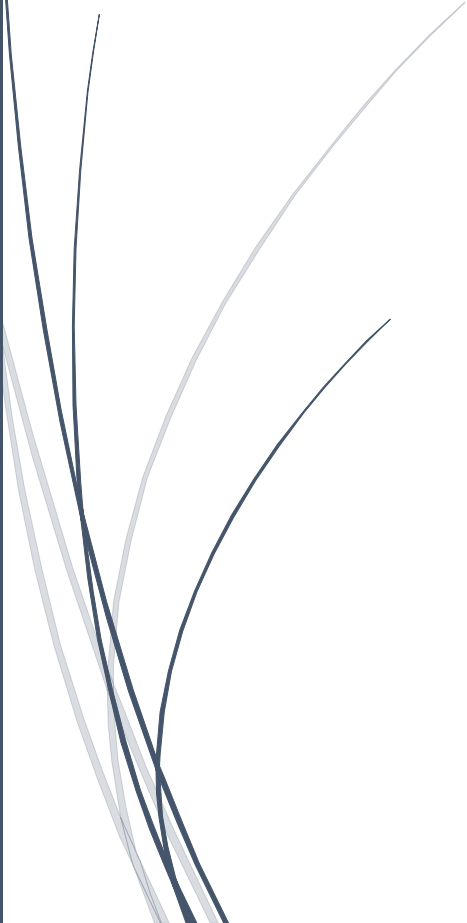


A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

19-1-2022

DISEÑO DE UNA MÁQUINA EXPENDEDORA DE REFRESCOS EN UNA FPGA

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Javier Martín Sanz 54728
Luis Pérez González 54793
Luis Torres del Nuevo 54887

Índice

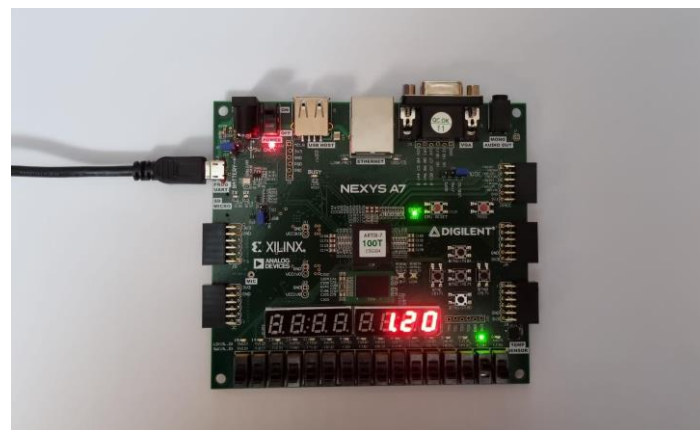
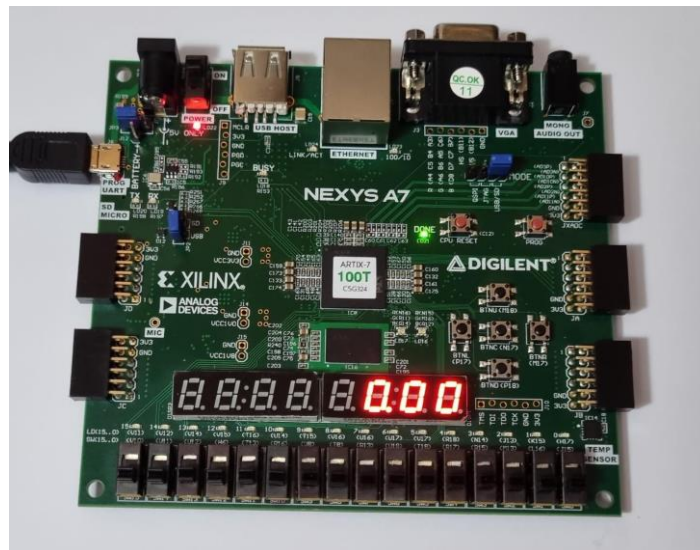
1. Introducción	2
1.1. Explicación y funcionamiento de la máquina expendedora.	2
1.2. Fotos del proceso de la máquina expendedora.	2
2. Código en VHDL (entidades).....	3
2.1. Top.....	3
2.1.1. Descripción de la fuente	3
2.2.2. Estructura interna.....	4
2.2. Synchrnzr	5
2.3. Debouncer	5
2.4. Edgedtctr	6
2.5. Counter.....	6
2.5.1. Descripción de la fuente	6
2.5.2. Estructura interna.....	7
2.6. FSM.....	7
2.6.1. Descripción de la fuente	7
2.6.2. Estructura interna.....	9
2.7. FSM_SW	9
2.7.1. Descripción de la fuente	9
2.7.2. Estructura interna.....	9
2.8. Display_control.....	9
2.8.1. Descripción de la fuente	9
2.8.2. Estructura interna.....	10
3. Testbench	11
3.1. Display	11
3.2. Counter	12

1. Introducción

1.1. Explicación y funcionamiento de la máquina expendedora.

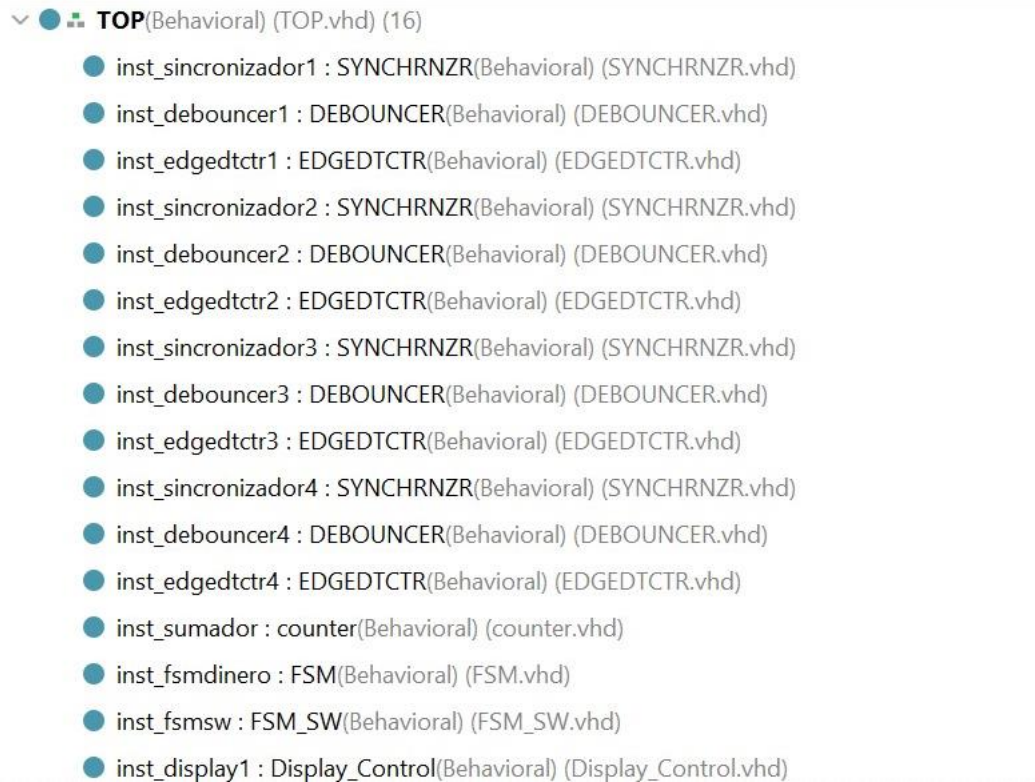
Nuestro proyecto consiste en la elección del tipo de bebida a través de uno de los interruptores (SW), una vez seleccionado la bebida para comprobar que es la correcta se enciende un led verde. En el display comprobamos que empieza desde 0.0 euros y pulsando los botones indicados con el precio introduciremos el dinero hasta los 1.20 euros, cuando superemos el precio, se pondrán rallas indicando que hemos superado el precio.

1.2. Fotos del proceso de la máquina expendedora.



2. Código en VHDL (entidades)

En nuestro proyecto podemos diferenciar una serie de componentes que en su conjunto definen nuestra entidad general que es **Top**.



```
▼ ● ■ TOP(Behavioral) (TOP.vhd) (16)
  ● inst_sincronizador1 : SYNCHRNZR(Behavioral) (SYNCHRNZR.vhd)
  ● inst_debouncer1 : DEBOUNCER(Behavioral) (DEBOUNCER.vhd)
  ● inst_edgedtctr1 : EDGEDTCTR(Behavioral) (EDGEDTCTR.vhd)
  ● inst_sincronizador2 : SYNCHRNZR(Behavioral) (SYNCHRNZR.vhd)
  ● inst_debouncer2 : DEBOUNCER(Behavioral) (DEBOUNCER.vhd)
  ● inst_edgedtctr2 : EDGEDTCTR(Behavioral) (EDGEDTCTR.vhd)
  ● inst_sincronizador3 : SYNCHRNZR(Behavioral) (SYNCHRNZR.vhd)
  ● inst_debouncer3 : DEBOUNCER(Behavioral) (DEBOUNCER.vhd)
  ● inst_edgedtctr3 : EDGEDTCTR(Behavioral) (EDGEDTCTR.vhd)
  ● inst_sincronizador4 : SYNCHRNZR(Behavioral) (SYNCHRNZR.vhd)
  ● inst_debouncer4 : DEBOUNCER(Behavioral) (DEBOUNCER.vhd)
  ● inst_edgedtctr4 : EDGEDTCTR(Behavioral) (EDGEDTCTR.vhd)
  ● inst_sumador : counter(Behavioral) (counter.vhd)
  ● inst_fsmdinero : FSM(Behavioral) (FSM.vhd)
  ● inst_fsmsw : FSM_SW(Behavioral) (FSM_SW.vhd)
  ● inst_display1 : Display_Control(Behavioral) (Display_Control.vhd)
```

2.1. Top

2.1.1. Descripción de la fuente

Como podemos ver en la imagen anterior, vemos que Top es nuestra fuente principal, encargada de unir los componentes de tal manera que consiga un correcto funcionamiento de la máquina expendedora. En esta fuente podemos diferenciar una serie de entradas y salidas:

```

clk: in std_logic;
PUSH_10CENT: in std_logic;
PUSH_20CENT: in std_logic;
PUSH_50CENT: in std_logic;
PUSH_1EURO: in std_logic;
reset: in std_logic;
segmentos: out std_logic_vector(7 downto 0);
digsel: out std_logic_vector(7 downto 0);
sw1: in std_logic;
sw2: in std_logic;
sw3: in std_logic;
sw4: in std_logic;
led: out std_logic_vector(3 downto 0)

```

CLK: Entrada la de señal de reloj de 100 MHz, es usada por todos los componentes del sistema como podremos mas adelante en su estructura interna.

PUSH_10CENT, PUSH_20CENT, PUSH_50CENT, PUSH_1EURO: se trata de 4 entradas y cada una representa un botón de la placa que hemos implementado en el archivo. Xdc

digsel: Salida de un vector de 8 elementos, se usa para elegir el display que queremos encender perteneciendo a cada uno de los elementos del vector uno de los displays.

Segmentos: salida de un vector de 8 elementos, gracias a ella podemos encender cada uno de los segmentos de un display incluyendo el punto.

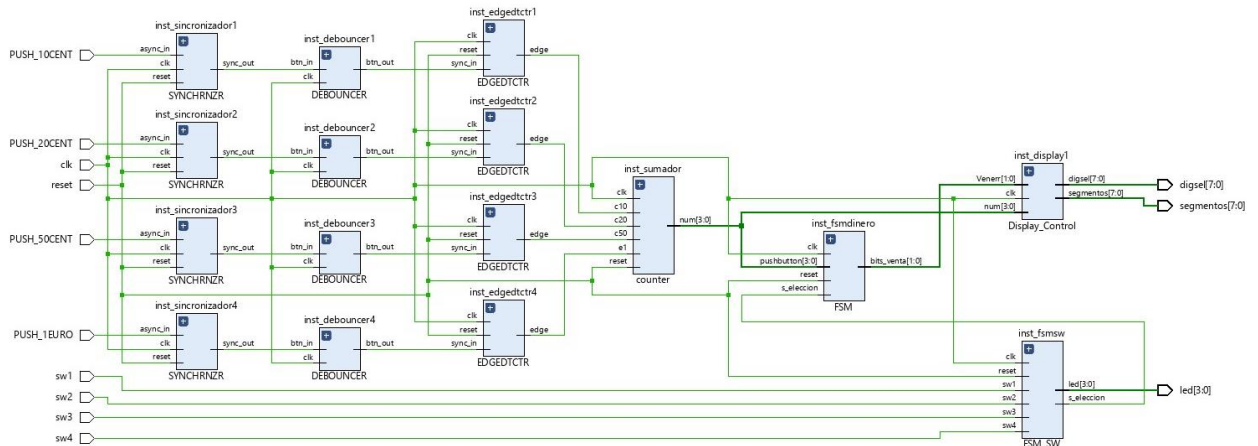
Sw1, Sw2, Sw3, Sw4: es una entrada de tal manera que gracias a ella podemos identificar el tipo de bebida que elegimos moviendo los 4 interruptores asignados.

Led: vector de salida de 4 elementos de tal manera que irán asignados a los sw para identificar que hemos hecho una elección correcta de la bebida.

2.2.2. Estructura interna

Como podemos ver en la siguiente imagen el diseño elaborado queda definido gracias a la unión de las diferentes fuentes que se consigue a través de señales.

El diagrama RTL de la entidad TOP queda definido de la siguiente manera:

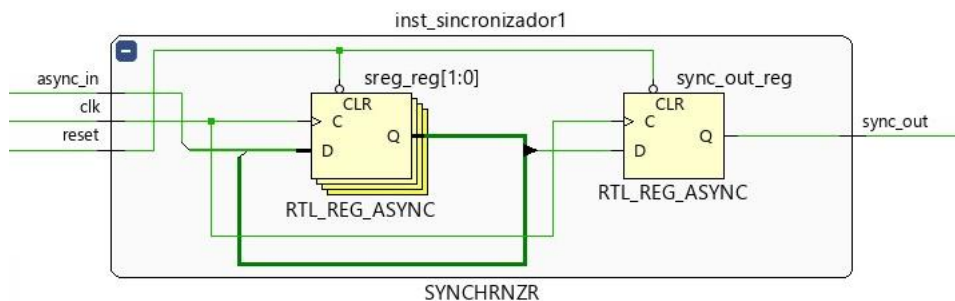


2.2. Synchrnzr

Se emplearán 4 sincronizadores, uno por cada señal de entrada producida por los botones que simbolizan las diferentes monedas que se introducen en la máquina.

El sincronizador se encarga, como su propio nombre indica, en sincronizar las señales de entrada con la señal de reloj de la placa, evitando así la metaestabilidad ocurrida cuando se produce un cambio en la señal de entrada durante un flanco del reloj, pues la pulsación del botón se puede producir de forma asíncrona al reloj.

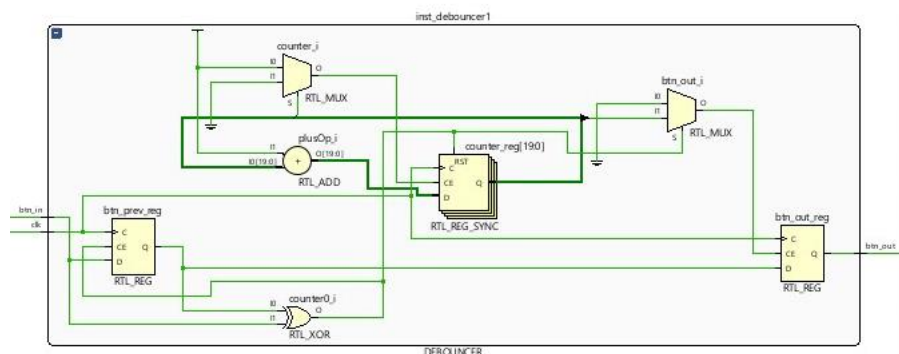
Tendrá además una entrada de reset y una para el clk.



2.3. Debouncer

Se requiere para evitar el efecto de los rebotes que se generan al producir las señales con los botones. Para ello se utilizan flip flops que son capaces de almacenar el estado anterior, si durante un periodo de tiempo amplio el valor actual concuerda con ese valor almacenado se puede asumir que el estado es estable. Tendremos en nuestra fuente TOP cuatro al igual que sincronizadores, una por cada botón.

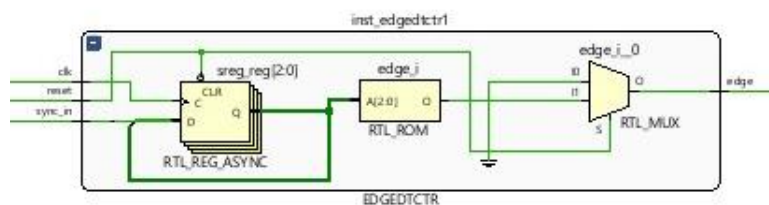
Tendrá además una entrada de reset y una para el clk.



2.4. Edgedtctr

El tiempo que se pulsa un botón no tiene una duración determinada, por lo que se pueden encontrar inconvenientes a la hora de registrar la señal introducida. Este módulo se emplea para que cada vez que se produzca un flanco de bajada en el pulsador, la señal de salida del módulo se active, contando así solamente los flancos de subida del pulsador, evitando así que se produzcan cambios continuamente mientras el botón se encuentra pulsado.

Tendrá además una entrada de reset y una para el clk.



2.5. Counter

2.5.1. Descripción de la fuente

Con la fuente contador podemos ir sumando las cantidades que hayamos introducido con los diversos botones. Tiene una serie de entradas y salidas:

s_eleccion: entrada para unir el estado S0 con la salida de la FMS_SW.

Clk: entrada de la señal de reloj.

Reset: entrada usando el propio de la placa.

c10, c20, c50, e1: 4 salidas individuales y cada una representa un botón de la placa que hemos implementado en el archivo. Xdc

Num: vector de salida de 4 bits con el que gracias a él mostramos la suma de la cuenta en los displays.

2.5.2. Estructura interna

2.6. FSM

2.6.1. Descripción de la fuente

Esta fuente consiste en una máquina de estados encargada de coordinar las distintas fases a la hora de introducir las diferentes monedas. La entidad tendrá como entradas y salidas:

s_eleccion: se trata de una entrada para unir el estado S0 con la salida de la FMS_SW.

Clk: entrada de la señal de reloj.

Reset: entrada usando el propio de la placa.

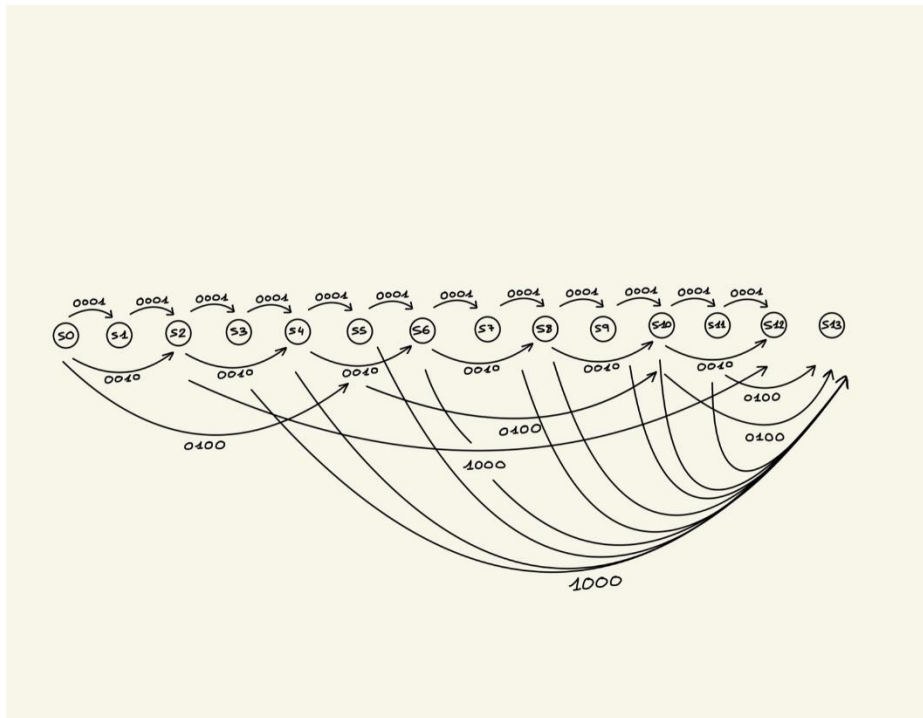
Bits_venta: vector de salida de dos bits generado para saber si se consigue llegar al precio del refresco o se supera.

pushbutton: vector de entrada de 4 bits para diferenciar la cantidad de dinero que introduce cada botón de la placa.

Cada uno de los estados, que se declaran a través de una enumeración (type):

```
TYPE EstadosMonedas IS (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13);  
SIGNAL present_state, next_state: EstadosMonedas;  
n
```

Los estados de la FSM están representados en el siguiente diagrama:



S0: 0.00 euros

S1: 0.10 euros

S2: 0.20 euros

S3: 0.30 euros

S4: 0.40 euros

S5: 0.50 euros

S6: 0.60 euros

S7: 0.70 euros

S8: 0.80 euros

S9: 0.90 euros

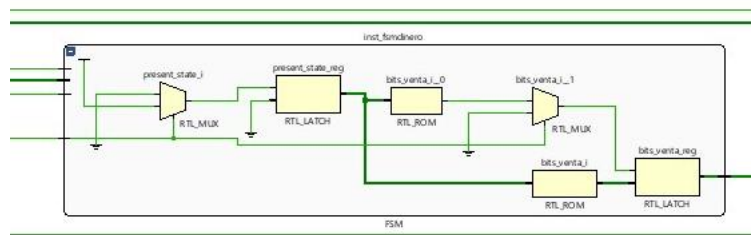
S10: 1.00 euros

S11: 1.10 euros

S12: 1.20 euros

S13: error

2.6.2. Estructura interna



2.7. FSM_SW

2.7.1. Descripción de la fuente

Esta fuente es creada para conseguir unir los sw con el estado inicial de tal manera que al mover el interruptor y se encienda el led podamos pasar al estado de 0.0 euros.

Esta fuente consta de una serie de entradas y salidas:

Clk: entrada de la señal de reloj.

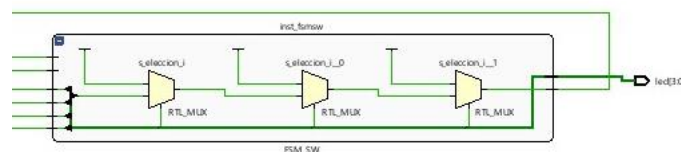
Reset: entrada usando el propio de la placa.

Sw1, Sw2, Sw3, Sw4: es una entrada de tal manera que gracias a ella podemos identificar el tipo de bebida que elegimos moviendo los 4 interruptores asignados.

Led: vector de salida de 4 elementos de tal manera que irán asignados a los sw para identificar que hemos hecho una elección correcta de la bebida.

S_eleccion: salida que nos permite pasar al estado inicial.

2.7.2. Estructura interna



2.8. Display_control

2.8.1. Descripción de la fuente

La entidad Display_Control es la encargada de mostrar en los displays la cuenta del dinero introducido en la máquina. Cuenta con una serie de entradas y salidas:

Num: vector de entrada de 4 bits con el que gracias a el mostramos la suma de la cuenta en los displays.

Clk: entrada de la señal de reloj.

Venerr: vector de entrada de dos bits generado para saber si se consigue llegar al precio del refresco o se supera.

Digsel: vector de salida de 8 bits que indica los displays que queremos encender de la placa.

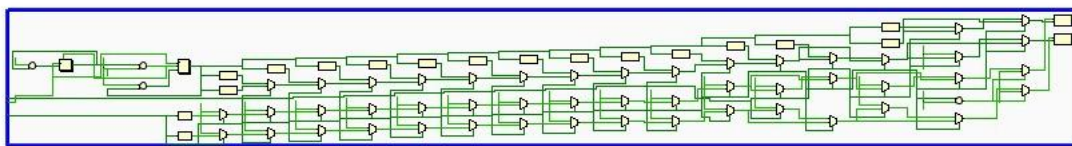
Segmentos: vector de salida de 8bits que indica qué segmentos del display se van a encender

Debido a que los cátodos de los segmentos son comunes a los ocho displays, se mostraría el mismo dígito en cada display. Para que en los displays no muestren lo mismo, los displays se manejarán en periodos de 1.6ms, es decir, como tenemos 8 displays, cada display se ilumina durante 1/8 periodo de forma consecutiva.

En la siguiente figura se muestra el primer process:

```
process (clk)
begin
    --Periodo 1.6 ms-> clk_counter=160000
    if rising_edge(clk) then
        clk_counter<=clk_counter + 1;
        --periodo/8 = 0.2 ms -> clk_counter=20000
        if clk_counter>=20000 then
            clk_counter<=0;
            digitos<=digitos +1;
        end if;
        if digitos > 7 then
            digitos<=0;
        end if;
    end if;
end process;
```

2.8.2. Estructura interna



3. Testbench

3.1. Display

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_Display_Control is
end tb_Display_Control;

architecture tb of tb_Display_Control is

    component Display_Control
        port (num      : in std_logic_vector (3 downto 0);
              clk      : in std_logic;
              Venerr    : in std_logic_vector (1 downto 0);
              digsel    : out std_logic_vector (7 downto 0);
              segmentos : out std_logic_vector (7 downto 0));
    end component;

    signal num      : std_logic_vector (3 downto 0);
    signal clk      : std_logic;
    signal Venerr    : std_logic_vector (1 downto 0);
    signal digsel    : std_logic_vector (7 downto 0);
    signal segmentos : std_logic_vector (7 downto 0);

begin

    dut : Display_Control
    port map (num      => num,
              clk      => clk,
              Venerr    => Venerr,
              digsel    => digsel,
              segmentos => segmentos);

    -- Clock generation
    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';

    -- EDIT: Check that clk is really your main clock signal
    clk <= TbClock;

    stimuli : process
    begin
        -- EDIT Adapt initialization as needed
        num <= (others => '0');
        Venerr <= (others => '0');

        YOURRESETSIGNAL <= '0';
        wait for 100 ns;

        -- EDIT Add stimuli here
        wait for 100 * TbPeriod;

        -- Stop the clock and hence terminate the simulation
        TbSimEnded <= '1';
        wait;
    end process;

end tb;

-- Configuration block below is required by some simulators. Usually no need to edit.

configuration cfg_tb_Display_Control of tb_Display_Control is
    end if;

end process;

end Behavioral;
```

3.2. Counter

```
entity tb_counter is
end tb_counter;

architecture tb of tb_counter is

    component counter
        port (clk      : in std_logic;
              c10     : in std_logic;
              c20     : in std_logic;
              c50     : in std_logic;
              e1      : in std_logic;
              reset    : in std_logic;
              num      : out std_logic_vector (3 downto 0));
    end component;

    signal clk      : std_logic;
    signal c10     : std_logic;
    signal c20     : std_logic;
    signal c50     : std_logic;
    signal e1      : std_logic;
    signal reset    : std_logic;
    signal num      : std_logic_vector (3 downto 0);

    constant TbPeriod : time := 1000 ns; -- EDIT Put right period here
    signal TbClock : std_logic := '0';
    signal TbSimEnded : std_logic := '0';

begin

    dut : counter
    port map (clk => clk,
              c10 => c10,
              c20 => c20,
              c50 => c50,
              e1  => e1,
              reset => reset,
              num => num);

    -- Clock generation
    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';

    -- EDIT: Check that clk is really your main clock signal
    clk <= TbClock;
```

```

stimuli : process
begin
    -- EDIT Adapt initialization as needed
    c10 <= '0';
    c20 <= '0';
    c50 <= '0';
    e1 <= '0';

    -- Reset generation
    -- EDIT: Check that reset is really your reset signal
    reset <= '1';
    wait for 100 ns;
    reset <= '0';
    wait for 100 ns;

    -- EDIT Add stimuli here
    wait for 100 * TbPeriod;

    -- Stop the clock and hence terminate the simulation
    TbSimEnded <= '1';
    wait;
end process;

end tb;

-- Configuration block below is required by some simulators. Usually no need to edit.

configuration cfg_tb_counter of tb_counter is
    for tb
        end for;
end cfg tb counter;

```