CINTA DE CORRER

TRABAJO EN GRUPO VHDL

SISTEMAS ELECTRÓNICOS DIGITALES

Grupo 24:

Laura Álvarez Pinedo, 53811

Ignacio Dopazo López, 55218

Adrián Gómez García, 55269

Curso 2022-2023

Tutor: Rubén Núñez

Índice

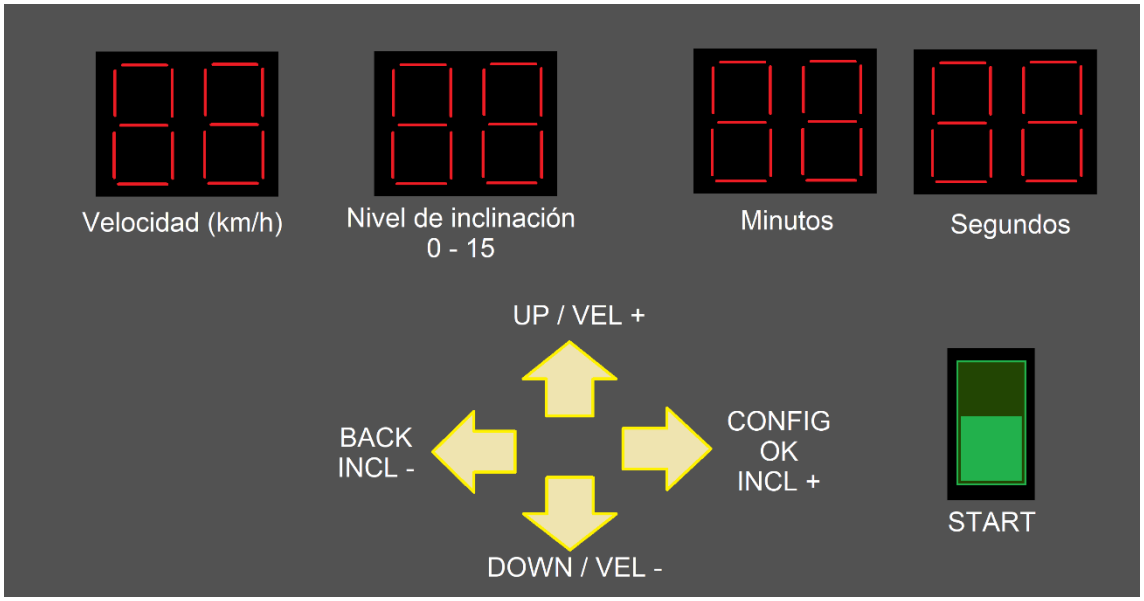
Introducción

El proyecto elegido para implementar con FPGA programando en VHDL es una cinta de correr.

Tendrá las siguientes funcionalidades:

- Configuración de una cuenta atrás de tiempo (modo cuenta hacia abajo y el tiempo)
- Cuenta hacia arriba de tiempo
- Configuración de velocidad
- Configuración de inclinación

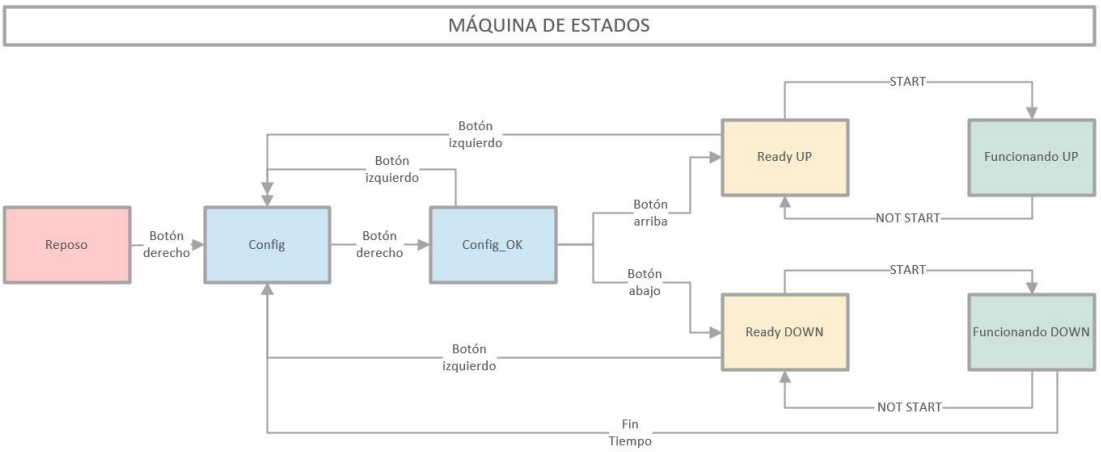
Todo ello se haría con la siguiente interfaz de usuario, que se representa simbólicamente en la siguiente imagen:



Resumen de algunos puntos clave

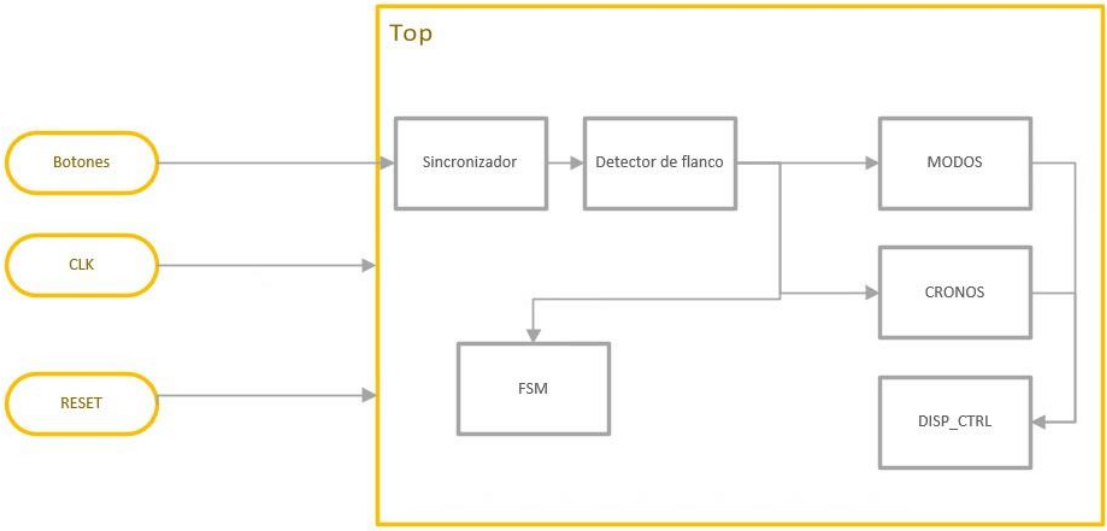
Máquina de estados

Es la entidad que coordina el funcionamiento del sistema, activando y desactivando otras entidades y haciendo que en cada momento las entradas tengan distintas funcionalidades, puesto que el número de botones utilizados como entradas es limitado. El diagrama de estados es el siguiente:



Entidad top

Es la entidad que conecta una serie de entidades en las que se divide el proyecto. Se muestra el siguiente diagrama a modo de esquema resumido:



División del proyecto
El proyecto cuenta con cuatro grandes áreas diferenciadas:

- Máquina de estados (FSM)
- Control del tiempo (CRONO)
- Control del modo, es decir, velocidad e inclinación (MODOS)
- Control de los displays (DISP_CTRL)

En un primer momento se establecen las entradas y salidas que la entidad top debe tener y posteriormente se hace lo propio con las entidades anteriores. Estas entidades realizan tareas que todavía son relativamente complejas por lo que se subdividen a su vez en entidades más pequeñas. Algunas de esas entidades son comunes a varias de las cuatro entidades arriba mencionadas (por ejemplo, la entidad Incremento), pero en líneas generales se pueden desarrollar y probar de forma independiente.

Información complementaria

El desarrollo del proyecto se ha seguido en GitHub, por lo que toda la información de la memoria se encuentra de forma más extendida en la Wiki del proyecto.

Además, se ha grabado un video demostrativo del funcionamiento del proyecto en la placa Nexys4 DDR:

https://upm365.sharepoint.com/:v:/s/SED_TRABAJO/Eb7U0kOS7YtLg7PLMiYsLpkBrqlYxP3vceiudpXGVf1QAQ?e=Zl2Dbd

Entidad Top

Para seguir una estructura de diseño Top-Bottom, se crea una entidad global de arquitectura estructural cuyo funcionamiento se base en los diferentes componentes que la forman.

Entidad

Generic

```
    button_width     : positive := 4;
-- Numero de botones
    speed_width      : positive := 4;
-- Bits para la comunicacion con el motor (número que va a mostrar el bcd)
    incl_width       : positive := 4;
-- Bits para la comunicacion con el motor (número que va a mostrar el bcd)
    digits           : positive := 8;
-- Numero de displays de 7 segmentos de la placa
    digits_range     : positive := 3;
    in_time_width    : positive := 13;
-- Ancho para segundos de carga
    out_time_width   : positive := 13;
    divide           : positive:= 100000000;
-- Ancho para segundos
    conversion       : positive:=30;
    max_time         : positive:=5940;
-- El valor corresponde a 99 Minutos, valor maximo mostrable
    conversion_speed : positive := 1;
    conversion_incl  : positive := 1;
    max : positive   := 15
```

Entradas

```
   BUTTONS     : in std_logic_vector(button_width-1 downto 0);
--Botones de la placa
   RESET       : in std_logic;
-- RESET de todo el sistema, asincrono
   CLK         : in std_logic;
-- Reloj
   START       : in std_logic;
-- Para que la cinta comience a funcionar
```

Salidas

```
    SPEED       : out std_logic_vector (speed_width - 1 downto 0);
-- Al control del motor de velocidad
    INCL        : out std_logic_vector (incl_width - 1 downto 0);
-- Al control del motor de inclinacion
    BCD_DATA    : out std_logic_vector (7 downto 0);
-- Salida datos a los displays
    BDC_SEL     : out std_logic_vector (digits - 1 downto 0)
-- Seleccion del digito
    leds        : out std_logic_vector(6 downto 0)
```

Arquitectura

En esta entidad se llaman y asocian todas las entidades más sencillas, por lo que en la arquitectura de tipo estructural se crean componentes para las siguientes entidades:

- SYNCHRNZR
- EDGEDTCTR
- FSM
- CRONO
- MUX2A1
- MODOS
- DISP_CTRL

Entidad SYNCHRNZR (Sincronizador)

Debido al uso de pulsadores se puede producir un rebote, es decir, que al pulsarlos esten un tiempo corto cambiando de estados hasta estabilizarse. Además, al ser entradas asíncronas, hay que sincronizarlas. Por estos motivos, para evitar la metaestabilidad se han añadido las entidades de SYNCHRNZR y EDGEDTCTR.

Entidad

Entradas

```
  CLK : in STD_LOGIC;
  ASYNC_IN : in STD_LOGIC; -- Entrada asincrona
  Reset : in STD_LOGIC; -- Reset asincrono
```

Salidas

```
  SYNC_OUT : out STD_LOGIC -- Salida sincrona
```

Arquitectura

```
signal sreg: std_logic_vector (1 downto 0); -- Se crea esta señal auxiliar
process (Reset,CLK)
begin
if reset='0' then
    sreg<=(others=>'0');
elsif rising_edge(CLK) then
    SYNC_OUT <= sreg(1);
    sreg<= sreg(0) & async_in;
end if;
end process;
```

De esta forma, ante una entrada asíncrona, esta solo se introduce en el sistema al llegar el flanco ascendente del reloj.

Entidad EDGEDTCTR (Detector de flancos)

Se trata de un detector de flancos para el acondicionamiento de la señal de entrada procedente de los botones. Al igual que en SYNCHRNZR se trata de la entidad utilizada en las prácticas de laboratorio de la asignatura. Esta entidad es necesaria puesto que las entradas por pulsador tienen una duración indeterminada, por lo que es necesario generar un único pulso de un ciclo de reloj cada vez que se pulsa el botón.

Entidad

Entradas

```
    CLK     : in STD_LOGIC; -- Señal de reloj
    SYNC_IN : in STD_LOGIC; -- Señal (sincrona)
    Reset   : in STD_LOGIC; -- Reset asincrono
```

Salidas

```
    EDGE : out STD_LOGIC; -- Indica cuando se ha producido un flanco negativo
```

Arquitectura

La arquitectura de esta entidad consiste en un registro de desplazamiento que va almacenando los tres últimos valores que ha tomado la señal de entrada, de manera que se puede comprobar si hay un flanco comparando el estado de la señal en los dos ciclos anteriores de reloj.

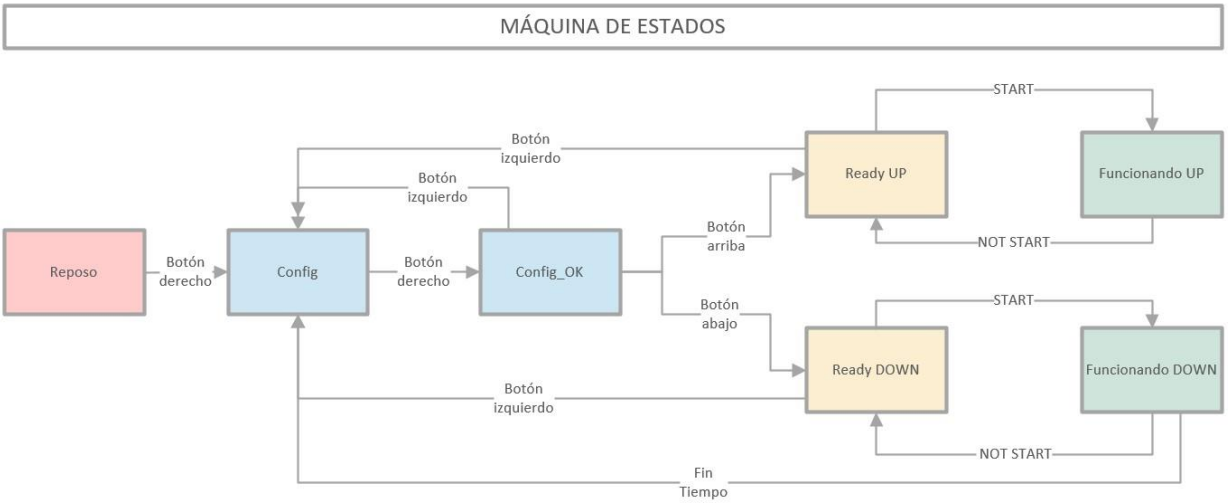
```
signal sreg: std_logic_vector (2 downto 0); -- Se crea esta señal auxiliar
process (Reset,CLK)
begin
if Reset='0' then
    sreg<=(others=>'0');
elsif rising_edge(CLK) then
    sreg<= sreg(1 downto 0) & sync_in;
end if;
end process;
```

Se detectan flancos negativos de manera que para que haya flanco los valores del registro tienen que ser 100 (botón pulsado en el primer ciclo y sin pulsar en los dos últimos).

```
    with sreg select
    EDGE <= '1' when "100",
            '0' when others;
```

Entidad FSM (Máquina de estados finita)

Es la máquina de estados que coordina el funcionamiento del sistema.



Entidad

Entradas

```
    TIME_FIN       : in  STD_LOGIC;
-- Se ha acabado el tiempo
    START          : in  STD_LOGIC;
-- Switch para pasar a funcionamiento
    BUTTON_CONFIG  : in  STD_LOGIC;
-- Boton para pasar a configuracion y ready
    BUTTON_BACK    : in  STD_LOGIC;
-- Boton para volver a la configuracion
    BUTTON_UP      : in  STD_LOGIC;
-- Boton para seleccionar cuenta hacia arriba
    BUTTON_DOWN    : in  STD_LOGIC;
-- Boton para seleccionar cuenta hacia abajo
    RESET          : in  STD_LOGIC;
-- Reset asincrono
    CLK            : in  STD_LOGIC;
-- Señal de reloj
```

Salidas

```
    UP_NDOWN       : out STD_LOGIC;
-- Habilitar cuenta hacia arriba o hacia abajo (salida negada)
    EN_MODOS       : out STD_LOGIC;
-- Habilitar configuracion de velocidad e inclinacion
    LOAD_EN_TIME   : out STD_LOGIC;
-- Habilitar configuracion de tiempo
    COUNT_EN_TIME  : out STD_LOGIC;
-- Habilitar la cuenta
    CRONO_EN_TIME  : out STD_LOGIC;
-- Habilitar CRONO
    MUX_SEL        : out STD_LOGIC;
-- Seleccionar o que se visualiza en los displays
    leds           : out std_logic_vector(6 downto 0)
```

Arquitectura

Funcionamiento

Los estados que forman parte de esta máquina de estados son los siguientes:

- Reposo: la máquina acaba de encenderse o resetearse. Desde este estado se pasa a la configuración pulsando el botón derecho.
- Configuración: en este estado el usuario puede configurar mediante los botones superior e inferior el tiempo que desea que funcione la cinta de correr. Una vez haya terminado de configurar el tiempo, se pasa a configuración_ok pulsando el botón derecho.
- Configuacion_ok: en este paso se tiene que elegir si se desea funcionar con un tiempo ascendente pulsando el botón superior (desde 0 hasta que se desee parar) o si se quiere funcionar con un tiempo descendente pulsando el botón inferior (desde el tiempo introducido anteriormente hasta 0). Si se pulsa el botón izquierdo, se vuelve a configuración (puede volver a configurar el tiempo deseado), mientras que si pulsa el botón derecho se pasa a ready.
- Listo (Ready): cada modo de funcionamiento tiene el suyo. El sistema está configurado y listo para comenzar a funcionar cuando se active START con el interruptor. Una vez más si se pulsa el botón izquierdo se vuelve a configuración.
- Funcionando: como se ha mencionado anteriormente, puede estar funcionando con una cuenta atrás de tiempo o con una cuenta hacia arriba. En ambos casos, mientras la máquina está en este estado, se puede configurar la velocidad de la cinta (subir de nivel con el botón superior y bajar con el inferior) y su inclinación (subir de nivel con el botón derecho y bajar con el izquierdo). En cualquier momento deja de funcionar si se desactiva START.

La máquina de estados es la encargada de ir haciendo funcionar unas entidades u otras, de manera que en función del estado en el que se encuentre, el funcionamiento del sistema es completamente distinto (por ejemplo: los botones realizan distintas acciones en función del contexto).

Entidad MODOS

Esta entidad es la encargada de controlar el funcionamiento de los niveles de velocidad e inclinación. Para ello, determina cuales son las entradas de interés en cada caso y trabaja con la entidad INCREMENT para obtener las salidas deseadas.

Entidad

Generic

```
    speed_width : positive := 4;
-- Número de bits de los datos de salida de velocidad
    incl_width  : positive := 4;
-- Número de bits de los datos de salida de inclinación
    conversion_speed : positive := 1;
-- Relación entre el nivel de velocidad y su correspondencia en km/h
    conversion_incl : positive := 1;
-- Relación entre el nivel de inclinación y su correspondencia en grados
    max : positive := 15
-- Nivel máximo a alcanzar para no sufrir un overflow
```

Entradas

```
    SPEED_UP   : in  STD_LOGIC; -- Botón para subir de nivel de velocidad
    SPEED_DOWN : in  STD_LOGIC; -- Botón para bajar de nivel de velocidad
    INCL_UP    : in  STD_LOGIC; -- Botón para subir de nivel de inclinación
    INCL_DOWN  : in  STD_LOGIC; -- Botón para bajar de nivel de inclinación
    CLK        : in  STD_LOGIC; -- Señal de reloj
    RESET      : in  STD_LOGIC; -- Reset asíncrono
    ENABLE     : in  STD_LOGIC; -- Chip enable
```

Salidas

```
    SPEED      : out STD_LOGIC_VECTOR (speed_width-1 downto 0);
-- Control del motor de velocidad
    INCL       : out STD_LOGIC_VECTOR (incl_width-1 downto 0);
-- Control del motor de inclinacion
    SPEED_DATA : out STD_LOGIC_VECTOR (speed_width-1 downto 0);
-- Nivel de velocidad activo (dato para mostrar en pantalla)
    INCL_DATA  : out STD_LOGIC_VECTOR (incl_width-1 downto 0)
-- Nivel de inclinación activo (dato para mostrar en pantalla)
```

Arquitectura

Entidades empleadas

Incrementador (INCREMENT) : permite realizar el aumento y decremento de los niveles según los botones que se pulsen con un contador. Se controla el máximo y el mínimo y devuelve los niveles correspondientes en cada momento (realizando la conversión que fuera necesaria).

```
component INCREMENT is
 Generic (

     out_width :positive ;
     conversion: positive;
     maximo: positive
 );
 Port (
     ENTRY_UP   : in  STD_LOGIC; --'1' equals +1
     ENTRY_DOWN : in  STD_LOGIC; --'1' equals -1
     CLK        : in  STD_LOGIC;
     RESET      : in  STD_LOGIC;
     ENABLE     : in  STD_LOGIC;
     OUTP       : out UNSIGNED (out_width-1 downto 0)
 );
end component;
```

Señales

Se han incluido estas dos señales para trabajar de forma interna.

```
signal speed_level : unsigned (speed_width-1 downto 0);
signal incl_level : unsigned (incl_width-1 downto 0);
```

Asignaciones a las entidades

En esta entidad, se ha utilizado INCREMENT tanto para controlar los niveles de velocidad como para los de inclinación. Por eso se han hecho dos instanciaciones.

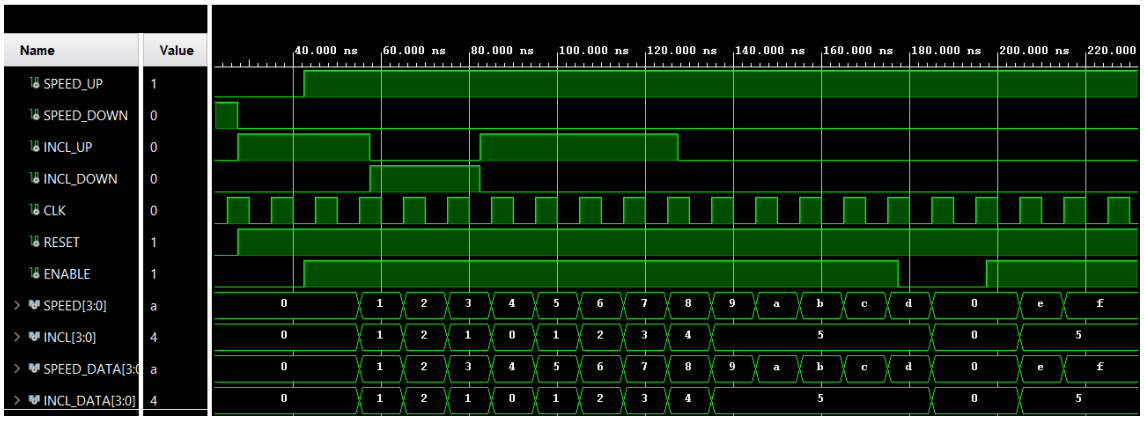
```
velocidad : INCREMENT
 generic map (
     out_width => speed_width,
      conversion =>  conversion_speed,
    maximo => max
 )
 port map (
     ENTRY_UP => SPEED_UP,
     ENTRY_DOWN => SPEED_DOWN,
     CLK => CLK,
     RESET => RESET,
      ENABLE => ENABLE,
     OUTP => speed_level
 );

inclinacion : INCREMENT
 generic map (
     out_width => incl_width,
```

```
     conversion => conversion_incl,
    maximo => max
)
port map (
    ENTRY_UP => INCL_UP,
    ENTRY_DOWN => INCL_DOWN,
    CLK => CLK,
    RESET => RESET,
    ENABLE => ENABLE,
    OUTP => incl_level
);
```

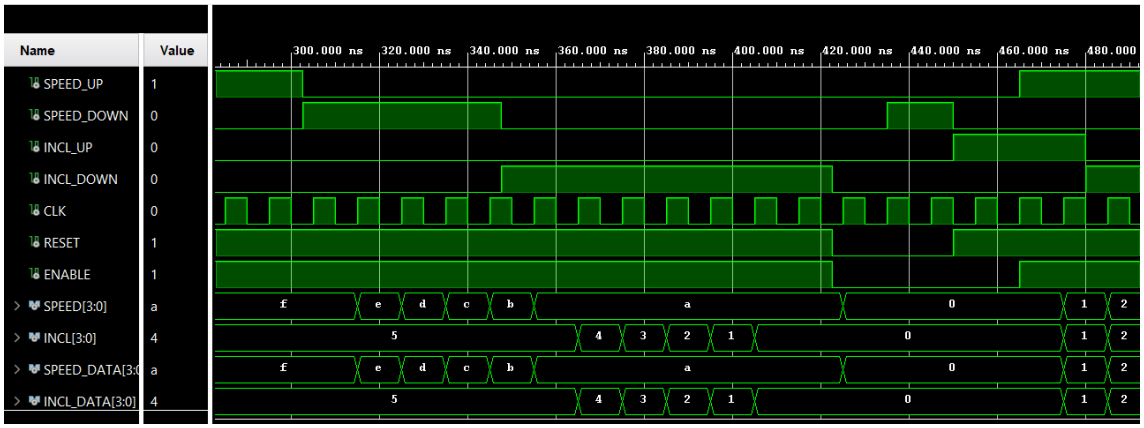
Implementación

```
process (CLK)
begin
if rising_edge (CLK) then
    if ENABLE = '1' then
        SPEED_DATA <= STD_LOGIC_VECTOR(speed_level);
        SPEED <= STD_LOGIC_VECTOR(speed_level / conversion_speed);
    else
        SPEED_DATA <= (others => '0');
        SPEED <= (others => '0');
    end if;

    if ENABLE = '1' then
        INCL_DATA <= STD_LOGIC_VECTOR(incl_level);
        INCL <=STD_LOGIC_VECTOR(incl_level / conversion_incl);
    else
        INCL_DATA <= (others => '0');
        INCL <= (others => '0');
    end if;
end if;
end process;
```

Simulaciones



Al igual que en CRONO, a la vez que se comprueba el correcto funcionamiento de la entidad MODOS, también se comprueba la aplicación de INCREMENT. Aquí se puede ver cómo hasta que no se ponen a nivel alto tanto el RESET como el ENABLE, no importa qué botones se esten pulsando. También se observa cómo los valores se quedan dentro de los límites establecidos. Observando que cuando se llega el mínimo, aunque se siga pulsando el botón de bajar los niveles no van a disminuir y lo mismo ocurre al alcanzar el máximo. Todas las salidas se actualizan con el flanco positivo del reloj. Se aprecia como las salidas en SPEED y SPEED_DATA (por ejemplo) coinciden, puesto que conversion = 1. Cuando ENABLE = 0, los valores vuelven a 0 (la cinta se para y deja de estar inclinada, no importando las entradas que esten activas), sin embargo, los valores de velocidad e inclinación que tenía antes de la parada se guardan, por lo que al volver a activarlo se vuelve a como estaba y la cuenta sigue a partir de ahí. Es una parada de seguridad, tras la que se puede seguir con el programa.



Por otra parte, si RESET = 0, los valores de velocidad vuelven a 0 pero esta vez, cuando se vuelve a activar si que se empieza la cuenta desde 0. Se trata del final del programa.

Entidad CRONO

La entidad Crono tiene como objetivo funcionar como un reloj que pueda realizar cuentas tanto ascendentes, hasta un máximo fijado por los parámetros genéricos de su entidad o contar decreciendo el contador desde una carga asignada. La propia entidad, mediante el uso de componentes de otras entidades se encarga también del control de dicha configuración de carga.

Entidad

Generic

```
    in_time_width : positive := 7;
-- Este valor es el ancho de datos de entrada, es decir de la carga
    out_time_width : positive := 13;
-- Este valor es el ancho de los datos de salida
    divide: positive:= 100000000;
-- Este valor es el encargado de fijar el valor para el cual el componente FREQ_DIV generará un flanco positivo, a partir de la señal de reloj propia de la placa de 100MHz
    conversion: positive:=30;
-- El valor conversion indica cuanto incrementará la carga por cada pulsación de entrada. Es decir, un pulso de boton de aumentar LOAD se reflejará internamente en la cuenta como la adición de 30 segundos
    max_time: positive:=5940
-- Es el máximo tiempo del reloj
```

Entradas

```
    UP_INC    : in  STD_LOGIC; -- Incremento de la carga
    DOWN_INC  : in  STD_LOGIC; -- Decremento de la carga
    UP_NDOWN  : in  STD_LOGIC; -- Habilitar cuenta hacia arriba o hacia abajo
    CE        : in  STD_LOGIC; -- Count enable
    CLK       : in  STD_LOGIC; -- Señal de reloj
    RESET     : in  STD_LOGIC; -- Reset asincrono
    ENABLE    : in  STD_LOGIC; -- Chip enable en la configuracion
    LOAD_ENABLE:  IN STD_LOGIC; -- WHEN '1' LOAD
```

Salidas

```
    TIMES     : out STD_LOGIC_VECTOR (out_time_width-1 downto 0);
-- Salida del cronometro en segundos
    LOAD_OUT: out STD_LOGIC_VECTOR (in_time_width-1 downto 0);
-- Carga asíncrona
    ENDING: out STD_LOGIC
-- Flag de finalización de la cuenta, utilizado en la entidad de la máquina de estados
```

Arquitectura

La arquitectura de la entidad se puede dividir en tres partes, la generación del pulso cada segundo, la modificación de la carga y por último el funcionamiento propio del reloj como cuenta ascendente y descendente.

Señales

```
 signal Load: UNSIGNED (in_time_width-1 downto 0);
 signal load_i : unsigned(Times'range);
```

Estas dos señales sirven para la carga, ya sea del componente de incremento y luego para la asignación dentro del process del propio CRONO

```
 signal count_i: unsigned(Times'range):=(others=>'0');
```

Señal interna para la cuenta en segundos, que al final será asignada a la salida de la cuenta.

```
 signal clk_1_sec: std_logic ;
```

Señal que funciona como salida del divisor de frecuencia y actúa como un pulso cada segundo

Configuración de la Carga

La carga se configura mediante el uso de un component del tipo INCREMENT:

```
 increment_load: INCREMENT
     generic map(
         out_width => in_time_width,
         conversion=>conversion,
         maximo=>max_time
     )
     port map(
         ENTRY_UP=>UP_INC,
         ENTRY_DOWN=>DOWN_INC,
         CLK=>CLK,
         RESET=>RESET,
         ENABLE=>LOAD_ENABLE,
         OUTP=>LOAD
     );
```

Configuración de la frecuencia de reloj

La configuración de la frecuencia para conseguir un flanco positivo cada segundo se consigue con el componente FREQ_ DIV:

```
component FREQ_DIV is
 Generic(
     DIVIDE  : positive := 10
 );
 Port (
     CLK_IN : in STD_LOGIC;
      RESET: in std_logic ;
     CLK_OUT : out STD_LOGIC
 );
end component;
```

Cuenta ascendente y descendente

La funcionalidad se realiza mediante un process up_down_counter:

```
process(CLK,clk_1_sec, RESET,LOAD_ENABLE)
```

Como lista de sensibilidad se incluyen las señales de reloj y las entradas asíncronas

Definición de Variables

Estas variables son propias del process y se utilizan para asignaciones

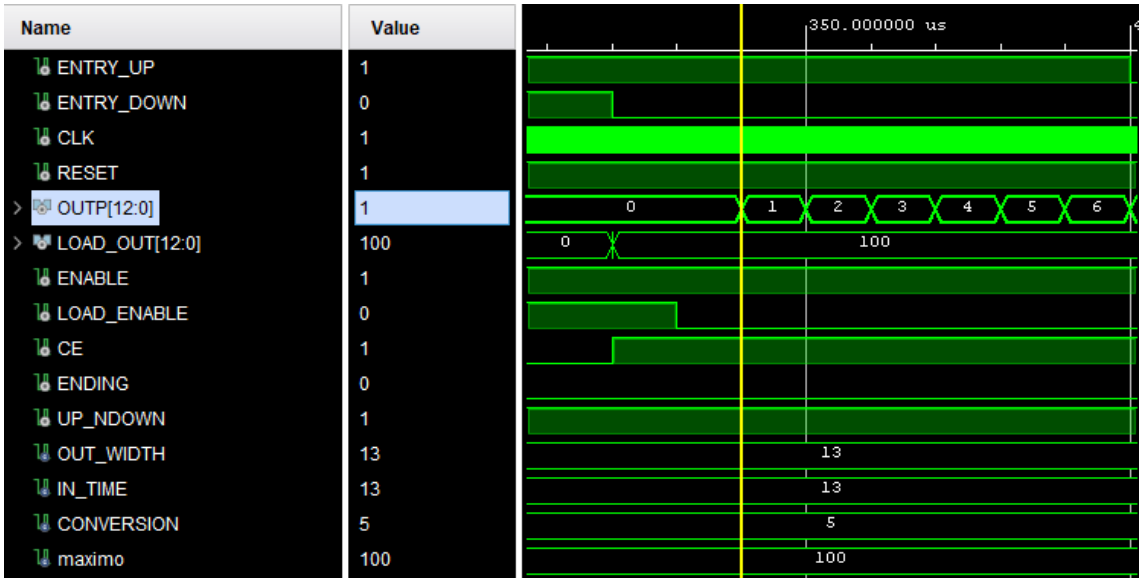
```
   variable ceros: unsigned(LOAD'length-1 downto 0):= (others=>'0');
   variable final: std_logic :='0';
-- se crea una variable para jugar con la logica que implica
   variable maxi: unsigned (Times'range):=to_unsigned(max_time, Times'length);
```

Código del reloj

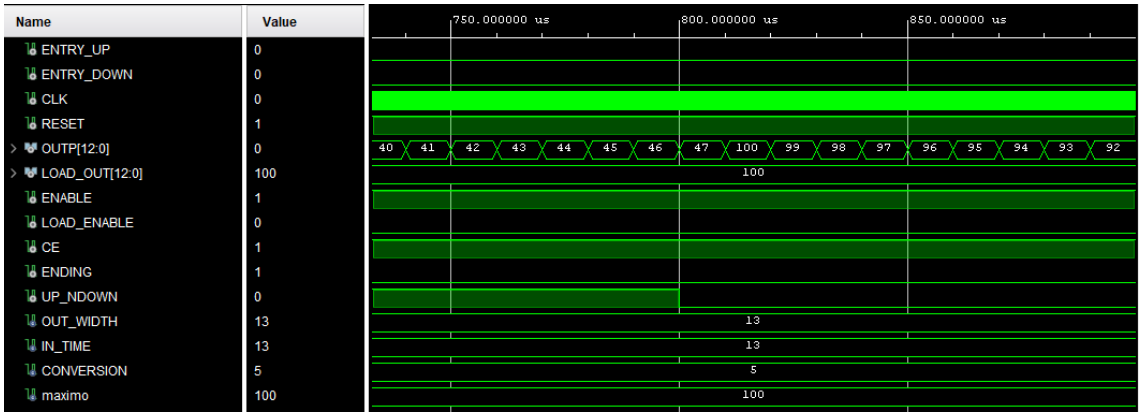
```
  rising_edge(clk_1_sec) THEN
              IF ENABLE='1' THEN
               if  CE='1'  THEN
                 if UP_NDOWN='1'  THEN
                       -- CONTADO HACIA ARRIBA DEL RELOJ HASTA máximo
                      count_i<=(count_i+1);
                          if count_i > maxi-1 then
--debido a que count_i es una señal, la comprobación de final debe hacerse para un valor anterior ya que así para el siguiente periodo ya valdrá 0 esta fuese una variable, la condicion de final sería la normal
                              count_i<=maxi;
                          end if;

                  ELSIF UP_NDOWN='0' and final='0' THEN  --CONTADO HACIA ABAJO
                     load_i<=load_i-1;
                     count_i<=load_i;
                          if load_i<=ceros then
--debido a que load_i es una señal, la comprobación de final debe hacerse para un valor anterior ya que así para el siguiente periodo ya valdrá 0
                              final:='1';
                          end if;
                     end if;
                  end if;
               end if;
              end if;
         ending<=final;
 end process;

          LOAD_OUT<=STD_LOGIC_VECTOR(LOAD);
          TIMES<=STD_LOGIC_VECTOR(count_i);
```

Simulaciones

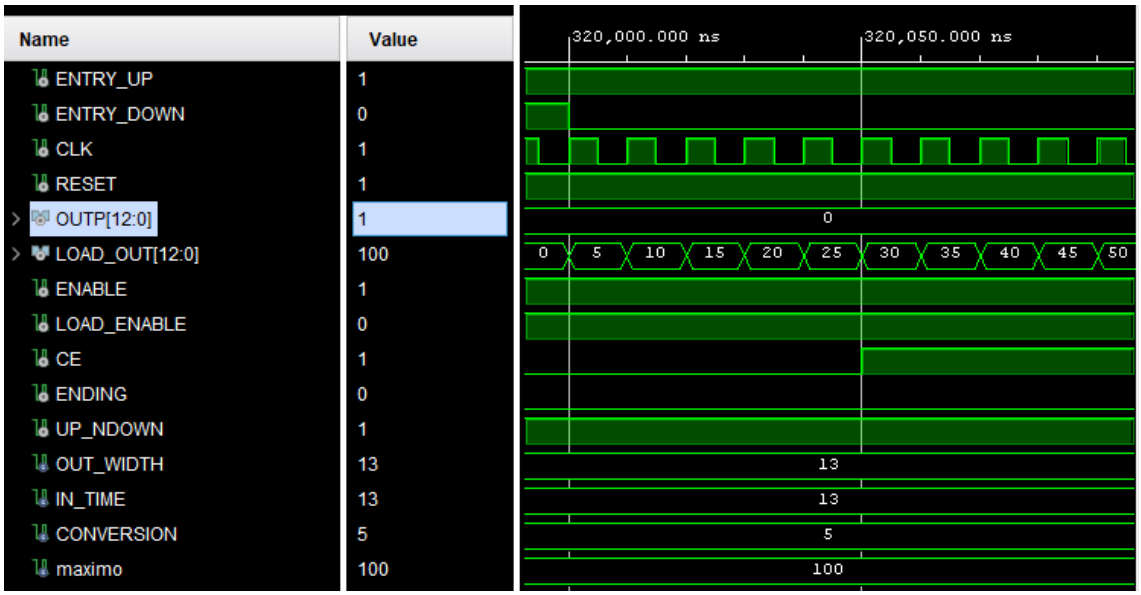
Se ha realizado un test bench para asegurar el correcto funcionamiento de la entidad. La cuenta se activa siempre y cuando se den las condiciones de CE= 1 ENABLE=1 y LOAD_Enable=0 y este activo uno de los dos modos de cuenta.



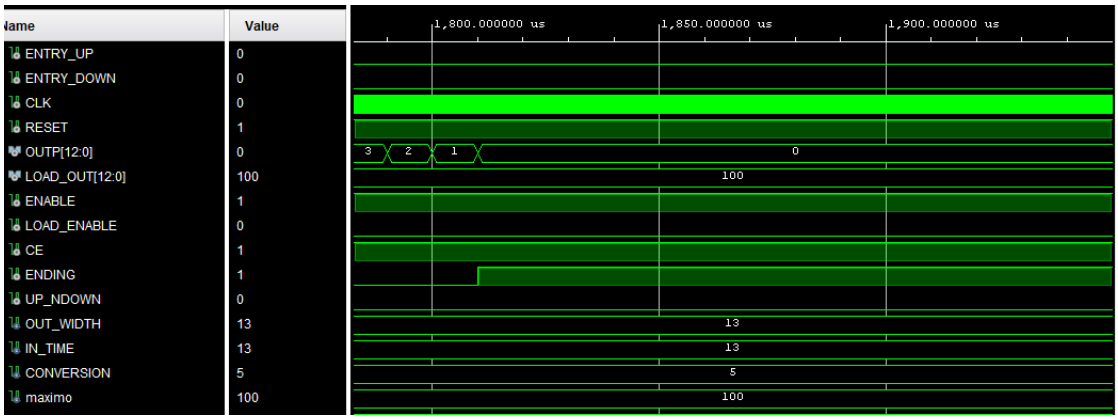
Se observa como cuando se activa la cuenta negativa la arquitectura espera a que haya un ciclo del nuevo reloj para empezar a contar hacia a abajo desde la carga.



Se comprueba también el correcto funcionamiento del componente increment ya que la carga se incrementa con cada entrada en la cantidad fijada. Si se activan las dos entradas a la vez la carga no se modifica.



Una vez la carga llega a cero el contador se detiene y se activa la señal de finalización.



- Entidad Increment

La función de esta entidad es incrementar una variable ante una entrada del sistema. Esta variable podrá ser incrementada o decrementada (según la entrada) entre unos valores mínimos y máximos y en saltos fijos (establecido en los genéricos).

Entity

Generic

```
     out_width :positive;
     conversion: positive;
-- conversion para que una entrada implique x en la salida
     maximo: positive
```

Entradas

```
    ENTRY_UP   : in  STD_LOGIC; -- '1' equals +1
    ENTRY_DOWN : in  STD_LOGIC; -- '1' equals -1
    CLK        : in  STD_LOGIC; -- Entrada de reloj
    RESET      : in  STD_LOGIC; -- Reset asincrono a nivel bajo
    ENABLE     : in  STD_LOGIC; -- Activación del chip
```

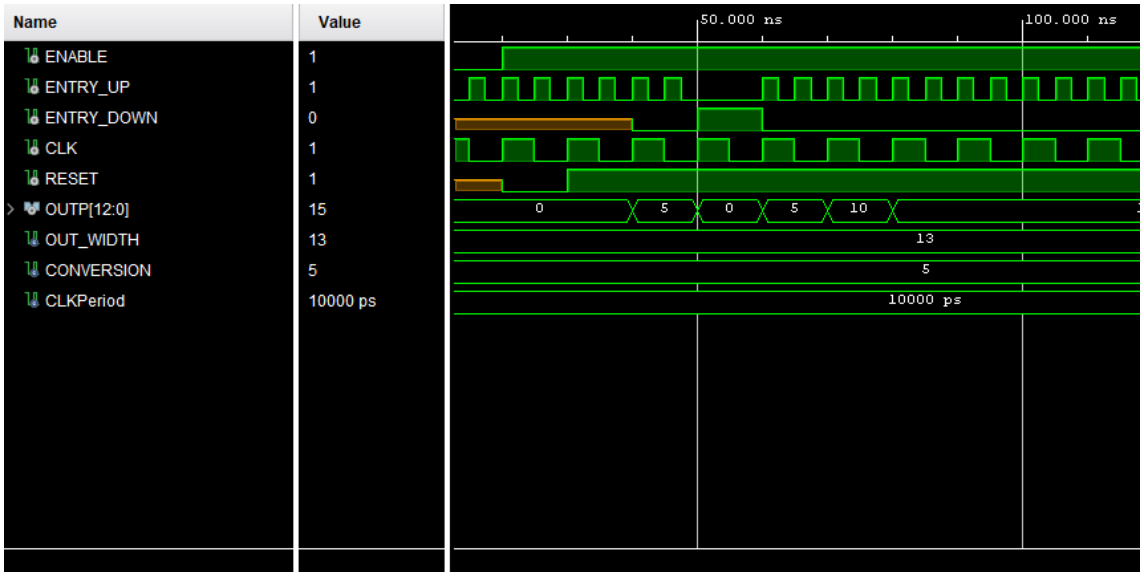
Salidas

```
    OUTP       : out UNSIGNED (out_width-1 downto 0)
-- Incremento total resultado de las diferentes entradas
```

Arquitectura

```
architecture Behavioral of INCREMENT is
 begin
   incrementador: process (RESET,CLK)
      variable increment: integer:=0;
      variable count: integer:=0;
-- se ha escogido usar variables internas dentro del process para no lidiar
con asignaciones después de los ciclos de reloj.
  begin
      if Reset='0' then -- reset de la salida y la cuenta interna
          increment:=0;
          count:=0;
      elsif rising_edge(CLK) and ENABLE='1' then -- cuenta síncrona
          if ENTRY_UP='1' and ENTRY_DOWN='0' and increment<maximo then
               count := count + 1;
               increment := (count) * conversion;
          end if ;
          if ENTRY_DOWN='1'and ENTRY_UP='0' and increment>0 then
               count := count - 1;
               increment := (count) * conversion;
          end if ;
    end if;
    OUTP<= TO_UNSIGNED (increment,OUTP'length) ;
  end process;
end Behavioral;
```

Simulaciones



En los resultados de la simulación podemos observar como el RESET a nivel bajo implica que no se realiza ningún incremento y se fija a 0 el valor de la salida. Una vez activado el nivel alto del RESET, la función del incremento de la carga se realizará con cada flanco de subida del reloj, siempre y cuando la señal de activación este a nivel lógico 1, al igual que el decremento, hasta llegar al máximo fijado, en este caso, 15.

- Entidad FREQ_DIV (Divisor de frecuencia)

Para el uso de los displays no se va a utilizar una frecuencia de 100MHz como tiene el reloj de la placa. Es más conveniente utilizar frecuencias mucho más bajas, que no llegan ni a los kHz. Por ello se crea esta entidad, para que esa parte del sistema utilice un reloj mucho más lento. Para poder ajustar posteriormente la frecuencia de reloj a lo que se quiera se ha implementado de forma que se puede alterar dicho valor cambiando el valor del genérico.

Debe tenerse en cuenta que el número de señales de reloj que pueden utilizarse en la placa es limitado. En este proyecto, por lo tanto, se limitará su uso al funcionamiento como señal de reloj de los displays y el decodificador de BDC a 7 segmentos asociado a ellos, funcionando el resto de la placa a la frecuencia de reloj original de la placa.

Entidad

Generic

```
  DIVIDE : integer := 10;
-- factor por el que se divide la frecuencia del reloj de entrada, que habitualmente será el reloj de la placa, de 100MHz. El valor por defecto es 10, de manera que la frecuencia de salida sería de 10MHz.
```

Ports

```
  CLK_IN : in std_logic;
 -- señal de reloj de entrada, habitualmente el de la placa, de 100MHz
  CLK_OUT : out std_logic;
-- señal de reloj de salida
  RESET : std_logic;
-- señal de reset.
```

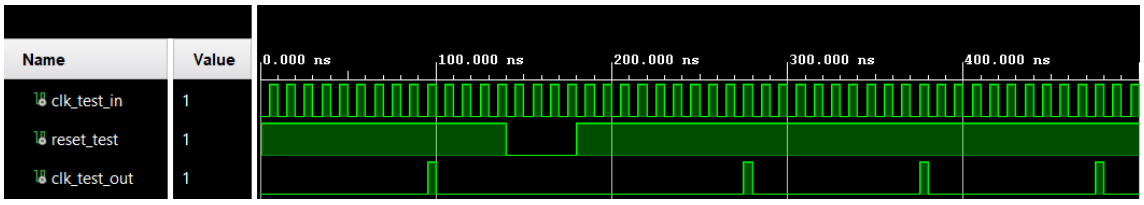
Aunque en teoría no es necesaria la señal de reset, se añade por si ocurriese algún fallo completamente ajeno al funcionamiento del programa, como que por interferencias, el valor del contador que internamente utiliza la entidad pasase a tener un número negativo

Arquitectura

Consiste en un contador cuyo valor se incrementa en una unidad en cada ciclo del reloj de entrada. Cuando el contador alcanza el valor especificado por el genérico "DIVIDE" la salida vale 1 durante un ciclo. La señal de reloj generada por lo tanto no es cuadrada, es rectangular, pero esto no tiene mayores consecuencias de cara a su uso como señal de reloj para otras entidades.

```
process (CLK_IN, RESET)
begin
    CLK_OUT <= '0';
    if (RESET = '0') then
        counter <= 0;
    elsif (rising_edge(CLK_IN)) then
        counter <= counter +1;
        if (counter = (DIVIDE-1)) then
            counter <= 0;
            CLK_OUT <= '1';
        end if;
    end if;
end process;
```

Simulaciones



Se trata de una entidad sencilla donde con un simple testbench se puede comprobar su funcionamiento. Se aprecia cómo cada 10 pulsos de reloj CLK_IN, se obtiene un pulso en CLK_OUT. Además, con el RESET se consigue reiniciar la cuenta de forma que una vez RESET = 1, se vuelven a empezar los 10 pulsos de CLK_IN necesarios para obtener uno de CLK_OUT.

Entidad MUX2A1 (Multiplexor 2 a 1)

En esta entidad, puesto que en CRONO se realiza tanto la carga inicial del tiempo, como la cuenta del reloj propiamente, se ha realizado un multiplexor para determinar si se trata del tiempo de carga o del reloj.

Entidad

Generic

```
   width : psoitive := 13
```

Entradas

```
    DATA1       : in  std_logic_vector(width-1 downto 0);
    DATA2       : in  std_logic_vector(width-1 downto 0);
-- Dos entradas del ancho deseado (configurable mediante el genérico destinado
a tal fin)
    SEL         : in  std_logic
entrada de selección de un bit que permita seleccionar entre las dos entradas
de datos
```

Salida

```
    DATA_OUTPUT : out std_logic_vector(width-1 downto 0);
```

Arquitectura

Se implementa en esta ocasión una arquitectura de tipo BEHAVIORAL.

```
process (SEL)
begin
    if (SEL = '1') then
        DATA_OUTPUT <= DATA2;
    else
        DATA_OUTPUT <= DATA1;
    end if;
end process;
```

Entidad DISP_CTRL (Control de los displays)

Esta entidad es la entidad encargada de representar toda la información necesaria en los displays. Para ello, recibe los datos de interés de otras entidades en formato binario sin signo. Posteriormente, debe convertir los datos a BCD para poder representarlos en los displays. Adicionalmente, debe controlar que los displays se vayan encendiendo de uno en uno con la cifra correspondiente. Para hacer todo esto, esta entidad está dividida a su vez en entidades que realizan tareas de menor complejidad.

Aunque en la placa empleada para la implementación del proyecto todos los displays se encuentran juntos, en la realidad la representación de cara al usuario sería algo de la siguiente forma, como ya se adelantaba en la introducción del proyecto. No obstante, el orden de los displays es el mismo que en la imagen y el funcionamiento sería análogo.

Entidad

Generic

```
      speed_width : positive := 4;
-- Número de bits de los datos de velocidad
      incl_width  : positive := 4;
-- Número de bits de los datos de inclinación
      time_width  : positive := 13;
-- Número de bits de los datos de tiempos
      digits      : positive := 8
-- Número de dígitos de la placa
      digits_range : positive := 3
-- Rango para la expresion del numero de digitos de la placa en binario
```

Entradas

```
       SPEED        : in STD_LOGIC_VECTOR (speed_width-1 downto 0);
-- Dato de velocidad (0 a 16)
       INCL         : in STD_LOGIC_VECTOR (incl_width-1 downto 0);
-- Dato de inclinación (0 a 16)
       TIME_DATA    : in STD_LOGIC_VECTOR (time_width-1 downto 0);
-- Dato de tiempo (en segundos)
       CLK          : in STD_LOGIC);
-- Señal de reloj
       RESET        : in std_logic;
-- Señal de RESET asincrono
```

Salidas

```
       DISP_SEL     : out STD_LOGIC_VECTOR (digits-1 downto 0);
-- Selección del dígito (ira rotando para encender cada vez un digito)
       DISP_DATA    : out STD_LOGIC_VECTOR (6 downto 0); -- 7 segmentos
```

Arquitectura

Funcionamiento

La entidad DISP_CTRL es una entidad que une una serie de entidades que son las que realizan cada una de las tareas. Es por ello que se emplea una arquitectura estructural.

Por una parte es necesario convertir los datos recibidos a valores de una sola cifra y a BCD. Para ello se emplea la entidad DEC2BCD (Decimal a BCD), que recibe números codificados en binario de hasta dos cifras obteniendo una cifra por separado en cada una de las salidas. Estas salidas en BCD ya pueden ser interpretadas posteriormente por el decodificador de BCD a 7 segmentos.

Nótese que se ha hablado de que los datos recibidos son minutos y segundos, pero la entrada de esta entidad DISP_CTRL es el tiempo en segundos. Para ello se ha creado una entidad (TIME_CONV) que convierte los segundos a minutos y segundos.

Todo esto permite hacer llegar a los displays la información necesaria para el control de los 7 segmentos. Sin embargo, como solo se puede controlar un display de 7 segmentos cada vez, hay que añadir una entidad que tiene una doble funcionalidad: por un lado permite elegir cuál de los displays está encendido en cada momento y por otra parte selecciona que datos llegan al decodificador de BCD a 7 segmentos. De esta forma se puede utilizar un único decodificador en lugar de que cada uno de los dígitos tenga asociado uno.

Para lo anterior, la frecuencia de funcionamiento puede ser mucho menor de 100MHz, puesto que el ojo humano a partir de unas decenas de hercios no va a percibir que los displays no están encendidos todo el tiempo y que se van encendiendo de uno en uno. Para utilizar una frecuencia menor se emplea un divisor de frecuencia.

Señales

```
signal dig_sel2decod_and_mux : std_logic_vector (digits_range-1 downto 0);
signal mux2dcdr              : std_logic_vector(3 downto 0);
signal internal_clk          : std_logic;`
signal speed7b  : std_logic_vector(6 downto 0);`
signal incl7b   : std_logic_vector(6 downto 0);`
signal data1    : std_logic_vector(3 downto 0);`
signal data2    : std_logic_vector(3 downto 0);`
signal data3    : std_logic_vector(3 downto 0);`
signal data4    : std_logic_vector(3 downto 0);`
signal data5    : std_logic_vector(3 downto 0);`
signal data6    : std_logic_vector(3 downto 0);`
signal data7    : std_logic_vector(3 downto 0);`
signal data8    : std_logic_vector(3 downto 0);`
signal minutos  : std_logic_vector(6 downto 0);`
signal segundos : std_logic_vector(6 downto 0);`
```

Entidades empleadas

- TIME_CONV (convertidor de tiempo): separa el tiempo (que se trabaja en segundos) en los minutos y segundos correspondientes (formato de reloj).

```
conversor_tiempo : TIME_CONV
    Generic map(
        input_width => time_width
        )
    Port map(
        DATA_IN => TIME_DATA,
        CLK => CLK,
        MIN => minutos,
        SEC => segundos
        );
```

- DEC2BCD (Convertidor de binario a decenas y unidades): separa los números en decenas y unidades (tanto los niveles de velocidad e inclinación como el tiempo) para mostrar cada dígito en un display.

```
speed_BCD_conversion : DEC2BCD Port map(
    DATA_IN => speed7b,
    CLK => CLK,
    BCD1 => data1, --Decenas
    BCD2 => data2 --Unidades
    );
incl_BCD_conversion : DEC2BCD Port map(
    DATA_IN => incl7b,
    CLK => CLK,
    BCD1 => data3, --Decenas
    BCD2 => data4 --Unidades
    );
minutes_BCD_conversion : DEC2BCD Port map(
    DATA_IN => minutos,
    CLK => CLK,
    BCD1 => data5, --Decenas
    BCD2 => data6 --Unidades
    );
seconds_BCD_conversion : DEC2BCD Port map(
    DATA_IN => segundos,
    CLK => CLK,
    BCD1 => data7, --Decenas
    BCD2 => data8 --Unidades
    );
```

- DIV_FREC (divisor de frecuencia): permite utilizar una frecuencia más baja que la del reloj de la placa para el control de los displays de 7 segmentos.

```
divisor_de_frecuencia : FREQ_DIV
    Generic map(
    DIVIDE => 10000 -- Este valor se ha cambiado a 2 para el TESTBENCH
    )
    Port map(
        CLK_IN => CLK,
        RESET => RESET,
        CLK_OUT => internal_clk
    );
```

- DIG_SEL (selector de dígitos): selecciona un dígito cada vez.

```
selector_de_digito : DIG_SEL Port map(
    CLK => internal_clk,
    RESET => RESET,
    UINT_SEL => dig_sel2decod_and_mux,
    BIN_SEL =>  disp_sel
    );
```

- MUX (multiplexor): permite seleccionar los datos que recibe el decodificador de BCD a 7 segmentos, es decir, determina qué dato va en cada display.

```
multiplexor : MUX Port Map(
    DATA_IN1 => data1,
    DATA_IN2 => data2,
    DATA_IN3 => data3,
    DATA_IN4 => data4,
    DATA_IN5 => data5,
    DATA_IN6 => data6,
    DATA_IN7 => data7,
    DATA_IN8 => data8,
    SEL => dig_sel2decod_and_mux,
    OUTPUT => mux2dcdr
    );
```

- DCDR7SEG (Decodificador de BCD a 7 segmentos): manejo de los displays (según el número que se quiere representar, se activan unos segmentos u otros).

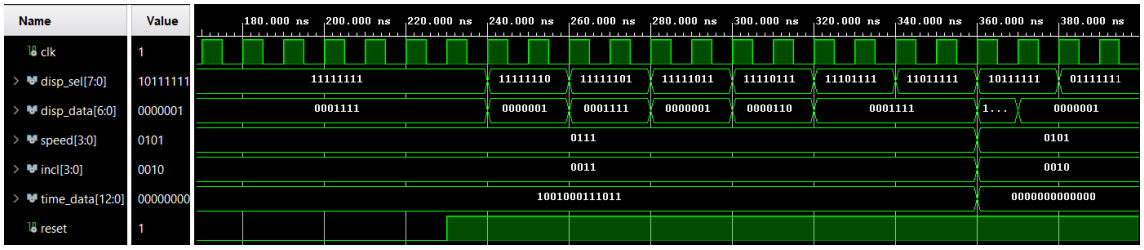
```
decod7seg : DCDR7SEG Port map(
  code => mux2dcdr,
  led => DISP_DATA
  );
```

Displays en funcionamiento

En las siguientes imágenes a "cámara lenta" se puede apreciar como se van iluminando los displays de uno en uno gracias a la selección hecha por DISP_SEL. Mientras que se ilumina cada uno de los displays, en paralelo se introducen los datos adecuados en el decodificador de BCD a 7 segmentos gracias al multiplexor de 8 entradas.

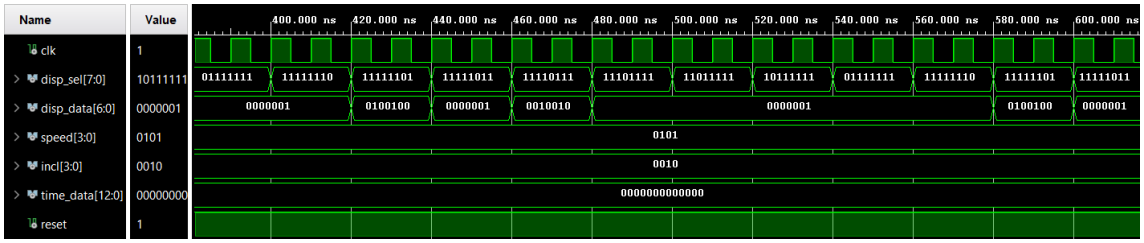
*Debido a la velocidad a la que cambian los displays (a pesar de que incluso se redujo para poder grabar el video) en algunas de las imágenes se ven dos displays encendidos a la vez. Es en cierto modo el efecto que se pretende conseguir de cara al ojo humano, haciendo que parezca que todos los displays están encendidos a la vez.

Simulaciones



Con el testbench se puede ver cómo mientras RESET = 0, ningún display se encuentra activo, por lo que no se va a ver nada en la pantalla. Una vez RESET = 1:

- DISP_SEL va cambiando (11111110, 11111101, etc), activando (a nivel bajo) el display correspondiente en cada momento.
- DISP_DATA se encarga de encender los leds correspondientes al número que tiene que mostrar en el display activo.



Así por ejemplo, cuando SPEED = 0101 (05), INCL = 0010 (02) y TIME_DATA = 0000000000000 (00:00):

- Cuando DISP_SEL = 11111110, en el display de la izquierda se activan los segmentos correspondientes a un 0 y cuando DISP_SEL = 11111101, en el display contiguo se activan los segmentos correspondientes a un 5. Mostrando de forma conjunta la velocidad 05.
- De igual forma cuando DISP_SEL = 11111011, en el display de la izquierda se activan los segmentos correspondientes a un 0 y al activar DISP_SEL = 11110111, en el display contiguo se activan los segmentos correspondientes a un 2. Mostrando de forma conjunta la inclinación 02.

Así se va actualizando y mostrando la información correspondiente en cada display y cuando llega a la derecha vuelve a empezar el ciclo. De esta forma también se observa cómo cuando los valores de entrada cambian, pueden tardar un poco en actualizarse, o no hacerlo de izquierda a derecha, puesto que la selección de dígitos sigue su ciclo. Sin embargo, esto no supone ningún problema puesto que se hace a una frecuencia muy elevada, no siendo perceptible para el ojo humano.

*NOTA: el divisor de frecuencia ha sido reconfigurado para el test para poder apreciar los cambios en un menor tiempo. En realidad, la velocidad de actualización de los displays sería del orden de 1000 veces superior.

- Entidad TIME_CONV (Convertidor a minutos y segundos)

Puesto que a la hora de contar se trabaja en segundos, es necesario convertir esos valores a minutos y segundos para poder expresarlos en los displays de una forma clara, en el formato de hora.

Entidad

Generic

```
input_width : positive := 13 -- Ancho del tiempo de entrada. Por defecto es de 13 bits, porque permite representar hasta el equivalente a 99 minutos en segundos, que es el máximo tiempo que se puede representar con los displays de 7 segmentos disponibles en la placa
```

Entradas

```
  DATA_IN     : in std_logic_vector(input_width-1 downto 0);
--Entrada de datos en segundos, procedentes de la entidad CRONOS
  CLK         : in std_logic;
--Entrada de reloj
```

Salidas

```
  MIN         : out std_logic_vector(6 downto 0); --Salida en minutos
  SEC         : out std_logic_vector(6 downto 0) -- Salida en segundos
```

Se emplea un ancho de 7 bits que es el que permite representar hasta 128 en decimal. Esto es porque para los displays no se va a poder emplear un ancho mayor.

Arquitectura

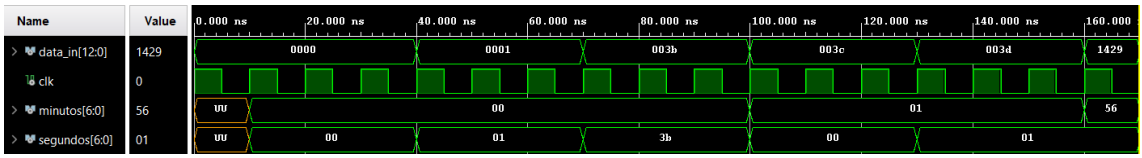
La arquitectura internamente va a utilizar tres variables:

```
    variable entrada    : unsigned (DATA_IN'range);
    variable iminutos   : unsigned (DATA_IN'range);
    variable isegundos  : unsigned (DATA_IN'range);

entrada := unsigned(DATA_in);
-- se conveirten los datos de entrada a unsigned para poder hacer operaciones matematicas
    iminutos := entrada/60;
    isegundos := entrada mod 60;
```

El tamaño de las entradas no es el mismo que el tamaño de las salidas, por lo que a las salidas solo deben asignarse tantos bits como ancho tengan estas últimas. Se asignan los bits más bajos. Los bits que quedan sin asignar, si se ha calculado bien el ancho de las salidas en función de los valores más altos que pueden resultar de las operaciones realizadas. En el caso concreto de este proyecto, el valor máximo de los segundos es 60 (se necesitan 6 bits) y el valor máximo de minutos no debería ser superior a 99 minutos (para lo que se requieren 7 bits, aunque no se comprueba en esta entidad que el número de minutos sea inferior a 99).

```
MIN <= std_logic_vector(iminutos(6 downto 0));
SEC <= std_logic_vector(isegundos(6 downto 0));
```

Simulaciones



A partir del testbench se puede observar cómo ante una entrada en segundos, te devuelve el valor convertido a minutos con sus segundos correspondientes. Así cada parte se asigna a una salida, que más tarde (con las conversiones necesarias de unidades y decenas) se podrá mostrar en el display en el formato de un reloj de minutos:segundos.

- Entidad DEC2BCD (Convertidor a decenas y unidades)

Esta entidad se utiliza para convertir los valores con los que se trabaja (en binario), separando las decenas y las unidades para poder mostrar cada una en un display.

Entidad

Entradas

```
    DATA_IN : in  std_logic_vector(6 downto 0); -- Entrada (numero de 0 a 99)
    CLK     : in  std_logic; -- Señal de reloj
```

Salidas

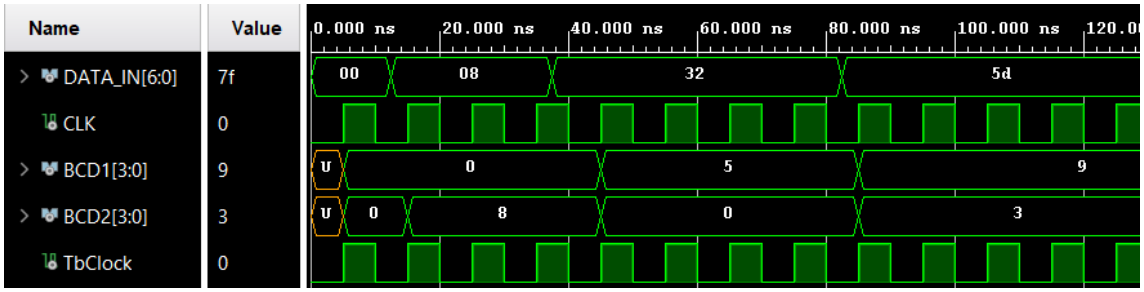
```
    BCD1    : out std_logic_vector(3 downto 0); -- Decenas
    BCD2    : out std_logic_vector(3 downto 0)  -- Unidades
```

Arquitectura

A partir de la entrada, se calculan las decenas dividiendo dicho número entre 10, y las unidades con el módulo de 10 de este. Así, luego cada valor se asigna a una salida.

```
 process (clk)
    variable decenas    : unsigned (DATA_IN'range);
    variable unidades   : unsigned (DATA_IN'range);
    variable entrada    : unsigned (DATA_IN'range);
begin
    if rising_edge(CLK) then
        entrada := unsigned(DATA_IN);
        decenas := entrada/10;
        unidades := entrada mod 10;
        BCD1 <= std_logic_vector(decenas(3 downto 0));
        BCD2 <= std_logic_vector(unidades(3 downto 0));
    end if;
end process;
```

Simulaciones



En el testbech se observa como ante una entrada en binario de 7 bits, se obtiene su resultado en dos salidas: una que contiene las decenas de dicho número en decimal y la otra con las unidades.

- Entidad DIG_SEL (selección del dígito)

Para poder controlar todos los displays de 7 segmentos se debe ir iluminando uno cada vez (puesto que el control de cada uno de los segmentos es común para los 8 displays de la placa). Con el fin de ir seleccionando cada uno de los displays y la información que le debe llegar a cada uno (mediante un multiplexor) se utiliza esta entidad.

Entidad

Generic

```
    DIGITS      : positive:=8; -- numero de displays
    DIGITS_RANGE: positive:=3 -- bits necesarios para codificar el num displays
```

Entradas

```
    CLK      : in std_logic;
    RESET    : in std_logic;
```

Salidas

```
    UINT_SEL: out std_logic_vector (DIGITS_RANGE-1 downto 0);
-- salida en binario (para utilizarla en multiplexor como entrada de selección)
    BIN_SEL : out std_logic_vector (DIGITS-1 downto 0)
-- seleccion del display
```

Arquitectura

El funcionamiento es relativamente parecido al del divisor de frecuencia, con la diferencia de que esta vez la salida es el contador interno que el divisor de frecuencia utilizaba. Se crea un contador interno que va aumentando en una unidad en cada ciclo de reloj. El valor de ese contador representa el dígito que está seleccionado en cada momento.

Por una parte se debe escribir la salida en binario, para lo cual hay que implementar un decodificador que active una de las ocho líneas que controlan los terminales comunes de los displays. Aunque podría hacerse en una entidad separada, dada la sencillez e inmediatez del código se implementa directamente aquí mediante un proceso:

```
signal contador : integer := 0; -- Señal auxiliar

process(CLK)
begin
    if (RESET = '0') then
        contador <= 0;
    elsif rising_edge(CLK) then
        contador <= contador + 1; -- Cada vez se va a seleccionar un display
(del 0 al 7)
        if (contador >= DIGITS-1) then
            contador <= 0;
        end if;
    end if;
end process;
```

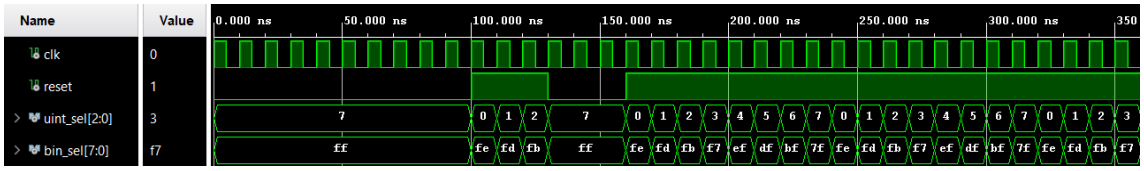
```
asignacion_salida: process (CLK)
begin
    if Reset='0' then
        BIN_SEL <= (others => '1');
        UINT_SEL <= (others=> '1');
    elsif rising_edge (CLK) then
        BIN_SEL <= (others => '1');
        BIN_SEL(contador) <= '0'; -- Se pone a 0 el bit correspondiente al
display seleccionado y los demas a 1
        UINT_SEL <= std_logic_vector(to_unsigned(contador, DIGITS_RANGE));
-- Se convierte ese valor a binario
        end if;
    end process;
```

Como se puede observar se ha implementado también en el mismo proceso la asignación de la salida para la selección en el multiplexor, que es directamente el valor del contador convertido a std_logic_vector. Por otra parte, cuando RESET se encuentra activo se ponen todas las salidas a 1 para que no se encienda ningún display. La salida para la selección del canal mediante el multiplexor (UINT_SEL) se pone toda a 1, lo que implica escoger un canal en el multiplexor, pero esto de cara al usuario es irrelevante porque al estar apagados los displays no se verá la información seleccionada.

Funcionamiento de la selección de dígitos

Se muestra como se van seleccionando los displays de uno en uno. (En algunas de las imágenes aparecen dos iluminados por la velocidad a la que cambian. Este de hecho es el efecto que se quiere conseguir de cara al ojo humano).

Simulaciones



Gracias al testbench se puede comprobar como cuando RESET = 0, no se encuentra ningún display seleccionado, por lo que UINT_SEL = 7 y BIN_SEL = ff. Sin embargo, una vez RESET = 1, a cada flanco ascendente del reloj se va seleccionando un display, empezando por 0 (en binario fe, puesto que se trata del primer display), hasta llegar al 7 (en binario 7f), desde donde vuelve otra vez a 0, empezando el ciclo de nuevo.

- Entidad MUX (Multiplexor)

Se trata de un multiplexor con 8 entradas y 1 salida. Es una entidad puramente combinacional y por ello no es necesario que disponga ni de RESET ni de señal de reloj. Se encarga de elegir qué entra al decodificador de BCD a 7 segmentos.

Entidad

Generic

```
        width   : positive := 3
-- seleccionar el ancho de las entradas y por lo tanto de la salida también
```

Entradas

```
        DATA_IN1    : in STD_LOGIC_VECTOR (width-1 downto 0);
        DATA_IN2    : in std_logic_vector (width-1 downto 0);
        DATA_IN3    : in STD_LOGIC_VECTOR (width-1 downto 0);
        DATA_IN4    : in std_logic_vector (width-1 downto 0);
        DATA_IN5    : in STD_LOGIC_VECTOR (width-1 downto 0);
        DATA_IN6    : in std_logic_vector (width-1 downto 0);
        DATA_IN7    : in STD_LOGIC_VECTOR (width-1 downto 0);
        DATA_IN8    : in std_logic_vector (width-1 downto 0);
-- 8 entradas de datos del ancho que se indique en el genérico
        SEL         : in std_logic_vector (2 downto 0)
-- entrada de selección de 3 bits, para poder seleccionar entre las 8 entradas.
```

Salidas

```
        OUTPUT      : out STD_LOGIC_vector(width-1 downto 0);
```

Arquitectura

Se ha implementado una arquitectura de tipo DATAFLOW con un único with - select.

```
with SEL select
    OUTPUT  <=  DATA_IN1 when "000",
                DATA_IN2 when "001",
                DATA_IN3 when "010",
                DATA_IN4 when "011",
                DATA_IN5 when "100",
                DATA_IN6 when "101",
                DATA_IN7 when "110",
                DATA_IN8 when "111",
                (others => '0') when others;
```

- Entidad DCDR7SEG (Decodificador de BCD a 7 segmentos)

Se trata de un decodificador de BCD a 7 segmentos como los empleados en el laboratorio de la asignatura. La implementación en Hardware consiste simplemente en un decodificador. Esta entidad no tiene ninguna función adicional.

Al tratarse de un decodificador cuya lógica es completamente combinacional, sin ningún elemento de memoria, se puede implementar sin señal de reloj, siendo por lo tanto una entidad asíncrona.

Entidad

Entradas

```
    code : IN std_logic_vector(3 DOWNTO 0);
-- Entrada en BCD. Tiene un ancho de 4 bits fijo, es decir, no se emplea un genérico porque este valor es siempre igual para el código BCD
```

Salidas

```
    led : OUT std_logic_vector(6 DOWNTO 0)
-- Salidas para el control de los displays de 7 segmentos. De nuevo no se utiliza un genérico para el ancho del vector porque el valor es siempre el mismo
```

Arquitectura

Es una arquitectura de tipo DATAFLOW. Tan solo consiste en una condición implementada con with-select. En función de la entrada en BCD se traduce a los segmentos del display que se tienen que iluminar. Las salidas son a nivel bajo.

```
WITH code SELECT
    led <=  "0000001" WHEN "0000",
            "1001111" WHEN "0001",
            "0010010" WHEN "0010",
            "0000110" WHEN "0011",
            "1001100" WHEN "0100",
            "0100100" WHEN "0101",
            "0100000" WHEN "0110",
            "0001111" WHEN "0111",
            "0000000" WHEN "1000",
            "0000100" WHEN "1001",
            "1111110" WHEN others;
```

Constraints (Restricciones)

Siguiendo la plantilla de la Nexys-4-DDR-Master, se han activado los componentes necesarios para trabajar con la placa.

Clock signal

```
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports CLK ];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz

create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
CLK];
```

Interruptores

Únicamente se emplea un interruptor, como medida de seguridad, para activar y desactivar el sistema.

```
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports {
START }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
```

LEDs

Los LEDs se han activado para la comprobación del correcto funcionamiento del sistema. De forma que unos (los centrales) muestran en que estado se encuentra el programa, y otros (los de los laterales) muestran en que nivel de velocidad e inclinación (en binario) se encuentra la máquina.

Display 7 segmentos

Aquí se activan todos segmentos y todos los displays (8).

Botones

Se van a necesitar tanto el botón del RESET, como los cuatro botones activados para avanzar en el programa y subir y bajar de niveles y el tiempo (el del medio no se usa).