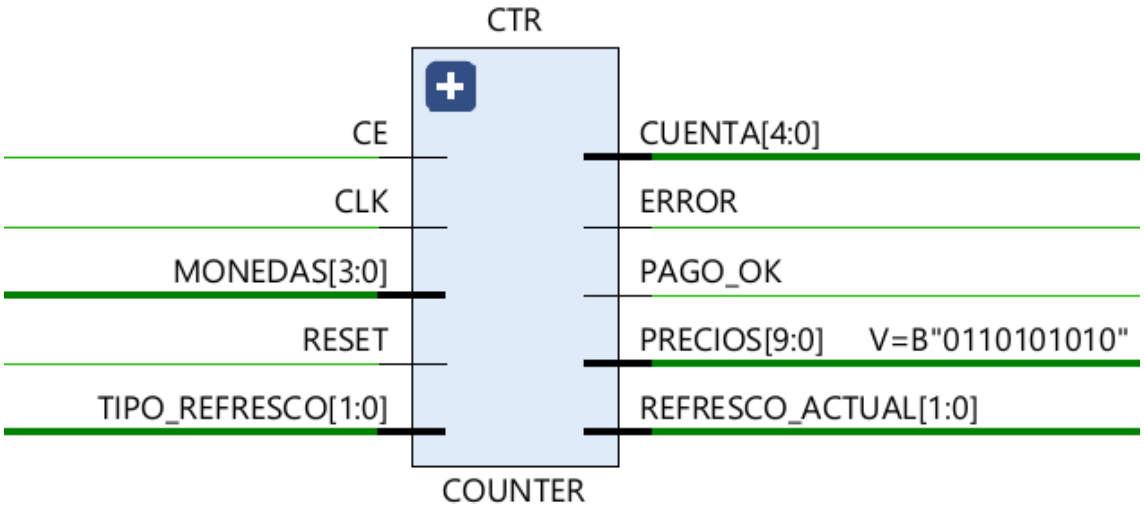
Esta entidad es un componente indispensable en la interfaz de usuario de la máquina expendedora, permitiendo una interacción precisa y fiable entre el usuario y la máquina. Su implementación garantiza que las señales de entrada sean procesadas correctamente, lo cual es esencial para el funcionamiento 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_REFRESCO**, que es un vector de 2 elementos y representa las dos opciones de refresco. Además, mediante la entrada **MONEDAS** (un vector de 4 elementos), se introduce el dinero en la máquina. Cada bit del vector **MONEDAS** 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 **PAGO_OK**, que se activa cuando la cantidad ingresada coincide con el precio del refresco seleccionado. Si se excede el monto necesario, se activa la salida **ERROR**. 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 **REFRESCO_ACTUAL** y **PRECIOS** 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.

```
--Array que contiene el precio de los refrescos
type ARRAY_PRECIOS is ARRAY (0 to NUM_REFRESCOS - 1) of std_logic_vector(TAM_CUENTA - 1 downto 0);

--Señales de registro
signal REG_CUENTA: std_logic_vector(TAM_CUENTA - 1 downto 0) := (others => '0');
signal REG_ERROR: std_logic;
signal REG_PAGO_OK: std_logic;
signal LISTA_PRECIOS: ARRAY_PRECIOS := ("01010","01101");
signal REG_REFRESCO_ACTUAL: std_logic_vector(NUM_REFRESCOS - 1 downto 0);
```

Para facilitar la implementación, definimos un array llamado **ARRAY_PRECIOS**, que contiene los precios de los refrescos. Los valores se asignan a la señal **LISTA_PRECIOS**, donde **LISTA_PRECIOS(0)** equivale al valor binario "01010" (1€), y **LISTA_PRECIOS(1)** corresponde al valor binario "01101" (1,30€). Asimismo, utilizamos la señal **CUENTA_SIG**, que acumula el total de dinero ingresado en la máquina.

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

```
process(CLK, CE, RESET)
begin
        if RESET = '1' then
            REG_CUENTA <= (others => '0');  --Se reinicia la cuenta a 0euros
            REG_REFRESCO_ACTUAL <= TIPO_REFRESCO; --Se reinicia el tipo de refresco seleccionado
        elsif rising_edge(CLK) then
            if CE = '1' then
                if REG_REFRESCO_ACTUAL /= "00" then
                    --Suma Monedas
                    if MONEDAS = "0001" then
                        REG_CUENTA <= REG_CUENTA + "00001";  -- +10centimos
                    elsif MONEDAS = "0010" then
                        REG_CUENTA <= REG_CUENTA + "00010";  -- +20centimos
                    elsif MONEDAS = "0100" then
                        REG_CUENTA <= REG_CUENTA + "00101";  -- +50centimos
                    elsif MONEDAS = "1000" then
                        REG_CUENTA <= REG_CUENTA + "01010";  -- +1euro
                    end if;
                end if;
            end if;
        end if;
```

Cuando la entrada de habilitación **CE** está desactivada, la señal **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 **CUENTA_SIG** 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.

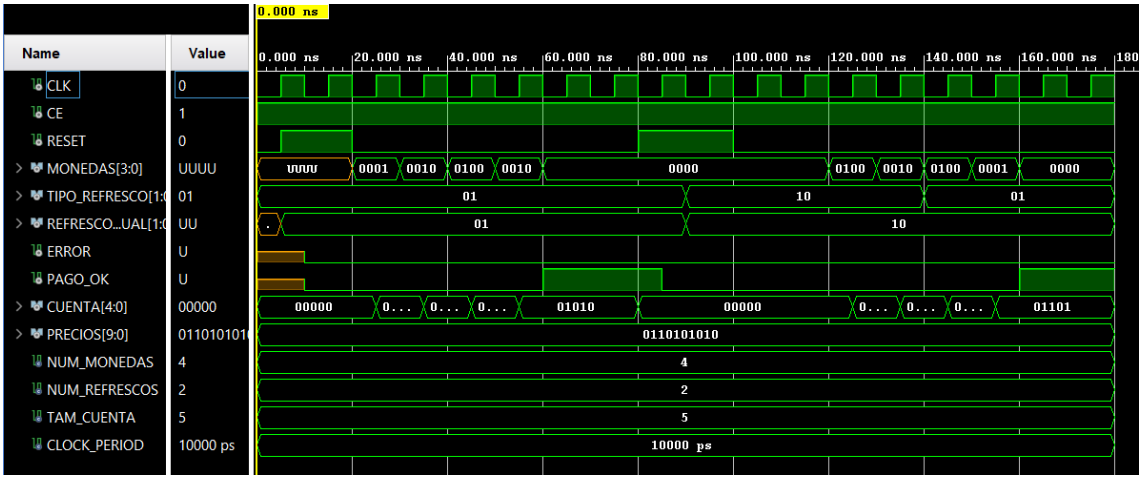
```
if REG_REFRESCO_ACTUAL = "01" then
    if REG_CUENTA > LISTA_PRECIOS(0) then
        REG_ERROR <= '1';  --Se ha introducido mas de 1€
        REG_PAGO_OK <= '0';
    elsif REG_CUENTA = LISTA_PRECIOS(0) then
        REG_PAGO_OK <= '1'; --Se ha introducido la cantidad correcta
        REG_ERROR <= '0';
    elsif REG_CUENTA < LISTA_PRECIOS(0) then
        REG_PAGO_OK <= '0';
        REG_ERROR <= '0'; --No se ha introducido la cantidad suficiente
    end if;
```

Si el monto acumulado es igual al precio del refresco (**REFRESCO_ACTUAL** = "01" o "10" según el caso), se activa la salida **PAGO_OK**; si lo supera, se activa la señal **ERROR**; y si no es suficiente, la máquina sigue esperando más monedas. Este mismo procedimiento se aplica a ambos refrescos, comparando la cantidad acumulada con los valores en **LISTA_PRECIOS**. Finalmente, las señales se conectan a las salidas de la entidad para compartir la información con los otros bloques del sistema.

```
ERROR <= REG_ERROR;
PAGO_OK <= REG_PAGO_OK;
CUENTA <= REG_CUENTA;
PRECIOS <= LISTA_PRECIOS(1) & LISTA_PRECIOS(0);
REFRESCO_ACTUAL <= REG_REFRESCO_ACTUAL;
```

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:

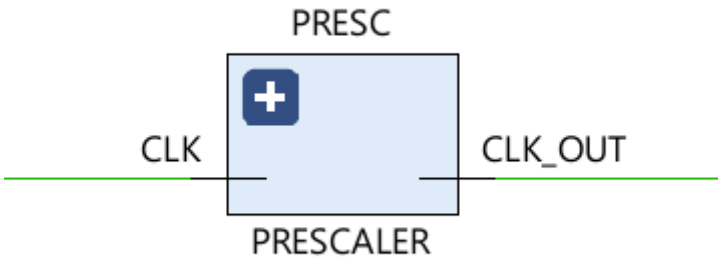


Al ejecutar el testbench, se verificó el correcto funcionamiento de la entidad. Cuando la entrada **CE** está activada, el **RESET** en 1 y ocurre el primer flanco positivo del reloj, la máquina comienza a aceptar monedas. En el primer caso, seleccionamos el refresco con precio de 1€. Al introducir una moneda de 20 céntimos como último valor, la salida **PAGO_OK** se activa, ya que la variable **CUENTA** alcanza el precio especificado.

Posteriormente, se realiza un reinicio y se selecciona el segundo refresco (1,30€). El mismo procedimiento se repite, y la opción de refresco queda registrada en la variable

REFRESCO_ACTUAL, lo que impide cambiar la selección mientras se introducen monedas. Aunque la entrada **TIPO_REFRESCO** cambie durante el proceso, la salida PAGO_OK se activa únicamente al alcanzar el precio correspondiente al segundo refresco.

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.

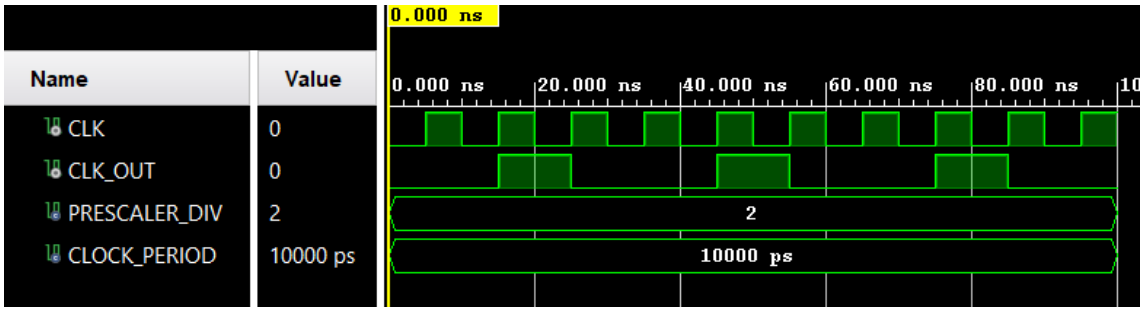
El **PRESCALER** funciona como un contador que incrementa su valor con cada pulso del reloj de entrada. Al alcanzar un valor predeterminado, el contador se reinicia y genera un pulso en la señal de salida del reloj. Por ejemplo, si se configura para un valor de 100, el **PRESCALER** producirá un pulso 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
        if CLK_BUFFER(CLK_BUFFER'length - 1) = '1' then
            CLK_BUFFER <= (others => '0');
        else
            CLK_BUFFER <= CLK_BUFFER + 1;
        end if;
    end if;
end process;
```

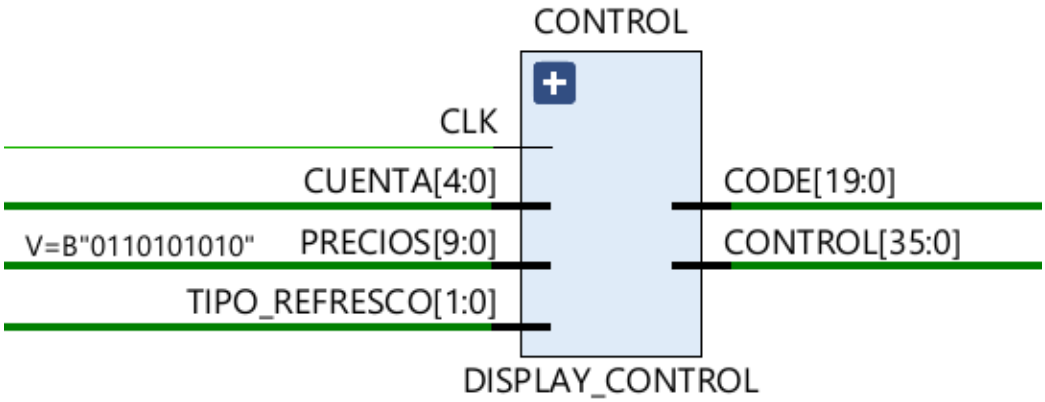
Se ejecuta en cada borde ascendente del reloj de entrada CLK. Si el bit más significativo de **CLK_BUFFER** (CLK_BUFFER (CLK_BUFFER'length - 1)) es '1', esto significa que el contador ha alcanzado su valor máximo y debe reiniciarse, lo cual se hace asignando cero a todo el vector **CLK_BUFFER**. Si no es '1', el contador simplemente se incrementa en uno.

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 uno de los componentes principales del proyecto, ya que administra toda la lógica necesaria para controlar los displays. Para ello, necesita como entradas información esencial como el precio de los productos, el refresco seleccionado y la cantidad de dinero ingresada. La señal de reloj proviene de la entidad **PRESCALER**, que determina la frecuencia de actualización de los displays. Cuenta únicamente con dos salidas: una para el control de los dígitos del display y otra para el código correspondiente, que puede representar números o letras. Estas salidas pasan a través de la máquina de estados antes de llegar al decoder. Esto es necesario porque la entidad de control administra los displays para todos los estados, enviando únicamente las señales de control específicas según el estado activo. Es importante señalar que el punto decimal se incluye dentro del vector de control de segmentos, aumentando su longitud a nueve elementos, siendo el menos significativo el punto decimal.

A nivel de implementación, se han creado arrays de vectores tanto para el control como para los códigos, de manera que cada array tiene 4 vectores que corresponden a los estados de la máquina. Esto se ilustra en la siguiente imagen:

```
--Arrays que contienen todos los controles y codes de los estados
type CONTROL_TOTAL is array (0 to NUM_ESTADOS - 1) of std_logic_vector(NUM_DISPLAYS - 1 downto 0);
type CODE_TOTAL is array(0 to NUM_ESTADOS - 1) of std_logic_vector(TAM_CODE - 1 downto 0);

--Señales de registro
signal CONTROL_SIG : CONTROL_TOTAL;
signal CODE_SIG : CODE_TOTAL;
```

A continuación, se detalla el proceso encargado de calcular la lógica de control. Este proceso se ejecuta en cada flanco positivo del reloj de salida del prescaler. Dentro del proceso, se actualiza un dígito para cada estado siguiendo la lógica definida.

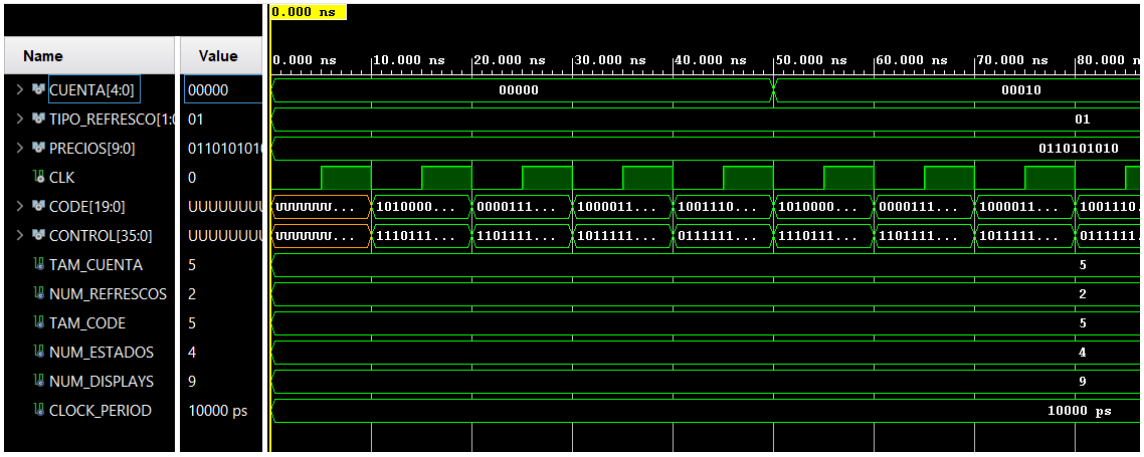
Cada elemento de la señal de control opera de forma independiente y gestiona un estado específico. En cada sentencia case, se actualiza el siguiente dígito junto con su código correspondiente. Por ejemplo, en el estado de pago, uno de los dígitos que requiere cálculo es el primer decimal de la cantidad 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:

```
case CONTROL_SIG(1) is --Muestra el precio del refresco y lo que falta por pagar
    when "111111101" =>
        CONTROL_SIG(1) <= "111111011";
        --Muestra el primer decimal del precio que falta por pagar
        --Por ejemplo si faltan 70cents por pagar muestra un 7
        if TIPO_REFRESCO = "01" then
            if PRECIOS(TAM_CUENTA - 1 downto 0) - CUENTA >= "01010" then
                CODE_SIG(1) <= PRECIOS(TAM_CUENTA - 1 downto 0) - "01010" - CUENTA;
            else
                CODE_SIG(1) <= PRECIOS(TAM_CUENTA - 1 downto 0) - CUENTA;
            end if;
         elsif TIPO_REFRESCO = "10" then
            if PRECIOS((TAM_CUENTA*2) - 1 downto TAM_CUENTA) - CUENTA >= "01010" then
                CODE_SIG(1) <= PRECIOS((TAM_CUENTA*2) - 1 downto TAM_CUENTA) - "01010" - CUENTA;
            else
                CODE_SIG(1) <= PRECIOS((TAM_CUENTA*2) - 1 downto TAM_CUENTA) - CUENTA;
            end if;
        end if;
        --DP <= '1';
```

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 son el resultado de concatenar los elementos de los arrays de control y código.

```
CONTROL <= CONTROL_SIG(3)&CONTROL_SIG(2)&CONTROL_SIG(1)&CONTROL_SIG(0);
CODE <= CODE_SIG(3)&CODE_SIG(2)&CODE_SIG(1)&CODE_SIG(0);
```

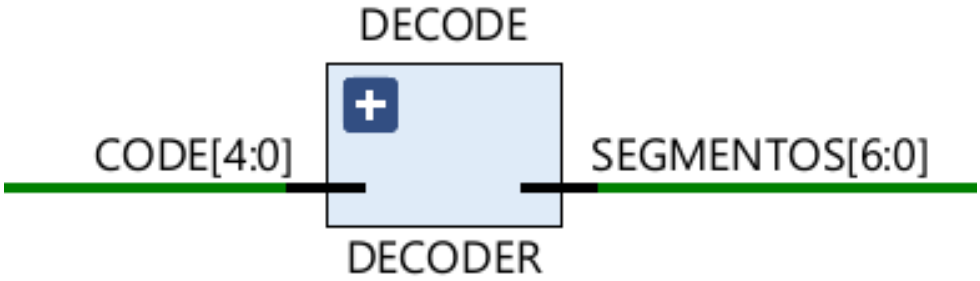
TESTBENCH:



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. En la siguiente imagen, se describe únicamente el control del primer estado por este motivo:

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.

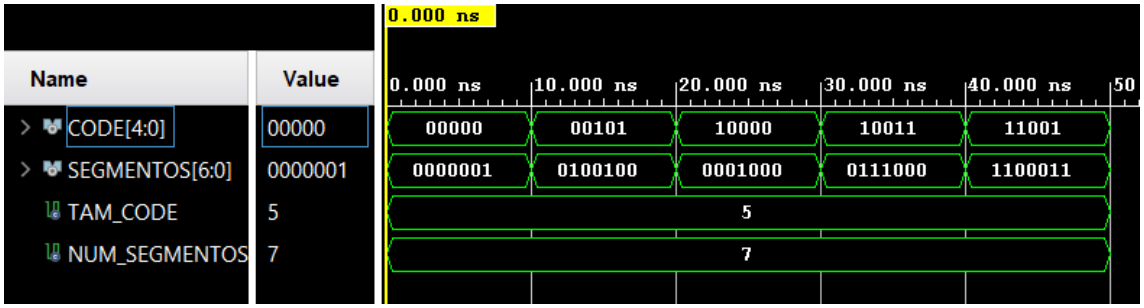
La entidad **DECODER** utiliza dos parámetros genéricos: **TAM_CODE**, que representa la longitud del vector de entrada (cantidad de bits en la entrada), y **NUM_SEGMENTOS**, que indica el tamaño del vector de salida (cantidad de segmentos a controlar). **CODE** es la entrada del decodificador, mientras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

```
with CODE select
    SEGMENTOS <= "0000001" when "00000",
    "1001111" when "00001",
    "0010010" when "00010",
    "0000110" when "00011",
    "1001100" when "00100",
    "0100100" when "00101",
    "0100000" when "00110",
    "0001111" when "00111",
    "0000000" when "01000",
    "0000100" when "01001",
    "0001000" when "10000", -- LETRA A
    "1000010" when "10001", -- LETRA D
    "0110000" when "10010", -- LETRA E
    "0111000" when "10011", -- LETRA F
    "1110001" when "10100", -- LETRA L
    "1100010" when "10101", -- LETRA O
    "0011000" when "10110", -- LETRA P
    "1111010" when "10111", -- LETRA R
    "1110000" when "11000", -- LETRA T
    "1100011" when "11001", -- LETRA U
    "1111110" when others;
```

En esta sección del código se realiza el proceso de decodificación. Cada línea dentro de la estructura with ... select asigna un patrón de bits a **SEGMENTOS** basado en el valor de **CODE**. Por ejemplo, cuando **CODE** toma el valor "00000", **SEGMENTOS** se establece como "0000001", lo que podría representar un número o una letra en un display de siete segmentos. La última línea, when others, actúa como una condición por defecto que se ejecuta si **CODE** no coincide con ninguno de los casos definidos anteriormente.

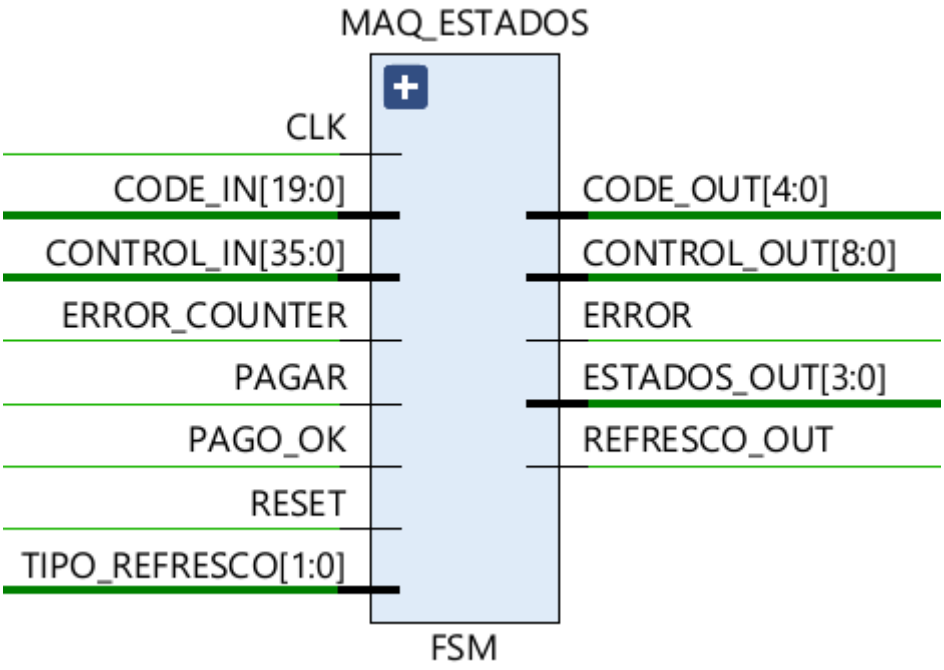
El decodificador está diseñado para manejar tanto números como algunas letras, lo que permite mostrar distintos caracteres en el display de siete segmentos.

TESTBENCH:



Finalmente, se verifica que cada combinación de entrada en **CODE** genera la salida esperada en **SEGMENTOS**. 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: **ERROR_COUNTER** (error en el pago del refresco), **PAGO_OK** y **TIPO_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 **STATES**, comenzando siempre desde el estado de reposo (S0).

```
type STATES is (S0,S1,S2,S3);
signal CURRENT_STATE: STATES := S0;
signal NEXT_STATE: STATES;
```

A partir de aquí, podemos dividir la entidad en tres procesos principales: state_register, nextstate y output_control.

```
STATE_REGISTER: process(RESET, CLK)
begin
    if RESET = '0' then
        CURRENT_STATE <= S0;
    elsif rising_edge(CLK) then
        CURRENT_STATE <= NEXT_STATE;
    end if;
end process;
```

El primer proceso, state_register, tiene un funcionamiento simple: actualiza el estado actual de la máquina en cada flanco de reloj y la devuelve al estado de reposo cuando se activa la entrada **RESET**.

```
NEXTSTATE: process(CLK) --, PAGAR, PAGO_OK, ERROR_COUNTER, CURRENT_STATE)
begin
    NEXT_STATE <= CURRENT_STATE;
    case CURRENT_STATE is
        when S0 =>
            if PAGAR = '1' and TIPO_REFRESCO /= "00" then
                NEXT_STATE <= S1;
            end if;
        when S1 =>
            if PAGO_OK = '1' then
                NEXT_STATE <= S2;
            elsif ERROR_COUNTER = '1' then
                NEXT_STATE <= S3;
            end if;
        when S2 =>
            if PAGAR = '0' then
                NEXT_STATE <= S0;
            end if;
        when S3 =>
            if RESET = '0' then
                NEXT_STATE <= S0;
            end if;
    end case;
end process;
```

El proceso nextstate gestiona la transición entre estados de acuerdo con el diagrama de estados. Por ejemplo, si estamos en el estado de reposo (S0) y se activa la señal PAGAR, se pasará inmediatamente al siguiente estado, de modo que la variable NEXT_STATE tomará el valor S1. Posteriormente, este cambio se actualiza en la línea 47, estableciendo **CURRENT_STATE** como S1, siguiendo así las transiciones definidas en el diagrama.

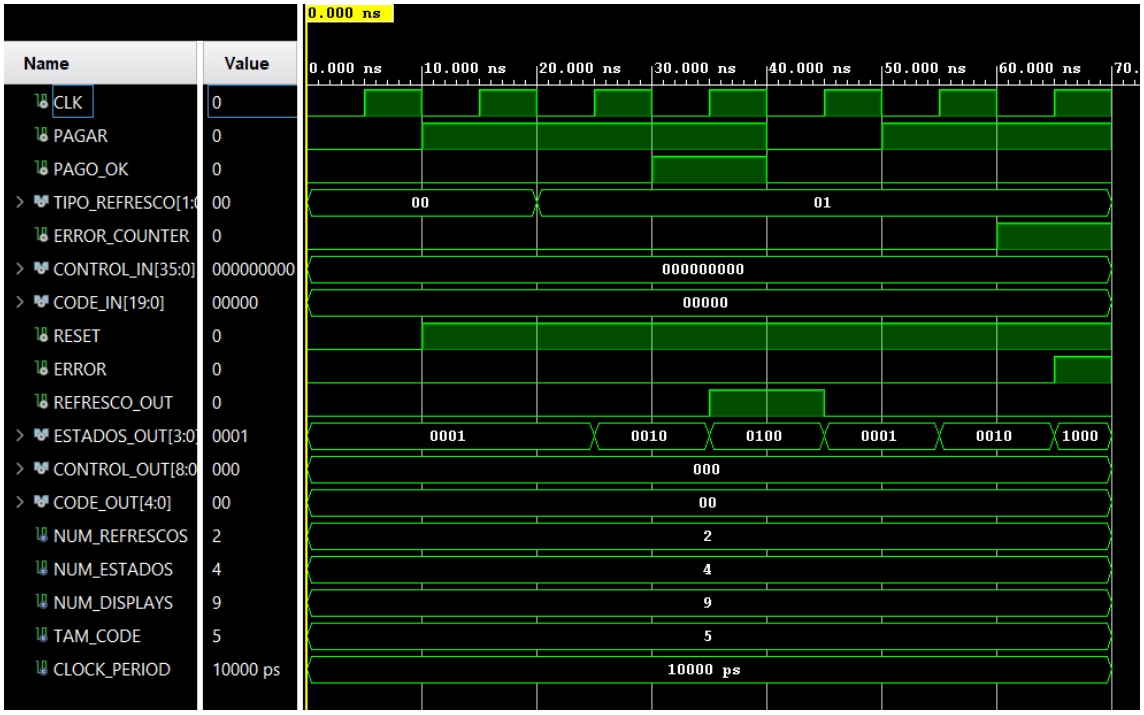
```
OUTPUT_CONTROL: process(CLK, CURRENT_STATE)
begin
    case CURRENT_STATE is
        when S0 =>
            ERROR <= '0';
            REFRESCO_OUT <= '0';
            ESTADOS_OUT <= "0001";
            CONTROL_OUT <= CONTROL_IN(NUM_DISPLAYS - 1 DOWNTO 0);
            CODE_OUT <= CODE_IN(TAM_CODE - 1 DOWNTO 0);

        when S1 =>
            ERROR <= '0';
            REFRESCO_OUT <= '0';
            ESTADOS_OUT  <= "0010";
            CONTROL_OUT <= CONTROL_IN((NUM_DISPLAYS * 2) - 1 DOWNTO NUM_DISPLAYS);
            CODE_OUT <= CODE_IN((TAM_CODE * 2) - 1 DOWNTO TAM_CODE);
```

Por último, el proceso output_control se encarga de gestionar las salidas en función del estado actual. Por ejemplo, en los estados de reposo o de pago, las variables **ERROR** y **REFRESCO_OUT** permanecerán en cero, ya que en estos estados aún no se ha completado el importe necesario para obtener el refresco. En el estado dos, se activa la señal **REFRESCO_OUT**, y en el estado tres, se activa la señal ERROR. Además, este proceso también regula las etapas de los displays, ajustando su configuración según el estado en el que se encuentre la máquina.

Adicionalmente, se utiliza una variable llamada **ESTADOS_OUT**, que no se había mencionado antes. Esta variable se asigna a cuatro LEDs de la placa para indicar en tiempo real la fase en la que se encuentra el sistema durante la simulación.

TESTBENCH:



Al simular esta entidad, podemos observar cómo avanzamos a través del diagrama de estados gracias a la variable auxiliar **ESTADOS_OUT**. Iniciamos en el estado de reposo y avanzamos al estado 1 al seleccionar **PAGAR** y un tipo de refresco. Al activarse **PAGO_CORRECTO**, pasamos al estado 2, donde la máquina entrega el refresco (se activa **REFRESCO_OUT**), y al desactivar la señal **PAGAR** (poniéndola en 0), volvemos al estado de reposo. Si repetimos el proceso, pero en esta ocasión ocurre un error en el contador, se pasa al estado 3, donde se activa la señal **ERROR** de la máquina.

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/Franxcus/M-quina_Expendedora.git