



POLITÉCNICA

UNIVERSIDAD
POLITÉCNICA
DE MADRID

CRONÓMETRO VHDL

SISTEMAS ELECTRÓNICOS DIGITALES

Realizado por:

Raúl Herranz Rodríguez - EE403 - 54928

Carlos Murillo Pagador - EE403 - 54933

Alejandro Sacristán Jiménez – EE403 - 54936

Repositorio: <https://github.com/asacristanj/Cronometro-VHDL-Trabajo-SED-2021->

ÍNDICE

1. INTRODUCCIÓN.....	2
2. INSTRUCCIONES GENERALES.....	2
3. DIAGRAMA GENERAL	3
4. ESTRATEGIAS Y ALGORITMOS.....	3
5. DESCRIPCIÓN VHDL DE LOS BLOQUES FUNCIONALES.....	4
5.2. Modo cronómetro.....	6
5.2.1. Prescaler	6
5.2.2. Entidad Modo Cronómetro	7
5.3. Control Ánodo.	9
5.3.1. Decodificador	9
5.3.2. Entidad Control Ánodo.	9
5.4. Entidad Top	12
6. SIMULACIÓN	14
6.1. Simulación Modo Cronómetro.....	14
6.2. Simulación Control Ánodo.	16
7. ANEXO. CRONÓMETRO DE TRES FUNCIONES	18
7.1. Introducción.....	18
7.2. Instrucciones.....	18
7.3. Diagrama General	20
7.4. Estrategia y algoritmos.	20
7.5. Descripción VHDL de los bloques funcionales.	21
7.6. Simulación.....	37
8. CONSTRAINS.....	39

1. INTRODUCCIÓN

El objetivo del trabajo consiste en realizar un cronómetro utilizando VHDL como lenguaje de descripción Hardware, e implementarlo en una FPGA Nexys DDR4.

A través de 4 de los 8 displays de 7 segmentos se mostrará el tiempo transcurrido desde que se ha pulsado el botón que denominaremos “Start”, mientras que en los otros 4 se mostrará de forma permanente el mensaje “Cr--”, como diminutivo de cronometro. Por lo que el primer reto que debemos afrontar será el de mostrar diferentes dígitos en cada uno de los displays, ya que los pines de conexión a los 7 segmentos son comunes a los 8 displays, por lo que debemos controlar el ánodo de cada uno de ellos y sincronizarlo de manera adecuada.

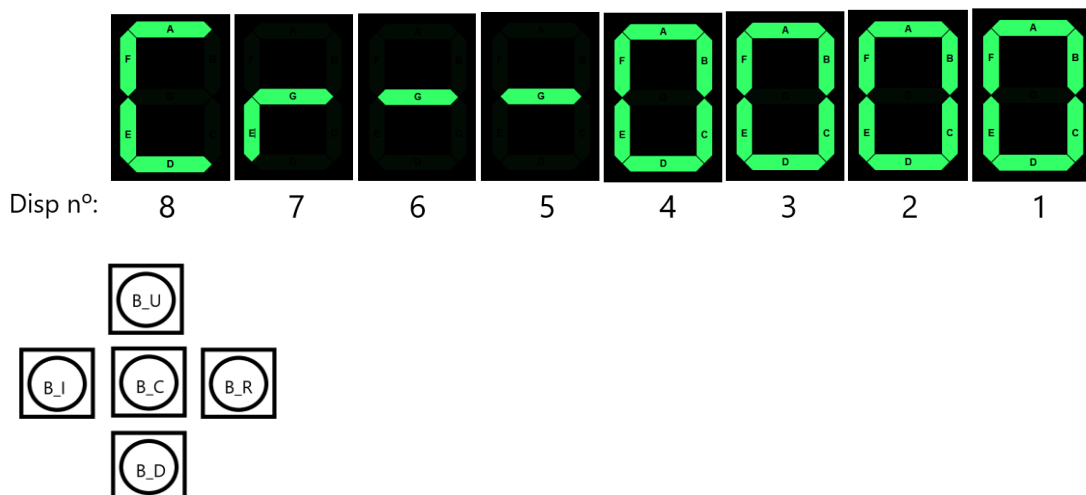
Nuestro segundo reto será crear una entidad capaz de gestionar la temporización y modificar el valor de las señales de salida en tiempo real al transcurrir el tiempo, así como gestionar de manera correcta las entradas de “Pausa” y “Reset” utilizando una máquina de estados.

Por otra parte, dispondrá de una entrada de habilitación “Enable_A” que deberá estar a nivel alto para poder iniciar la temporización. Esta entrada tiene el objetivo de hacer que este cronometro pueda ser implementado en otros diseños de mayor envergadura, como por ejemplo un cronometro con varios modos.

Las entradas de nuestro diseño serán los botones propios de la placa, debido a la alta cantidad de ruido que pueden generar estos, se deberá implementar un sincronizador con el objetivo de reducir el ruido y evitar, en la medida de lo posible, los rebotes que se pudiesen general.

2. INSTRUCCIONES GENERALES.

Interfaz:



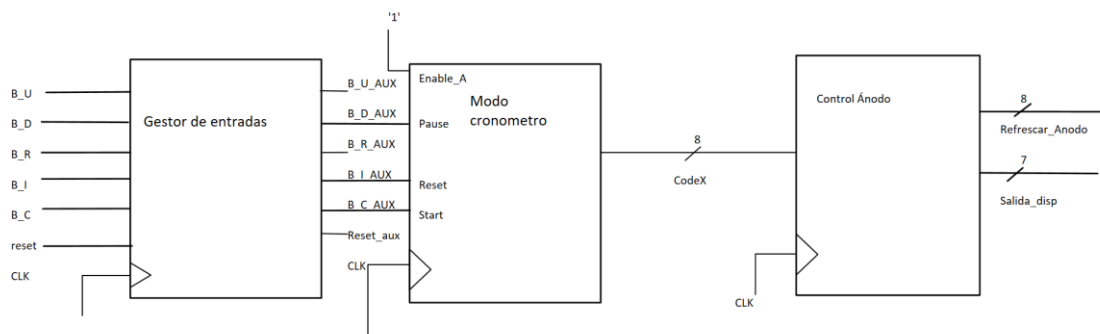
Los displays nº 1,2,3,4 se corresponderán con:

1	Unidades de Segundo
2	Decenas de segundo
3	Unidades de Minuto
4	Decenas de Minuto

Botonera:

B_U	-
B_D	Pausa
B_R	-
B_I	Reset
B_C	Start

3. DIAGRAMA GENERAL



4. ESTRATEGIAS Y ALGORITMOS

En primer lugar, para que la organización del proyecto sea óptima y la forma de trabajar sea fructuosa, se repartirán roles a la hora de organizar el equipo para el proyecto.

Aunque todos hayamos participado en el análisis y diseño del proyecto, los roles son los siguientes:

Arquitecto del sistema: Raúl Herranz

Programador: Carlos Murillo

Responsable de calidad: Alejandro Sacristán.

Una vez repartidos los roles y organizado el equipo, se pensará claramente el proyecto que se quiere hacer y el objetivo que se quiere alcanzar. Una vez aclarado esto, se utilizará una metodología Top-Down, consistente en dividir todo el proyecto en bloques o módulos hasta llegar a las partes más sencillas. Esta metodología se ha elegido así para reutilizar más partes y para que cada integrante pueda trabajar en paralelo, además de que facilita el diseño de testbenchs y el avance en el proyecto.

En primer lugar, el modulo Top, que contendrá todo el trabajo en el mismo con todo perfectamente sincronizado y agrupado.

En este top tendremos otros componentes que ejecutarán tareas más simples.

El primero será el gestor de entradas, que a través de un detector de flancos y un sincronizador por cada entrada, en nuestro caso 6, nos dará como resultado una señal con

mayor inmunidad al ruido que la original, ya que las señales generadas por botones presentan una alta cantidad, pudiendo generar rebotes. Se ha realizado para 6 botones con el objetivo de gestionar la botonera de la placa y también el botón de reset de la misma. Aunque solamente para este proyecto solo necesitamos 3 entradas (Start, Pausa, Reset) se ha realizado con el objetivo de extender las funcionalidades del diseño.

Segundo, el modo cronometro, en su interior contará con un preescaler que convierta la señal generada por el reloj de 100MHz de la placa a una frecuencia de 1Hz, para poder realizar la temporización correctamente. También contará con una pequeña máquina de estados que gestionará la marcha, paro y reinicio de la cuenta. Por salida tendrá 8 vectores de 4 bits en código BCD+codificación propia para los números 0xA, 0xB, 0xC, 0xD, 0xE y 0xF,

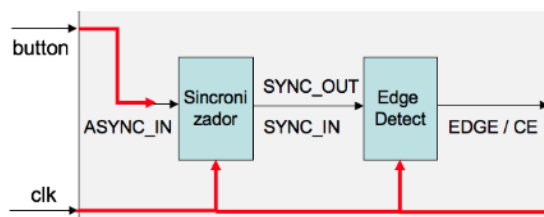
Tercero, Control ánodo, debido a que en la FPGA utilizada los segmentos de los 8 displays comparten pines, debemos controlar los ánodos de los displays, es decir debemos crear una entidad que sincronice el valor de los pines de los segmentos en función del ánodo activo, esto se realiza a una frecuencia de 10KHz, frecuencia que hemos obtenido de introducir un preescaler. A esta velocidad el parpadeo no es visible para el ojo humano y da la sensación de que todos los displays están encendidos simultáneamente. Esta entidad será la que reciba los vectores de 4 bits, por lo que contará con un decodificador que de BCD+codificación propia a los 7 segmentos



5. DESCRIPCIÓN VHDL DE LOS BLOQUES FUNCIONALES.

5.1. Gestor de entradas

Se tratará de una entidad que contendrá la siguiente estructura repetida 6 veces, una por cada entrada a gestionar.



Cuya descripción VHDL es:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity GestorEntradas is
    Port (
        CLK : in std_logic;
        B_L : in std_logic;
        B_R : in std_logic;
        B_U : in std_logic;
        B_D : in std_logic;
        B_C : in std_logic;
```

```

        reset : in std_logic;
        B_L_out : out std_logic;
        B_R_out : out std_logic;
        B_U_out : out std_logic;
        B_D_out : out std_logic;
        B_C_out : out std_logic;
        reset_out : out std_logic
    );
end GestorEntradas;

architecture Behavioral of GestorEntradas is

    signal sync_auxL: std_logic;
    signal sync_auxR: std_logic;
    signal sync_auxU: std_logic;
    signal sync_auxD: std_logic;
    signal sync_auxC: std_logic;
    signal sync_auxReset: std_logic;

    COMPONENT SYNCHRNZR
    PORT (
        async_in : IN std_logic;
        clk: IN std_logic;
        sync_out : OUT std_logic
    );
    END COMPONENT;

    COMPONENT EDGEDTCTR
    PORT (
        sync_in : IN std_logic;
        clk: IN std_logic;
        edge : OUT std_logic
    );
    END COMPONENT;

begin
    Sincronizador: SYNCHRNZR PORT MAP(
        ASYNC_IN=>B_L,
        CLK=>clk,
        SYNC_OUT=>sync_auxL
    );

    DetectorFlanco: EDGEDTCTR PORT MAP(
        clk=>clk,
        SYNC_IN=>sync_auxL,
        EDGE=>B_L_out
    );

    Sincronizador2: SYNCHRNZR PORT MAP(
        ASYNC_IN=>B_R,
        CLK=>clk,
        SYNC_OUT=>sync_auxR
    );

    DetectorFlanco2: EDGEDTCTR PORT MAP(
        clk=>clk,
        SYNC_IN=>sync_auxR,
        EDGE=>B_R_out
    );

    Sincronizador3: SYNCHRNZR PORT MAP(
        ASYNC_IN=>B_U,
        CLK=>clk,
        SYNC_OUT=>sync_auxU
    );

    DetectorFlanco3: EDGEDTCTR PORT MAP(
        clk=>clk,
        SYNC_IN=>sync_auxU,
        EDGE=>B_U_out
    );

    Sincronizador4: SYNCHRNZR PORT MAP(
        ASYNC_IN=>B_D,
        CLK=>clk,

```

```

        SYNC_OUT=>sync_auxD
    );

    DetectorFlanco4: EDGEDTCTR PORT MAP(
        clk=>clk,
        SYNC_IN=>sync_auxD,
        EDGE=>B_D_out
    );

    Sincronizador5: SYNCHRNZR PORT MAP(
        ASYNC_IN=>B_C,
        CLK=>clk,
        SYNC_OUT=>sync_auxC
    );

    DetectorFlanco5: EDGEDTCTR PORT MAP(
        clk=>clk,
        SYNC_IN=>sync_auxC,
        EDGE=>B_C_out
    );

    Sincronizador6: SYNCHRNZR PORT MAP(
        ASYNC_IN=>reset,
        CLK=>clk,
        SYNC_OUT=>sync_auxReset
    );

    DetectorFlanco6: EDGEDTCTR PORT MAP(
        clk=>clk,
        SYNC_IN=>sync_auxReset,
        EDGE=>reset_out
    );

end Behavioral;

```

5.2. Modo cronómetro.

5.2.1. Preescaler

Antes de entrar en materia con las funcionalidades más relevantes introducidas en el Modo cronómetro, es necesario introducir el componente “clk1hz” el cual es un preescalar encargado de ofrecernos una señal periódica de 1Hz teniendo como entrada el reloj de la placa, esta operación se realiza con un contador relativamente sencillo.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clk1Hz is
    Port (
        CLK: in  STD_LOGIC;
        CLK_1hz : out STD_LOGIC
    );
end clk1Hz;

architecture Behavioral of clk1Hz is
    signal temporal: STD_LOGIC;
    signal contador: integer range 0 to 49999999 := 0;
begin
    divisor_frecuencia: process (CLK) begin
        if rising_edge(CLK) then
            if (contador = 49999999) then
                temporal <= NOT(temporal);
                contador <= 0;
            else
                contador <= contador+1;
            end if;
        end if;
    end process;

    CLK_1hz <= temporal;

end Behavioral;

```

5.2.2. Entidad Modo Cronómetro

Para hacer una mejor descripción de las distintas funcionalidades introducidas, las iremos dividiendo por partes, comenzando por las librerías utilizadas y la declaración de la entidad:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Modo_Crono is
  generic(
    width:positive:=3
  );
  Port (
    CLK : in std_logic;--entrada del reloj de la placa
    code1 : out std_logic_vector(width downto 0);--Salidas en BCD o
codificación propia
    code2 : out std_logic_vector(width downto 0);
    code3 : out std_logic_vector(width downto 0);
    code4 : out std_logic_vector(width downto 0);
    code5 : out std_logic_vector(width downto 0);
    code6 : out std_logic_vector(width downto 0);
    code7 : out std_logic_vector(width downto 0);
    code8 : out std_logic_vector(width downto 0);
    Enable_A : in std_logic;--Habilitación del componente
    Start : in std_logic;
    Pause : in std_logic;
    Reset : in std_logic
  );
end Modo_Crono;
```

Declaraciones de la Arquitectura:

```
architecture Behavioral of Modo_Crono is

  signal Start_s : std_logic := '0';
  signal Reset_s : std_logic := '0';

  signal clk_1hz : std_logic;

  COMPONENT clk1hz
  PORT (
    CLK: in STD_LOGIC;
    clk_1hz : out STD_LOGIC
  );
  END COMPONENT;
begin
  Inst_clk1hz: clk1hz
  PORT MAP (
    CLK => CLK,
    CLK_1hz => clk_1hz
  );
end;
```

Las señales Start_s y Reset_s serán los estados que habiliten la cuenta o su reinicio, mientras que la señal clk_1hz se usa como señal para contar segundo a segundo. Se debe destacar también la instanciación del componente clk1hz (prescaler descrito anteriormente)

Máquina de estados:

```
maquinaestados : process (Enable_A,Start,Pause,Reset)
begin
  if Enable_A = '1' and Start = '1' and Pause = '0'then
    Start_s<='1';
    Reset_s<='0';
  elsif Pause='1' then
    Start_s<='0';
    Reset_s<='0';
  elsif Reset = '1'then
```



```

        Reset_s<='1';
    end if;
end process;

```

Se observa que se trata de un máquina de estados asíncrona, en la que se puede apreciar como condición necesaria para iniciar la cuenta la entrada de habilitación Enable_A y la entrada Start(botón) a la misma vez que no se esté pulsando el Pausa, a continuación se gestiona cuando se pausa y cuando se reinicia.

Process principal de temporización:

```

process (clk_1hz, Start_s, Reset_s)
    subtype V is integer range 0 to 15;
    variable unit_sec : V :=0;
    variable unit_min : V :=0;
    variable dec_sec : V :=0;
    variable dec_min : V :=0;
begin

    if Reset_s='1' then
        unit_sec:=0;
        dec_sec:=0;
        unit_min:=0;
        dec_min:=0;
    elsif rising_edge(clk_1hz) and Start_s='1' then
        unit_sec:=unit_sec+1;
        if unit_sec=10 then
            unit_sec:=0;
            dec_sec:=dec_sec+1;
            if dec_sec=6 then
                dec_sec:=0;
                unit_min:=unit_min+1;
                if unit_min=10 then
                    unit_min:=0;
                    dec_min:=dec_min+1;
                    if dec_min=9 then
                        dec_min:=0;
                        unit_min:=0;
                        dec_sec:=0;
                        unit_sec:=0;
                    end if;
                end if;
            end if;
        end if;
    end if;

    end if;

end if;

code1 <= std_logic_vector(to_unsigned(unit_sec,code8'length));
code2 <= std_logic_vector(to_unsigned(dec_sec,code7'length));
code3 <= std_logic_vector(to_unsigned(unit_min,code6'length));
code4 <= std_logic_vector(to_unsigned(dec_min,code5'length));
end process;

```

Para comenzar se ha definido un tipo de variable entero con rango de 0 a 15, que es el rango de nuestros vectores de salida de 4 bits, esto lo hemos realizado para que el código sea más sencillo de ejecutar y de leer. De este tipo se crean 4 variables, correspondientes a las unidades y decenas de minutos y segundos.

Teniendo la señal Reset_s prioritaria, posteriormente, en cada flanco de reloj se sumará un segundo, y se gestionarán los límites para que entre dentro de la normalidad (que no se muestre más de 60 segundos y en su lugar sume 1 minuto por ejemplo).

Finalmente convertimos las variables utilizadas en vectores de 4 bits de salida, que se mostrarán posteriormente en los displays 1,2,3 y 4.

Por último, se muestra el process en el que de forma permanente, mientras Enable_A esté activado, hace que se muestre en la salida “Cr--”, se corresponden con la parte de codificación propia utilizada.

```

















Marca_cron : process (Enable_A)
begin
  if Enable_A='1' then
    code8<="1010";
    code7<="1011";
    code6<="1111";
    code5<="1111";
  end if;
end process;
end Behavioral;

```

5.3. Control Ánodo.

5.3.1. Decodificador

Inicialmente, antes de realizar la sincronización de la salida debemos convertir nuestros vectores de 4 bits codificados, en vectores de 7 bits, en los que cada bits se corresponderá con cada uno de los 7 segmentos de los displays. Para realizar esta tarea se ha diseñado un decodificador que seguiría la siguiente tabla:

Entrada	Salida	Hexadecimal	Display
0000	0000001	0	
0001	1001111	1	
0010	0010010	2	
0011	0000110	3	
0100	1001100	4	
0101	0100100	5	
0110	0100000	6	
0111	0001111	7	
1000	0000000	8	
1001	0000100	9	
1010	0110001	A	
1011	1111010	B	
1100	1110000	C	
1101	0110000	D	
1110	0001000	E	
1111	1111110	F	

5.3.2. Entidad Control Ánodo.

Nuevamente para explicar paso a paso esta entidad iremos separándola conceptualmente.

Para comenzar se muestran las librerías utilizadas y la declaración de la entidad.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

```

```

entity Control_Anodo is
  generic(
    width:positive:=3
  );
  port(
    CLK : in std_logic;
    code1 : in std_logic_vector(width downto 0);
    code2 : in std_logic_vector(width downto 0);
    code3 : in std_logic_vector(width downto 0);
    code4 : in std_logic_vector(width downto 0);
    code5 : in std_logic_vector(width downto 0);
    code6 : in std_logic_vector(width downto 0);
    code7 : in std_logic_vector(width downto 0);
    code8 : in std_logic_vector(width downto 0);
    refrescar_anodo : out std_logic_vector(7 downto 0); --vector que
pone a 1 el ánodo correspondiente para actualizar
    salida_disp : out std_logic_vector(6 downto 0) --salida de los
displays
  );
end Control_Anodo;

```

Como ya se ha comentado anteriormente esta entidad contará con los 8 vectores de 4 bits como entradas, y que se corresponderán con lo que debe mostrar cada display.

Como salidas tendremos refrescar_anodo, que pondrá a nivel alto o bajo los ánodos de cada display, y salida_disp, que enviará el código que corresponda a los displays.

A continuación, se muestran las instanciaciones de los decodificadores y de un preescaler a 10kHz análogo al de 1Hz descrito anteriormente,

```

architecture Behavioral of Control_Anodo is

  signal disp1 : std_logic_vector(6 downto 0);
  signal disp2 : std_logic_vector(6 downto 0);
  signal disp3 : std_logic_vector(6 downto 0);
  signal disp4 : std_logic_vector(6 downto 0);
  signal disp5 : std_logic_vector(6 downto 0);
  signal disp6 : std_logic_vector(6 downto 0);
  signal disp7 : std_logic_vector(6 downto 0);
  signal disp8 : std_logic_vector(6 downto 0);

  signal flag : integer := 1;

  signal clk_10khz : std_logic;

  COMPONENT clk10khz
    PORT (
      CLK: in  STD_LOGIC;
      clk_1hz : out STD_LOGIC
    );
  END COMPONENT;

  COMPONENT deco1
    PORT (
      code : IN std_logic_vector(3 DOWNT0 0);
      led : OUT std_logic_vector(6 DOWNT0 0)
    );
  END COMPONENT;

begin

  Inst_clk10khz: clk10khz
    PORT MAP (
      CLK => CLK,
      CLK_1hz => clk_10khz
    );

```

```

Inst_deco1: deco1 PORT MAP (
    code => code1,
    led => disp1
);
Inst_deco2: deco1 PORT MAP (
    code => code2,
    led => disp2
);
Inst_deco3: deco1 PORT MAP (
    code => code3,
    led => disp3
);
Inst_deco4: deco1 PORT MAP (
    code => code4,
    led => disp4
);
Inst_deco5: deco1 PORT MAP (
    code => code5,
    led => disp5
);
Inst_deco6: deco1 PORT MAP (
    code => code6,
    led => disp6
);
Inst_deco7: deco1 PORT MAP (
    code => code7,
    led => disp7
);
Inst_deco8: deco1 PORT MAP (
    code => code8,
    led => disp8
);

```

Se introduce una señal “flag” de tipo integer para realizar la cuenta correspondiente y así poder actualizar los displays en orden a una frecuencia de 10Khz como se muestra en el siguiente process:

```

process (clk_10khz)
begin
    if rising_edge(clk_10khz) then
        if flag=1 then
            refrescar_anodo(0) <= '0';
            refrescar_anodo(7 downto 1) <= "1111111";
            salida_disp <= disp1;
            flag<=2;
        end if;
        if flag=2 then
            refrescar_anodo(1) <= '0';
            refrescar_anodo(0) <= '1';
            refrescar_anodo(7 downto 2) <= "111111";
            salida_disp <= disp2;
            flag<=3;
        end if;
        if flag=3 then
            refrescar_anodo(2) <= '0';
            refrescar_anodo(1 downto 0) <= "11";
            refrescar_anodo(7 downto 3) <= "11111";
            salida_disp <= disp3;
            flag<=4;
        end if;
        if flag=4 then
            refrescar_anodo(3) <= '0';
            refrescar_anodo(2 downto 0) <= "111";
            refrescar_anodo(7 downto 4) <= "1111";
            salida_disp <= disp4;
            flag<=5;
        end if;
        if flag=5 then
            refrescar_anodo(4) <= '0';
            refrescar_anodo(3 downto 0) <= "1111";
            refrescar_anodo(7 downto 5) <= "111";
            salida_disp <= disp5;
            flag<=6;
        end if;
    end if;
end process;

```

```

        if flag=6 then
            refrescar_anodo(5) <= '0';
            refrescar_anodo(4 downto 0) <= "11111";
            refrescar_anodo(7 downto 6) <= "11";
            salida_disp <= disp6;
            flag<=7;
        end if;
        if flag=7 then
            refrescar_anodo(6) <= '0';
            refrescar_anodo(5 downto 0) <= "111111";
            refrescar_anodo(7) <= '1';
            salida_disp <= disp7;
            flag<=8;
        end if;
        if flag=8 then
            refrescar_anodo(7) <= '0';
            refrescar_anodo(6 downto 0) <= "1111111";
            salida_disp <= disp8;
            flag<=1;
        end if;
    end if;
end process;
end Behavioral;

```

5.4. Entidad Top

En la Entidad Top se recogerán todas las entidades anteriores y se realizarán las respectivas conexiones entre ellas a través de diferentes señales señales.

Destacamos que la entrada “Enable_A” se fuerza directamente a 1 en todo momento.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TopDefinitivo is
    Port (
        CLK : in std_logic;
        B_L : in std_logic;
        B_R : in std_logic;
        B_U : in std_logic;
        B_D : in std_logic;
        B_C : in std_logic;
        reset : in std_logic;
        refrescar_anodo : out std_logic_vector(7 downto 0); --vector que pone a 1
        el ánodo correspondiente para actualizar
        salida_disp : out std_logic_vector(6 downto 0); --salida de los displays
        led : out std_logic
    );
end TopDefinitivo;

architecture Behavioral of TopDefinitivo is

    signal sync_auxL: std_logic;
    signal sync_auxR: std_logic;
    signal sync_auxU: std_logic;
    signal sync_auxD: std_logic;
    signal sync_auxC: std_logic;

    signal B_L_aux: std_logic;
    signal B_R_aux: std_logic;
    signal B_U_aux: std_logic;
    signal B_D_aux: std_logic;
    signal B_C_aux: std_logic;
    signal Reset_aux: std_logic;

    signal code1_A : std_logic_vector(3 downto 0);
    signal code2_A : std_logic_vector(3 downto 0);

```

```

signal code3_A : std_logic_vector(3 downto 0);
signal code4_A : std_logic_vector(3 downto 0);
signal code5_A : std_logic_vector(3 downto 0);
signal code6_A : std_logic_vector(3 downto 0);
signal code7_A : std_logic_vector(3 downto 0);
signal code8_A : std_logic_vector(3 downto 0);

signal code1_aux : std_logic_vector(3 downto 0);
signal code2_aux : std_logic_vector(3 downto 0);
signal code3_aux : std_logic_vector(3 downto 0);
signal code4_aux : std_logic_vector(3 downto 0);
signal code5_aux : std_logic_vector(3 downto 0);
signal code6_aux : std_logic_vector(3 downto 0);
signal code7_aux : std_logic_vector(3 downto 0);
signal code8_aux : std_logic_vector(3 downto 0);

COMPONENT GestorEntradas
  PORT (
    CLK : in std_logic;
    B_L : in std_logic;
    B_R : in std_logic;
    B_U : in std_logic;
    B_D : in std_logic;
    B_C : in std_logic;
    reset : in std_logic;
    B_L_out : out std_logic;
    B_R_out : out std_logic;
    B_U_out : out std_logic;
    B_D_out : out std_logic;
    B_C_out : out std_logic;
    reset_out : out std_logic
  );
END COMPONENT;

COMPONENT Modo_Crono
  PORT (
    CLK : in std_logic;
    code1 : out std_logic_vector(3 downto 0);
    code2 : out std_logic_vector(3 downto 0);
    code3 : out std_logic_vector(3 downto 0);
    code4 : out std_logic_vector(3 downto 0);
    code5 : out std_logic_vector(3 downto 0);
    code6 : out std_logic_vector(3 downto 0);
    code7 : out std_logic_vector(3 downto 0);
    code8 : out std_logic_vector(3 downto 0);
    Enable_A : in std_logic;
    Start : in std_logic;
    Pause : in std_logic;
    Reset : in std_logic
  );
END COMPONENT;

COMPONENT Control_anodo
  PORT (
    CLK : in std_logic;
    code1 : in std_logic_vector(3 downto 0);
    code2 : in std_logic_vector(3 downto 0);
    code3 : in std_logic_vector(3 downto 0);
    code4 : in std_logic_vector(3 downto 0);
    code5 : in std_logic_vector(3 downto 0);
    code6 : in std_logic_vector(3 downto 0);
    code7 : in std_logic_vector(3 downto 0);
    code8 : in std_logic_vector(3 downto 0);
    refrescar_anodo : out std_logic_vector(7 downto 0);
    salida_disp : out std_logic_vector(6 downto 0)
  );
END COMPONENT;

```

```

begin

    GestorEntradas1 : GestorEntradas PORT MAP (
        CLK => CLK,
        B_L => B_L,
        B_R => B_R,
        B_U => B_U,
        B_D => B_D,
        B_C => B_C,
        reset => reset,
        B_L_out => B_L_aux,
        B_R_out => B_R_aux,
        B_U_out => B_U_aux,
        B_D_out => B_D_aux,
        B_C_out => B_C_aux,
        reset_out => Reset_aux
    );

    Modo_Crono1 : Modo_Crono PORT MAP (
        CLK=>clk,
        code1=>code1_aux,
        code2=>code2_aux,
        code3=>code3_aux,
        code4=>code4_aux,
        code5=>code5_aux,
        code6=>code6_aux,
        code7=>code7_aux,
        code8=>code8_aux,
        Enable_A=>'1',
        Start=>B_C_aux,
        Pause=>B_D_aux,
        Reset=>B_L_aux
    );

    Control_Anodo1 : Control_anodo PORT MAP (
        CLK=>clk,
        code1=>code1_aux,
        code2=>code2_aux,
        code3=>code3_aux,
        code4=>code4_aux,
        code5=>code5_aux,
        code6=>code6_aux,
        code7=>code7_aux,
        code8=>code8_aux,
        refrescar_anodo=>refrescar_anodo,
        salida_disp=>salida_disp
    );

end Behavioral;

```

6. SIMULACIÓN

Se ha realizado a través de TestBenchs la simulación de las entidades Modo Cronómetro y control de ánodo.

Todos ellos se han realizado con la ayuda de <https://vhdl.lapinoo.net/testbench/>

6.1. Simulación Modo Cronómetro.

Debido a que se está empleando un preescaler de 100Mhz a 1Hz, no es práctico en el testbench simular durante segundos, por lo que cambiamos momentáneamente el reloj de 1Hz en la entidad Modo cronometro por el Reloj CLK de entrada a la entidad, para que el testbench sea más intuitivo, y aunque

no es 100% realista, es lo suficientemente próximo a la realidad como para sacar conclusiones de los resultados.

El testbench empleado será:

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

entity Modo_Crono_tb is
end Modo_Crono_tb;

architecture tb of Modo_Crono_tb is

    component Modo_Crono
        port (CLK          : in std_logic;
              code1        : out std_logic_vector (3 downto 0);
              code2        : out std_logic_vector (3 downto 0);
              code3        : out std_logic_vector (3 downto 0);
              code4        : out std_logic_vector (3 downto 0);
              code5        : out std_logic_vector (3 downto 0);
              code6        : out std_logic_vector (3 downto 0);
              code7        : out std_logic_vector (3 downto 0);
              code8        : out std_logic_vector (3 downto 0);
              Enable_A     : in std_logic;
              Start        : in std_logic;
              Pause        : in std_logic;
              Reset        : in std_logic);
    end component;

    signal CLK          : std_logic;
    signal code1        : std_logic_vector (3 downto 0);
    signal code2        : std_logic_vector (3 downto 0);
    signal code3        : std_logic_vector (3 downto 0);
    signal code4        : std_logic_vector (3 downto 0);
    signal code5        : std_logic_vector (3 downto 0);
    signal code6        : std_logic_vector (3 downto 0);
    signal code7        : std_logic_vector (3 downto 0);
    signal code8        : std_logic_vector (3 downto 0);
    signal Enable_A     : std_logic;
    signal Start        : std_logic;
    signal Pause        : std_logic;
    signal Reset        : std_logic;

    constant TbPeriod : time := 100 ns; -- EDIT Put right period here
    signal TbClock     : std_logic := '0';
    signal TbSimEnded  : std_logic := '0';

begin

    dut : Modo_Crono
    port map (CLK          => CLK,
              code1        => code1,
              code2        => code2,
              code3        => code3,
              code4        => code4,
              code5        => code5,
              code6        => code6,
              code7        => code7,
              code8        => code8,
              Enable_A     => Enable_A,
              Start        => Start,
              Pause        => Pause,
              Reset        => Reset);

    -- 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
        Enable_A <= '0';
        Start <= '0';
        Pause <= '0';
    end process;

end tb;
```



```

-- 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
enable_a<='1';
wait for 100 ns;
start<='1';
wait for 100 ns;
start <= '0';
wait for 500 ns;
pause<='1';
wait for 100 ns;
pause <= '0';
wait for 200 ns;
reset<='1';
wait for 100 ns;
reset <= '0';
wait for 200 ns;
start<='1';
wait for 100 ns;
start <= '0';

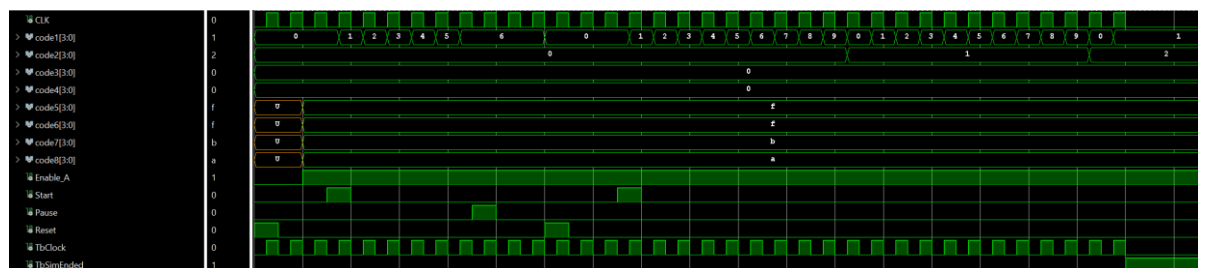
wait for 1000 ns;

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

end tb;

```

Obtenemos los siguientes resultados en la simulación.



Se observa que code(8,7,6,5) tienen el valor de abff en todo momento desde la activación de Enable_A, lo que es equivalente a Cr--, por otra parte se observa como cuando se provoca un pulso en la entrada Start comienza a sumar 1 segundo en cada pulso de reloj, esto se detiene cuando se genera el pulso el Pause, y posteriormente se reinicia con el pulso Reset, para volver a iniciar la cuenta cuando Start vuelve a estar a nivel alto.

6.2. Simulación Control Ánodo.

Testbench empleado:

```

library ieee;
use ieee.std_logic_1164.all;

entity Control_Anodo_tb is
end Control_Anodo_tb;

architecture tb of Control_Anodo_tb is

    component Control_Anodo
        port (CLK
              : in std_logic;
              code1
              : in std_logic_vector (3 downto 0);
              code2
              : in std_logic_vector (3 downto 0);

```

```

        code3      : in std_logic_vector (3 downto 0);
        code4      : in std_logic_vector (3 downto 0);
        code5      : in std_logic_vector (3 downto 0);
        code6      : in std_logic_vector (3 downto 0);
        code7      : in std_logic_vector (3 downto 0);
        code8      : in std_logic_vector (3 downto 0);
        refrescar_anodo : out std_logic_vector (7 downto 0);
        salida_disp  : out std_logic_vector (6 downto 0));
end component;

signal CLK          : std_logic;
signal code1        : std_logic_vector (3 downto 0);
signal code2        : std_logic_vector (3 downto 0);
signal code3        : std_logic_vector (3 downto 0);
signal code4        : std_logic_vector (3 downto 0);
signal code5        : std_logic_vector (3 downto 0);
signal code6        : std_logic_vector (3 downto 0);
signal code7        : std_logic_vector (3 downto 0);
signal code8        : std_logic_vector (3 downto 0);
signal refrescar_anodo : std_logic_vector (7 downto 0);
signal salida_disp   : std_logic_vector (6 downto 0);

constant TbPeriod : time := 100 ns; -- EDIT Put right period here
signal TbClock : std_logic := '0';
signal TbSimEnded : std_logic := '0';

begin

    dut : Control_Anodo
    port map (CLK          => CLK,
              code1        => code1,
              code2        => code2,
              code3        => code3,
              code4        => code4,
              code5        => code5,
              code6        => code6,
              code7        => code7,
              code8        => code8,
              refrescar_anodo => refrescar_anodo,
              salida_disp   => salida_disp);

    -- 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;

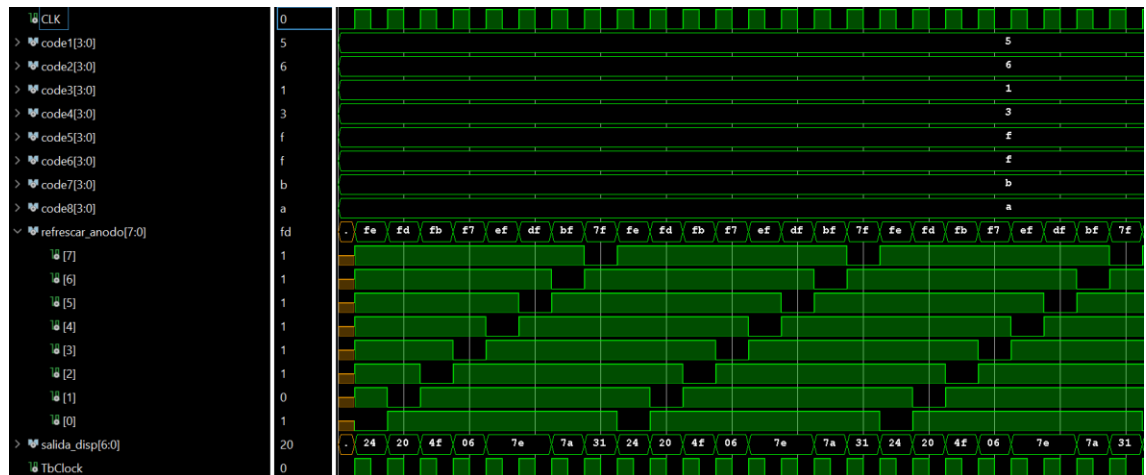
    stimuli : process
    begin
        -- EDIT Adapt initialization as needed
        code1 <= "0101";
        code2 <= "0110";
        code3 <= "0001";
        code4 <= "0011";
        code5 <= "1111";
        code6 <= "1111";
        code7 <= "1011";
        code8 <= "1010";

        -- Reset generation
        -- EDIT: Replace YOURRESETSIGNAL below by the name of your reset as I
        haven't guessed it
        --YOURRESETSIGNAL <= '1';
        --wait for 100 ns;
        --YOURRESETSIGNAL <= '0';
        wait for 8000 ns;

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

```

Resultados Obtenidos



Como se observa, los ánodos de cada display se encienden y apagan de forma ordenada e individualmente, a la vez que se establece el valor adecuado de los 7 segmentos en cada instante.

7. ANEXO. CRONÓMETRO DE TRES FUNCIONES

7.1. Introducción

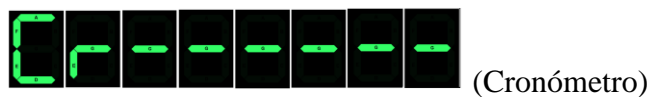
El objetivo será realizar la descripción en VHDL para la implementación de un cronómetro con tres funciones en una FPGA, modelo Nexys DDR4.

Las tres funciones de las que dispondremos serán: Un cronómetro, que cuente el tiempo transcurrido tras pulsar un botón con la posibilidad de pausarlo y reiniciarlo, un temporizador, (que cuente hacia atrás) que nos permita seleccionar el tiempo que queramos que cuente, pausarlo, y reiniciarlo, y por último un modo ajedrez, que pulsando dos botones activen y desactiven dos temporizadores de 10 minutos cada uno.

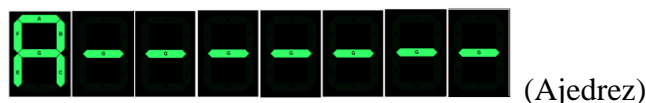
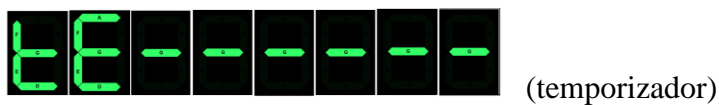
7.2. Instrucciones.

Seleccionando el modo

Inicialmente tendremos:



Si pulsamos B_U y B_D observamos que podremos cambiar a las siguientes opciones también:



Pulsando B5 entrará en funcionamiento el modo elegido.

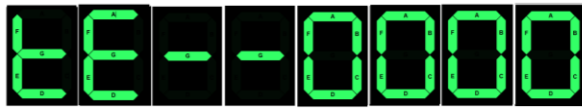
Para volver a la selección de modo cuando esté activo alguno de estos pulsaremos el Reset de la placa.

Modo Cronometro.

(Descrito anteriormente)

Modo Temporizador.

Inicialmente se mostrará lo siguiente:



Con el botón B1 modificamos el valor de los segundos, con B2 el de las decenas de segundos, B3, unidades de minuto, B4 decenas de minuto.

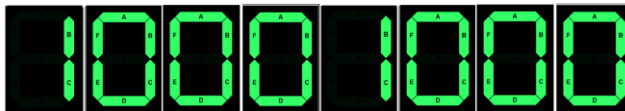
Para comenzar la cuenta atrás pulsamos B5.

Para Pausarlo pulsaremos B2, y para reiniciarlo B4.

Para volver a la selección de cuenta se deberá volver a la selección de modo y volver a elegir temporizador.

Modo Ajedrez

Inicialmente se mostrará:



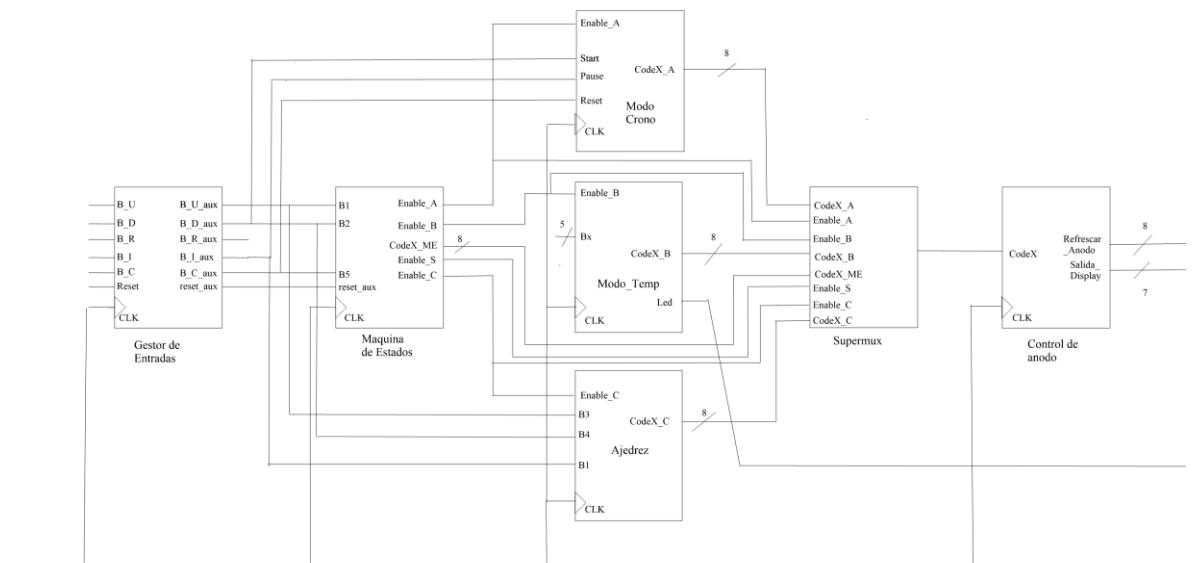
Llamaremos al reloj izquierdo Jugador A y al derecho Jugador B.

Para comenzar el juego, se pulsará o B4 o B3, si pulsamos B4 se iniciará el reloj de B, y si pulsamos B3 se iniciará el de A.

En caso de que se haya iniciado B, para pausarlo e iniciar A se pulsará B3. Repitiendo el ciclo hasta que uno de los jugadores se quede sin tiempo

Para reiniciar el juego se pulsará B1.

7.3. Diagrama General



7.4. Estrategia y algoritmos.

Nuevamente, elegiremos el método de Diseño Top-Down, permitiéndonos desglosar el objetivo final en tareas sencillas y simulables cada una de ellas.

- Gestor de Entradas, nuevamente será necesario añadir una sincronización de las entradas, la cual completamente idéntica a la descrita anteriormente. Compartiendo los mismos objetivos de reducir los rebotes producidos por los botones utilizados como entradas del diseño.
- Maquina de Estados, Gestionará el modo de funcionamiento a través de señales de habilitación (Enable_X), también incluirá 8 salidas que serán vectores de 4 bits codificados exactamente igual que las señales anteriormente descritas. Estas salidas servirán para que el usuario sepa a través de los displays cual es el modo que está eligiendo.
- Modo Cronometro, debido a que utilizamos la entidad ya descrita no profundizaremos en ella.
- Modo Temporizador, de forma interna dispondrá de una máquina de estados propia que gestione los estados de Seleccionar cuenta, cuenta, pausa, y reinicio. “Sel_Cuenta” será una entidad Instanciada dentro de esta, así como “Cuenta_atras” que será la entidad encargada de, introduciendo un tiempo, contará hacia atrás a través de una señal de habilitación.
- Modo Ajedrez, Consistirá en mostrar 2 relojes en los displays que contarán hacia 0 de manera alterna según vayan los jugadores pulsando sus

respectivos botones. Esta entidad aprovecha la entidad “Cuenta_atras” descrita en el apartado anterior.

- f) SuperMux, como Modo_crono, Modo_temp, Ajedrez, y la maquina de estados tienen como salidas vectores de 4 bits con destino a la salida de los displays, será necesario multiplexarlas, dependiendo de la función del programa que nos interese y no se superpongan unas a las otras, lo cual genera problemas, las señales que eligen que entradas gobernarán la salida serán las mismas señales de habilitación, las cuales son gobernadas por la máquina de estados principal.
- g) Control de ánodo, entidad exactamente igual que la anteriormente descrita.

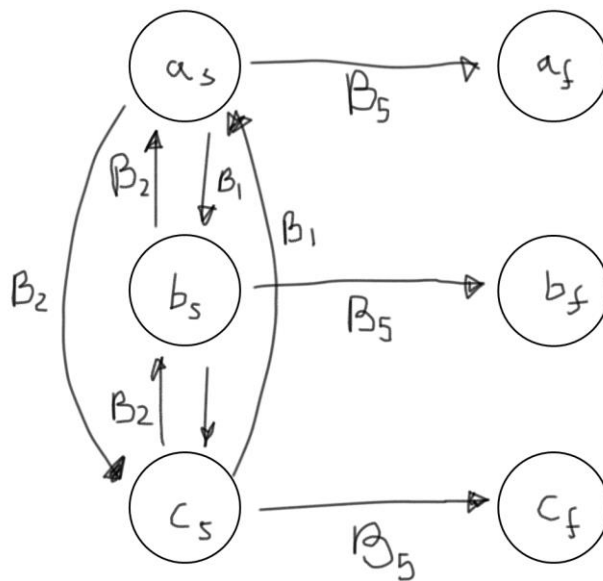


7.5. Descripción VHDL de los bloques funcionales.

No se describirán ni el gestor de entradas, ni modo cronometro ni control de ánodo, debido a que son exactamente iguales a las anteriormente descritas.

Máquina de Estados.

Se trata de una máquina de estados con actualización del estado Síncrona, en la que tenemos los estados (as,bs,cs,af,bf,cf), la dinámica de la máquina se entiende con mayor claridad en el siguiente diagrama.



Se trata de una máquina de Moore, ya que las salidas dependen únicamente del estado en el que se encuentre el sistema.

El reset hace que desde cualquier estado se vuelva al estado “as”.

Los nombres de los estados no son escogidos al azar, la primera letra indica el modo (a=crono, b=temp, c=ajedrez) mientras que la segunda indica si está en selección (s) o en funcionamiento (f).

Para describir esta funcionalidad se ha realizado la siguiente descripción:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity MaquinaEstados is
  Port (
    clk : in std_logic;
    B1 : in std_logic;
    B2 : in std_logic;
    B3 : in std_logic;
    B4 : in std_logic;
    B5 : in std_logic;
    code1 : out std_logic_vector(3 downto 0);
    code2 : out std_logic_vector(3 downto 0);
    code3 : out std_logic_vector(3 downto 0);
    code4 : out std_logic_vector(3 downto 0);
    code5 : out std_logic_vector(3 downto 0);
    code6 : out std_logic_vector(3 downto 0);
    code7 : out std_logic_vector(3 downto 0);
    code8 : out std_logic_vector(3 downto 0);
    Reset : in std_logic;
    Enable_A : out std_logic := '0';
    Enable_B : out std_logic := '0';
    Enable_C : out std_logic := '0';
    Enable_S : out std_logic := '1'
  );
end MaquinaEstados;

architecture Behavioral of MaquinaEstados is
  type estados is (as,bs,cs,af,bf,cf);
  signal estadoActual,estadoSiguiente:estados;

  signal clk_10khz : std_logic;

```

```

COMPONENT clk10khz
  PORT (
    CLK: in  STD_LOGIC;
    clk_1hz : out STD_LOGIC
  );
END COMPONENT;

begin
Inst_clk10khz: clk10khz
  PORT MAP (
    CLK => CLK,
    CLK_1hz => clk_10khz
  );
  state_reg:process(clk,reset)
  begin
    if reset='1' then
      estadoActual<=as;
    elsif rising_edge(clk) then
      estadoActual<=estadoSiguiente;
    end if;
  end process;

  gestionmaquinaestados:process(estadoActual,B1,B2,B5)
  begin
    estadoSiguiente<=estadoActual;
    case (estadoActual)is
      when as=>
        if B1='1' then
          estadoSiguiente<=bs;
        elsif B2='1' then
          estadosiguiente<=cs;
        elsif B5='1' then
          estadoSiguiente<=af;
        end if;
      when bs=>
        if B1='1' then
          estadoSiguiente<=cs;
        elsif B2='1' then
          estadosiguiente<=as;
        elsif B5='1' then
          estadoSiguiente<=bf;
        end if;
      when cs=>
        if B1='1' then
          estadoSiguiente<=as;
        elsif B2='1' then
          estadosiguiente<=bs;
        elsif B5='1' then
          estadoSiguiente<=cf;
        end if;
      when others => estadoSiguiente<=estadoActual;
    end case;
  end process;

  SalidasSelModo : process (estadoActual)
  begin
    case(estadoActual) is
      when as=>
        code1<="1111";
        code2<="1111";
        code3<="1111";
        code4<="1111";
        code5<="1111";
        code6<="1111";
        code7<="1011";
        code8<="1010";
        Enable_A<='0';
        Enable_B<='0';
        Enable_C<='0';
        Enable_S<='1';
      when bs=>
        code1<="1111";
        code2<="1111";
        code3<="1111";
        code4<="1111";
        code5<="1111";
    end case;
  end process;

```



```

        code6<="1111";
        code7<="1100";
        code8<="1101";
        Enable_A<='0';
        Enable_B<='0';
        Enable_C<='0';
        Enable_S<='1';
    when cs=>
        code1<="1111";
        code2<="1111";
        code3<="1111";
        code4<="1111";
        code5<="1111";
        code6<="1111";
        code7<="1111";
        code8<="1110";
        Enable_A<='0';
        Enable_B<='0';
        Enable_C<='0';
        Enable_S<='1';
    when af=>
        Enable_A<='1';
        Enable_B<='0';
        Enable_C<='0';
        Enable_S<='0';
    when bf=>
        Enable_A<='0';
        Enable_B<='1';
        Enable_C<='0';
        Enable_S<='0';
    when cf=>
        Enable_A<='0';
        Enable_B<='0';
        Enable_C<='1';
        Enable_S<='0';
    end case;
end process;

end Behavioral;

```

Modo temporizador.

Se trata de la entidad más compleja, no obstante, para su implementación se ha subdividido en entidades con tareas más sencillas:

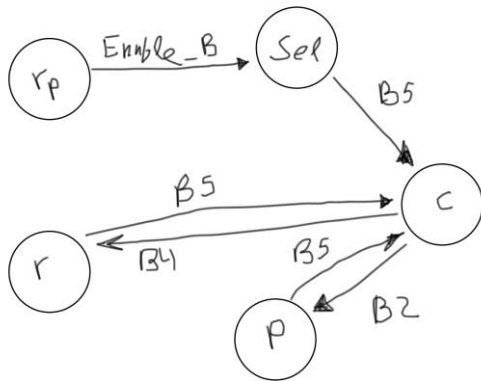
- ▼ ● Modo_Temp1 : Modo_Temp(Behavioral) (Modo_Temp.vhd) (3)
 - ▼ ● Sel_Cuenta1 : Sel_Cuenta(Behavioral) (Sel_Cuenta.vhd) (1)
 - Inst_clk10khz : clk10kHz(Behavioral) (clk10khz.vhd)
 - ▼ ● Cuenta_atras1 : Cuenta_atras(Behavioral) (Cuenta_atras.vhd) (1)
 - Inst_clk1hz : clk1Hz(Behavioral) (CLK1hz.vhd)
 - Mux_Temp1 : Mux_Temp(Behavioral) (Mux_Temp.vhd)

Sel cuenta es una entidad encargada de, mientras esté activada su señal de habilitación, generar unas señales de salida en función del número de veces que sea pulsado cada botón de los que correspondan en las instrucciones, por lo que su funcionamiento es muy similar al de un contador, estas señales generadas irán a la salida para ser mostrada en los displays y a la vez los cargará en la entidad cuenta atrás para que sea el valor de inicio de la cuenta.

En el momento que se pulsa B5 se pasa al estado de contar, por lo que ahora la salida quedará determinada por la entidad cuenta atrás, nuevamente para que no se

solapen las señales será necesario un multiplexor, que vendrá gobernado por la máquina de estados de moore del modo temporizador.

La máquina de estados de este modo viene dada por el siguiente diagrama:



Cuando Enable_B vuelve a valer 0 se regresa forzosamente al estado de reposo.

Para cumplir con esta funcionalidad se realiza la siguiente descripción:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity Modo_Temp is
  Port (
    CLK : in std_logic;
    B1 : in std_logic;
    B2 : in std_logic;
    B3 : in std_logic;
    B4 : in std_logic;
    B5 : in std_logic;
    code1 : out std_logic_vector(3 downto 0);
    code2 : out std_logic_vector(3 downto 0);
    code3 : out std_logic_vector(3 downto 0);
    code4 : out std_logic_vector(3 downto 0);
    code5 : out std_logic_vector(3 downto 0);
    code6 : out std_logic_vector(3 downto 0);
    code7 : out std_logic_vector(3 downto 0);
    code8 : out std_logic_vector(3 downto 0);
    Enable_B : in std_logic;
    led : out std_logic
  );
end Modo_Temp;

architecture Behavioral of Modo_Temp is
  type estados is (Sel,c,p,r,rp);
  signal estadoActual, siguienteEstado:estados;

  signal Enable_sel : std_logic:='0';
  signal Enable_count : std_logic:='0';
  signal Reset_aux : std_logic:='0';

  signal code1_Sel : std_logic_vector(3 downto 0);
  signal code2_Sel : std_logic_vector(3 downto 0);
  signal code3_Sel : std_logic_vector(3 downto 0);
  signal code4_Sel : std_logic_vector(3 downto 0);

  signal code1_Count : std_logic_vector(3 downto 0);
  signal code2_Count : std_logic_vector(3 downto 0);
  signal code3_Count : std_logic_vector(3 downto 0);
  signal code4_Count : std_logic_vector(3 downto 0);

  COMPONENT Sel_Cuenta

```

```

PORT (
    clk : in std_logic;
    B1 : in std_logic;
    B2 : in std_logic;
    B3 : in std_logic;
    B4 : in std_logic;
    Enable : in std_logic;
    code1_Sel : out std_logic_vector(3 downto 0);
    code2_Sel : out std_logic_vector(3 downto 0);
    code3_Sel : out std_logic_vector(3 downto 0);
    code4_Sel : out std_logic_vector(3 downto 0)
);
END COMPONENT;

COMPONENT Cuenta_atras
PORT (
    CLK : in std_logic;
    Enable_count : in std_logic;
    Reset : in std_logic;
    code1_in : in std_logic_vector(3 downto 0);
    code2_in : in std_logic_vector(3 downto 0);
    code3_in : in std_logic_vector(3 downto 0);
    code4_in : in std_logic_vector(3 downto 0);
    code1_out : out std_logic_vector(3 downto 0);--unidades de segundo
    code2_out : out std_logic_vector(3 downto 0);--decenas de segundo
    code3_out : out std_logic_vector(3 downto 0);--unidades de minuto
    code4_out : out std_logic_vector(3 downto 0);--decenas de minuto
    led : out std_logic
);
END COMPONENT;

COMPONENT Mux_Temp
PORT (
    clk: in std_logic;
    Enable_Sel : in std_logic;
    code1_Sel : in std_logic_vector(3 downto 0);
    code2_Sel : in std_logic_vector(3 downto 0);
    code3_Sel : in std_logic_vector(3 downto 0);
    code4_Sel : in std_logic_vector(3 downto 0);
    code1_Count : in std_logic_vector(3 downto 0);
    code2_Count : in std_logic_vector(3 downto 0);
    code3_Count : in std_logic_vector(3 downto 0);
    code4_Count : in std_logic_vector(3 downto 0);
    code1 : out std_logic_vector(3 downto 0);
    code2 : out std_logic_vector(3 downto 0);
    code3 : out std_logic_vector(3 downto 0);
    code4 : out std_logic_vector(3 downto 0)
);
END COMPONENT;

begin

Sel_Cuenta1: Sel_Cuenta PORT MAP(
    clk=>CLK,
    B1=>B1,
    B2=>B2,
    B3=>B3,
    B4=>B4,
    Enable=>Enable_sel,
    code1_Sel=>code1_Sel,
    code2_Sel=>code2_Sel,
    code3_Sel=>code3_Sel,
    code4_Sel=>code4_Sel
);

Cuenta_atras1 : Cuenta_atras PORT MAP(
    CLK => CLK,
    Enable_count => Enable_count,
    Reset => Reset_aux,
    code1_in => code1_Sel,
    code2_in => code2_Sel,
    code3_in => code3_Sel,
    code4_in => code4_Sel,
    code1_out => code1_Count,--unidades de segundo
    code2_out => code2_Count,--decenas de segundo

```

```

        code3_out => code3_Count,--unidades de minuto
        code4_out => code4_Count,--decenas de minuto
        led => led
    );

    Mux_Temp1 : Mux_Temp PORT MAP(
        Enable_Sel=>Enable_Sel,
        code1_Sel=>code1_Sel,
        code2_Sel=>code2_Sel,
        code3_Sel=>code3_Sel,
        code4_Sel=>code4_Sel,
        code1_Count=>code1_Count,
        code2_Count=>code2_Count,
        code3_Count=>code3_Count,
        code4_Count=>code4_Count,
        code1=>code1,
        code2=>code2,
        code3=>code3,
        code4=>code4,
        clk=>clk
    );

    modoTemp : process (Enable_B)
    begin
        if Enable_B='1' then
            code8<="1100";
            code7<="1101";
            code6<="1111";
            code5<="1111";
        end if;
    end process;

    state_reg: process (clk,enable_b)
    begin
        if enable_b='0' then
            estadoActual<=rp;
        elsif rising_edge(clk) then
            estadoActual<= siguienteEstado;
        end if;
    end process;

    maquinaestados : process (Enable_B,estadoActual, B2,B4,B5)
    begin
        siguienteEstado<=estadoActual;
        case estadoActual is
            when Sel=>
                if enable_sel='1' and B5='1' then
                    siguienteEstado<=c;
                end if;
            when c=>
                if B2='1' then
                    siguienteEstado<=p;
                elsif B4='1' then
                    siguienteEstado<=r;
                end if;
            when p=>
                if B4='1' then
                    siguienteEstado<=r;
                elsif B5='1' then
                    siguienteEstado<=c;
                end if;
            when r=>
                if B5='1' then
                    siguienteEstado<=c;
                end if;
            when rp=>
                if enable_b='1' then
                    siguienteEstado<=sel;
                end if;
        end case;
    end process;

    salidas: process(estadoActual)
    begin
        case estadoActual is
            when Sel=>

```

```

        enable_sel<='1';
        enable_count<='0';
        reset_aux<='1';
    when c=>
        enable_sel<='0';
        enable_count<='1';
        reset_aux<='0';
    when p=>
        enable_sel<='0';
        enable_count<='0';
        reset_aux<='0';
    when r=>
        enable_sel<='0';
        enable_count<='0';
        reset_aux<='1';
    when rp=>
        enable_sel<='0';
        enable_count<='0';
        reset_aux<='0';
    end case;
end process;
end Behavioral;

```

Modo ajedrez.

Es una entidad relativamente sencilla, cuenta con dos instanciaciones de Cuenta_atrás, una por el reloj de cada jugador.

```

▼ ● Ajedrez1 : Ajedrez(Behavioral) (Ajedrez.vhd) (2)
    > ● JugadorA : Cuenta_atras(Behavioral) (Cuenta_atras.vhd) (1)
    > ● JugadorB : Cuenta_atras(Behavioral) (Cuenta_atras.vhd) (1)

```

Contiene un process que funciona como una pequeña máquina de estados que gestiona la activación y desactivación de las respectivas cuentas cuando corresponda, así como su reinicio al pulsar B1.

Para lograr esta funcionalidad se realiza la siguiente descripción:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Ajedrez is
    Port (
        CLK : in std_logic;
        B1 : in std_logic;
        B2 : in std_logic;
        B3 : in std_logic;
        B4 : in std_logic;
        B5 : in std_logic;
        code1 : out std_logic_vector(3 downto 0);
        code2 : out std_logic_vector(3 downto 0);
        code3 : out std_logic_vector(3 downto 0);
        code4 : out std_logic_vector(3 downto 0);
        code5 : out std_logic_vector(3 downto 0);
        code6 : out std_logic_vector(3 downto 0);
        code7 : out std_logic_vector(3 downto 0);
        code8 : out std_logic_vector(3 downto 0);
        Enable_C : in std_logic
    );
end Ajedrez;

architecture Behavioral of Ajedrez is

    signal Reset_aux_A : std_logic:='0';
    signal Reset_aux_B : std_logic:='0';

    signal Enable_count_A : std_logic:='0';

```

```

signal Enable_count_B : std_logic:='0';

signal ledA : std_logic:='0';
signal ledB : std_logic:='0';

COMPONENT Cuenta_atras
PORT (
    CLK : in std_logic;
    Enable_count : in std_logic;
    Reset : in std_logic;
    code1_in : in std_logic_vector(3 downto 0);
    code2_in : in std_logic_vector(3 downto 0);
    code3_in : in std_logic_vector(3 downto 0);
    code4_in : in std_logic_vector(3 downto 0);
    code1_out : out std_logic_vector(3 downto 0);--unidades de segundo
    code2_out : out std_logic_vector(3 downto 0);--decenas de segundo
    code3_out : out std_logic_vector(3 downto 0);--unidades de minuto
    code4_out : out std_logic_vector(3 downto 0);--decenas de minuto
    led : out std_logic
);
END COMPONENT;

begin

    JugadorA : Cuenta_atras PORT MAP(
        CLK => CLK,
        Enable_count => Enable_count_A,
        Reset => Reset_aux_A,
        code1_in => "0000",
        code2_in => "0000",
        code3_in => "0000",
        code4_in => "0001",
        code1_out => code1,--unidades de segundo
        code2_out => code2,--decenas de segundo
        code3_out => code3,--unidades de minuto
        code4_out => code4,--decenas de minuto
        led => ledA
    );

    JugadorB : Cuenta_atras PORT MAP(
        CLK => CLK,
        Enable_count => Enable_count_B,
        Reset => Reset_aux_B,
        code1_in => "0000",
        code2_in => "0000",
        code3_in => "0000",
        code4_in => "0001",
        code1_out => code5,--unidades de segundo
        code2_out => code6,--decenas de segundo
        code3_out => code7,--unidades de minuto
        code4_out => code8,--decenas de minuto
        led => ledB
    );

    maquinaestados : process(B1,B3,B4,Enable_C)
    begin
        if Enable_C='0' or B1='1' then
            Reset_aux_A<='1';
            Reset_aux_B<='1';
            Enable_count_A<='0';
            Enable_count_B<='0';
        elsif Enable_C='1' and B4='1' then
            Enable_count_A<='1';
            Enable_count_B<='1';
            Reset_aux_A<='0';
            Reset_aux_B<='0';
        elsif Enable_C='1' and B3='1' then
            Enable_count_A<='0';
            Enable_count_B<='1';
            Reset_aux_A<='0';
            Reset_aux_B<='0';
        end if;
    end process;
end;

```

```

end process;

end Behavioral;

```

Entidad Supermux

Pese a lo larga que es su descripción VHDL es una entidad con un funcionamiento muy sencillo, que simplemente en función de la entrada de habilitación que esté activada copia el el valor de los vectores que corresponda en su salida.

Para ello se emplea la siguiente descripción:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SuperMux is
  Port (
    Enable_A : in std_logic;
    Enable_B : in std_logic;
    Enable_C : in std_logic;
    Enable_S : in std_logic;

    code1_ME : in std_logic_vector(3 downto 0);
    code2_ME : in std_logic_vector(3 downto 0);
    code3_ME : in std_logic_vector(3 downto 0);
    code4_ME : in std_logic_vector(3 downto 0);
    code5_ME : in std_logic_vector(3 downto 0);
    code6_ME : in std_logic_vector(3 downto 0);
    code7_ME : in std_logic_vector(3 downto 0);
    code8_ME : in std_logic_vector(3 downto 0);

    code1_A : in std_logic_vector(3 downto 0);
    code2_A : in std_logic_vector(3 downto 0);
    code3_A : in std_logic_vector(3 downto 0);
    code4_A : in std_logic_vector(3 downto 0);
    code5_A : in std_logic_vector(3 downto 0);
    code6_A : in std_logic_vector(3 downto 0);
    code7_A : in std_logic_vector(3 downto 0);
    code8_A : in std_logic_vector(3 downto 0);

    code1_B : in std_logic_vector(3 downto 0);
    code2_B : in std_logic_vector(3 downto 0);
    code3_B : in std_logic_vector(3 downto 0);
    code4_B : in std_logic_vector(3 downto 0);
    code5_B : in std_logic_vector(3 downto 0);
    code6_B : in std_logic_vector(3 downto 0);
    code7_B : in std_logic_vector(3 downto 0);
    code8_B : in std_logic_vector(3 downto 0);

    code1_C : in std_logic_vector(3 downto 0);
    code2_C : in std_logic_vector(3 downto 0);
    code3_C : in std_logic_vector(3 downto 0);
    code4_C : in std_logic_vector(3 downto 0);
    code5_C : in std_logic_vector(3 downto 0);
    code6_C : in std_logic_vector(3 downto 0);
    code7_C : in std_logic_vector(3 downto 0);
    code8_C : in std_logic_vector(3 downto 0);

    code1 : out std_logic_vector(3 downto 0);
    code2 : out std_logic_vector(3 downto 0);
    code3 : out std_logic_vector(3 downto 0);
    code4 : out std_logic_vector(3 downto 0);
    code5 : out std_logic_vector(3 downto 0);
    code6 : out std_logic_vector(3 downto 0);
    code7 : out std_logic_vector(3 downto 0);
    code8 : out std_logic_vector(3 downto 0)
  );
end SuperMux;

architecture Behavioral of SuperMux is
begin

```

```

process(Enable_A,Enable_B,Enable_C,Enable_S)
begin
    if Enable_A='1' then
        code1<=code1_A;
        code2<=code2_A;
        code3<=code3_A;
        code4<=code4_A;
        code5<=code5_A;
        code6<=code6_A;
        code7<=code7_A;
        code8<=code8_A;
    elsif Enable_B='1' then
        code1<=code1_B;
        code2<=code2_B;
        code3<=code3_B;
        code4<=code4_B;
        code5<=code5_B;
        code6<=code6_B;
        code7<=code7_B;
        code8<=code8_B;
    elsif Enable_C='1' then
        code1<=code1_C;
        code2<=code2_C;
        code3<=code3_C;
        code4<=code4_C;
        code5<=code5_C;
        code6<=code6_C;
        code7<=code7_C;
        code8<=code8_C;
    elsif Enable_S='1' then
        code1<=code1_ME;
        code2<=code2_ME;
        code3<=code3_ME;
        code4<=code4_ME;
        code5<=code5_ME;
        code6<=code6_ME;
        code7<=code7_ME;
        code8<=code8_ME;
    end if;
end process;
end Behavioral;

```

Entidad Top

Al igual que antes, la entidad Top recogerá todas las entidades dentro de sí misma, conteniendo también todas las señales que interconectan los diferentes bloques funcionales del diseño.

```

end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TOP is
    Port (
        CLK : in std_logic;
        B_L : in std_logic;
        B_R : in std_logic;
        B_U : in std_logic;
        B_D : in std_logic;
        B_C : in std_logic;
        reset : in std_logic;
        refrescar_anodo : out std_logic_vector(7 downto 0); --vector que pone a 1 el ánodo
        correspondiente para actualizar
        salida_disp : out std_logic_vector(6 downto 0); --salida de los displays
        led : out std_logic
    );
end TOP;

architecture Behavioral of TOP is

    signal sync_auxL: std_logic;
    signal sync_auxR: std_logic;
    signal sync_auxU: std_logic;
    signal sync_auxD: std_logic;

```



```

signal sync_auxC: std_logic;

signal B_L_aux: std_logic;
signal B_R_aux: std_logic;
signal B_U_aux: std_logic;
signal B_D_aux: std_logic;
signal B_C_aux: std_logic;
signal Reset_aux: std_logic;

signal Enable_A: std_logic;
signal Enable_B: std_logic;
signal Enable_C: std_logic;
signal Enable_S: std_logic;

signal code1_ME : std_logic_vector(3 downto 0);
signal code2_ME : std_logic_vector(3 downto 0);
signal code3_ME : std_logic_vector(3 downto 0);
signal code4_ME : std_logic_vector(3 downto 0);
signal code5_ME : std_logic_vector(3 downto 0);
signal code6_ME : std_logic_vector(3 downto 0);
signal code7_ME : std_logic_vector(3 downto 0);
signal code8_ME : std_logic_vector(3 downto 0);

signal code1_A : std_logic_vector(3 downto 0);
signal code2_A : std_logic_vector(3 downto 0);
signal code3_A : std_logic_vector(3 downto 0);
signal code4_A : std_logic_vector(3 downto 0);
signal code5_A : std_logic_vector(3 downto 0);
signal code6_A : std_logic_vector(3 downto 0);
signal code7_A : std_logic_vector(3 downto 0);
signal code8_A : std_logic_vector(3 downto 0);

signal code1_B : std_logic_vector(3 downto 0);
signal code2_B : std_logic_vector(3 downto 0);
signal code3_B : std_logic_vector(3 downto 0);
signal code4_B : std_logic_vector(3 downto 0);
signal code5_B : std_logic_vector(3 downto 0);
signal code6_B : std_logic_vector(3 downto 0);
signal code7_B : std_logic_vector(3 downto 0);
signal code8_B : std_logic_vector(3 downto 0);

signal code1_C : std_logic_vector(3 downto 0);
signal code2_C : std_logic_vector(3 downto 0);
signal code3_C : std_logic_vector(3 downto 0);
signal code4_C : std_logic_vector(3 downto 0);
signal code5_C : std_logic_vector(3 downto 0);
signal code6_C : std_logic_vector(3 downto 0);
signal code7_C : std_logic_vector(3 downto 0);
signal code8_C : std_logic_vector(3 downto 0);

signal code1_aux : std_logic_vector(3 downto 0);
signal code2_aux : std_logic_vector(3 downto 0);
signal code3_aux : std_logic_vector(3 downto 0);
signal code4_aux : std_logic_vector(3 downto 0);
signal code5_aux : std_logic_vector(3 downto 0);
signal code6_aux : std_logic_vector(3 downto 0);
signal code7_aux : std_logic_vector(3 downto 0);
signal code8_aux : std_logic_vector(3 downto 0);

COMPONENT GestorEntradas
PORT (
    CLK : in std_logic;
    B_L : in std_logic;
    B_R : in std_logic;
    B_U : in std_logic;
    B_D : in std_logic;
    B_C : in std_logic;
    reset : in std_logic;
    B_L_out : out std_logic;
    B_R_out : out std_logic;
    B_U_out : out std_logic;
    B_D_out : out std_logic;
    B_C_out : out std_logic;
    reset_out : out std_logic

```

```

    );
END COMPONENT;

COMPONENT MaquinaEstados
PORT (
    clk : in std_logic;
    B1 : in std_logic;
    B2 : in std_logic;
    B3 : in std_logic;
    B4 : in std_logic;
    B5 : in std_logic;
    code1 : out std_logic_vector(3 downto 0);
    code2 : out std_logic_vector(3 downto 0);
    code3 : out std_logic_vector(3 downto 0);
    code4 : out std_logic_vector(3 downto 0);
    code5 : out std_logic_vector(3 downto 0);
    code6 : out std_logic_vector(3 downto 0);
    code7 : out std_logic_vector(3 downto 0);
    code8 : out std_logic_vector(3 downto 0);
    Reset : in std_logic;
    Enable_A : out std_logic := '0';
    Enable_B : out std_logic := '0';
    Enable_C : out std_logic := '0';
    Enable_S : out std_logic := '1'
);
END COMPONENT;

COMPONENT Modo_Crono
PORT (
    CLK : in std_logic;
    code1 : out std_logic_vector(3 downto 0);
    code2 : out std_logic_vector(3 downto 0);
    code3 : out std_logic_vector(3 downto 0);
    code4 : out std_logic_vector(3 downto 0);
    code5 : out std_logic_vector(3 downto 0);
    code6 : out std_logic_vector(3 downto 0);
    code7 : out std_logic_vector(3 downto 0);
    code8 : out std_logic_vector(3 downto 0);
    Enable_A : in std_logic;
    Start : in std_logic;
    Pause : in std_logic;
    Reset : in std_logic
);
END COMPONENT;

COMPONENT Modo_Temp
PORT (
    CLK : in std_logic;
    B1 : in std_logic;
    B2 : in std_logic;
    B3 : in std_logic;
    B4 : in std_logic;
    B5 : in std_logic;
    code1 : out std_logic_vector(3 downto 0);
    code2 : out std_logic_vector(3 downto 0);
    code3 : out std_logic_vector(3 downto 0);
    code4 : out std_logic_vector(3 downto 0);
    code5 : out std_logic_vector(3 downto 0);
    code6 : out std_logic_vector(3 downto 0);
    code7 : out std_logic_vector(3 downto 0);
    code8 : out std_logic_vector(3 downto 0);
    Enable_B : in std_logic;
    led : out std_logic
);
END COMPONENT;

COMPONENT Ajedrez
PORT (
    CLK : in std_logic;
    B1 : in std_logic;
    B2 : in std_logic;
    B3 : in std_logic;
    B4 : in std_logic;
    B5 : in std_logic;
    code1 : out std_logic_vector(3 downto 0);
    code2 : out std_logic_vector(3 downto 0);

```

```

        code3 : out std_logic_vector(3 downto 0);
        code4 : out std_logic_vector(3 downto 0);
        code5 : out std_logic_vector(3 downto 0);
        code6 : out std_logic_vector(3 downto 0);
        code7 : out std_logic_vector(3 downto 0);
        code8 : out std_logic_vector(3 downto 0);
        Enable_C : in std_logic
        --led : out std_logic
    );
END COMPONENT;

COMPONENT SuperMux
PORT (
    Enable_A : in std_logic;
    Enable_B : in std_logic;
    Enable_C : in std_logic;
    Enable_S : in std_logic;

    code1_ME : in std_logic_vector(3 downto 0);
    code2_ME : in std_logic_vector(3 downto 0);
    code3_ME : in std_logic_vector(3 downto 0);
    code4_ME : in std_logic_vector(3 downto 0);
    code5_ME : in std_logic_vector(3 downto 0);
    code6_ME : in std_logic_vector(3 downto 0);
    code7_ME : in std_logic_vector(3 downto 0);
    code8_ME : in std_logic_vector(3 downto 0);

    code1_A : in std_logic_vector(3 downto 0);
    code2_A : in std_logic_vector(3 downto 0);
    code3_A : in std_logic_vector(3 downto 0);
    code4_A : in std_logic_vector(3 downto 0);
    code5_A : in std_logic_vector(3 downto 0);
    code6_A : in std_logic_vector(3 downto 0);
    code7_A : in std_logic_vector(3 downto 0);
    code8_A : in std_logic_vector(3 downto 0);

    code1_B : in std_logic_vector(3 downto 0);
    code2_B : in std_logic_vector(3 downto 0);
    code3_B : in std_logic_vector(3 downto 0);
    code4_B : in std_logic_vector(3 downto 0);
    code5_B : in std_logic_vector(3 downto 0);
    code6_B : in std_logic_vector(3 downto 0);
    code7_B : in std_logic_vector(3 downto 0);
    code8_B : in std_logic_vector(3 downto 0);

    code1_C : in std_logic_vector(3 downto 0);
    code2_C : in std_logic_vector(3 downto 0);
    code3_C : in std_logic_vector(3 downto 0);
    code4_C : in std_logic_vector(3 downto 0);
    code5_C : in std_logic_vector(3 downto 0);
    code6_C : in std_logic_vector(3 downto 0);
    code7_C : in std_logic_vector(3 downto 0);
    code8_C : in std_logic_vector(3 downto 0);

    code1 : out std_logic_vector(3 downto 0);
    code2 : out std_logic_vector(3 downto 0);
    code3 : out std_logic_vector(3 downto 0);
    code4 : out std_logic_vector(3 downto 0);
    code5 : out std_logic_vector(3 downto 0);
    code6 : out std_logic_vector(3 downto 0);
    code7 : out std_logic_vector(3 downto 0);
    code8 : out std_logic_vector(3 downto 0)
);
END COMPONENT;

COMPONENT Control_anodo
PORT (
    CLK : in std_logic;
    code1 : in std_logic_vector(3 downto 0);
    code2 : in std_logic_vector(3 downto 0);
    code3 : in std_logic_vector(3 downto 0);
    code4 : in std_logic_vector(3 downto 0);
    code5 : in std_logic_vector(3 downto 0);
    code6 : in std_logic_vector(3 downto 0);
    code7 : in std_logic_vector(3 downto 0);

```

```

        code8 : in std_logic_vector(3 downto 0);
        refrescar_anodo : out std_logic_vector(7 downto 0);
        salida_disp : out std_logic_vector(6 downto 0)
    );
END COMPONENT;

begin

    GestorEntradas1 : GestorEntradas PORT MAP (
        CLK => CLK,
        B_L => B_L,
        B_R => B_R,
        B_U => B_U,
        B_D => B_D,
        B_C => B_C,
        reset => reset,
        B_L_out => B_L_aux,
        B_R_out => B_R_aux,
        B_U_out => B_U_aux,
        B_D_out => B_D_aux,
        B_C_out => B_C_aux,
        reset_out => Reset_aux
    );

    MaquinaEstados1 : MaquinaEstados PORT MAP (
        clk=>clk,
        B1=>B_U_aux,
        B2=>B_D_aux,
        B3=>B_R_aux,
        B4=>B_L_aux,
        B5=>B_C_aux,
        code1=>code1_ME,
        code2=>code2_ME,
        code3=>code3_ME,
        code4=>code4_ME,
        code5=>code5_ME,
        code6=>code6_ME,
        code7=>code7_ME,
        code8=>code8_ME,
        Reset=>Reset,
        Enable_A=>Enable_A,
        Enable_B=>Enable_B,
        Enable_C=>Enable_C,
        Enable_S=>Enable_S
    );

    Modo_Crono1 : Modo_Crono PORT MAP (
        CLK=>clk,
        code1=>code1_A,
        code2=>code2_A,
        code3=>code3_A,
        code4=>code4_A,
        code5=>code5_A,
        code6=>code6_A,
        code7=>code7_A,
        code8=>code8_A,
        Enable_A=>Enable_A,
        Start=>B_C_aux,
        Pause=>B_D_aux,
        Reset=>B_L_aux
    );

    Modo_Temp1 : Modo_Temp PORT MAP (
        CLK=>clk,
        code1=>code1_B,
        code2=>code2_B,
        code3=>code3_B,
        code4=>code4_B,
        code5=>code5_B,
        code6=>code6_B,
        code7=>code7_B,
        code8=>code8_B,
        Enable_B=>Enable_B,
        led=>led,
        B1=>B_U_aux,
        B2=>B_D_aux,

```

```

        B3=>B_R_aux,
        B4=>B_L_aux,
        B5=>B_C_aux
    );

Ajedrez1 : Ajedrez PORT MAP(
    CLK=>clk,
    code1=>code1_C,
    code2=>code2_C,
    code3=>code3_C,
    code4=>code4_C,
    code5=>code5_C,
    code6=>code6_C,
    code7=>code7_C,
    code8=>code8_C,
    Enable_C=>Enable_C,
    --led=>led,
    B1=>B_U_aux,
    B2=>B_D_aux,
    B3=>B_R_aux,
    B4=>B_L_aux,
    B5=>B_C_aux
);

SuperMux1 : SuperMux PORT MAP(
    Enable_A=>Enable_A,
    Enable_B=>Enable_B,
    Enable_C=>Enable_C,
    Enable_S=>Enable_S,

    code1_ME=>code1_ME,
    code2_ME=>code2_ME,
    code3_ME=>code3_ME,
    code4_ME=>code4_ME,
    code5_ME=>code5_ME,
    code6_ME=>code6_ME,
    code7_ME=>code7_ME,
    code8_ME=>code8_ME,

    code1_A=>code1_A,
    code2_A=>code2_A,
    code3_A=>code3_A,
    code4_A=>code4_A,
    code5_A=>code5_A,
    code6_A=>code6_A,
    code7_A=>code7_A,
    code8_A=>code8_A,

    code1_B=>code1_B,
    code2_B=>code2_B,
    code3_B=>code3_B,
    code4_B=>code4_B,
    code5_B=>code5_B,
    code6_B=>code6_B,
    code7_B=>code7_B,
    code8_B=>code8_B,

    code1_C=>code1_C,
    code2_C=>code2_C,
    code3_C=>code3_C,
    code4_C=>code4_C,
    code5_C=>code5_C,
    code6_C=>code6_C,
    code7_C=>code7_C,
    code8_C=>code8_C,

    code1=>code1_aux,
    code2=>code2_aux,
    code3=>code3_aux,
    code4=>code4_aux,
    code5=>code5_aux,
    code6=>code6_aux,
    code7=>code7_aux,
    code8=>code8_aux
);

```

```

Control_Anodo1 : Control_anodo PORT MAP(
    CLK=>clk,
    code1=>code1_aux,
    code2=>code2_aux,
    code3=>code3_aux,
    code4=>code4_aux,
    code5=>code5_aux,
    code6=>code6_aux,
    code7=>code7_aux,
    code8=>code8_aux,
    refrescar_anodo=>refrescar_anodo,
    salida_disp=>salida_disp
);

end Behavioral;

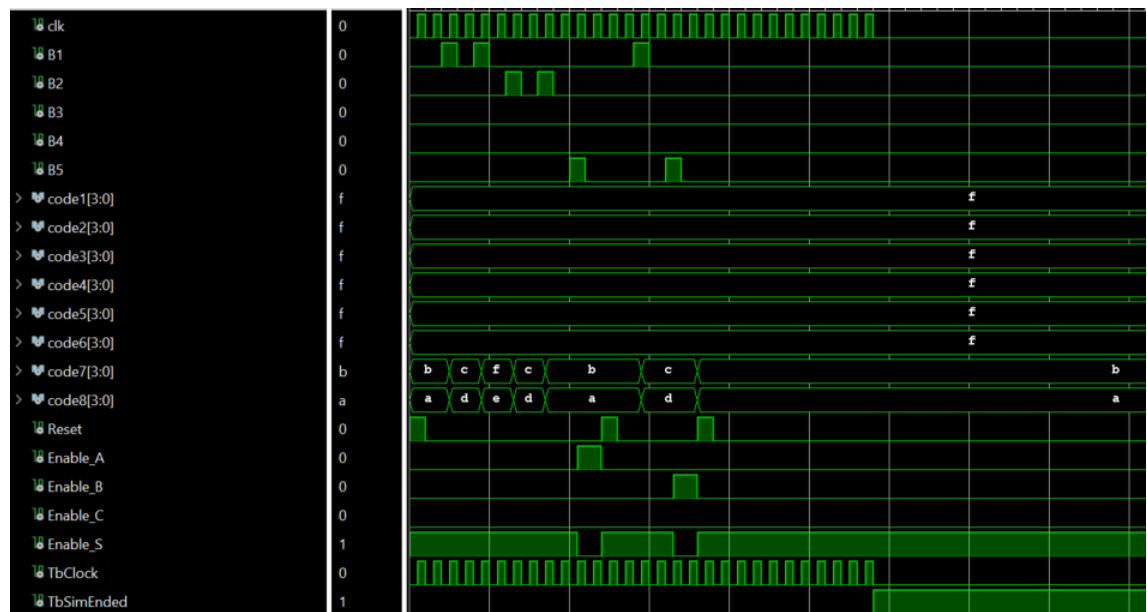
```

7.6. Simulación.

Todos los relojes de 1Hz y 10KHz se han sustituido momentáneamente por el reloj CLK de entrada con el objetivo de simplificar el testbench.

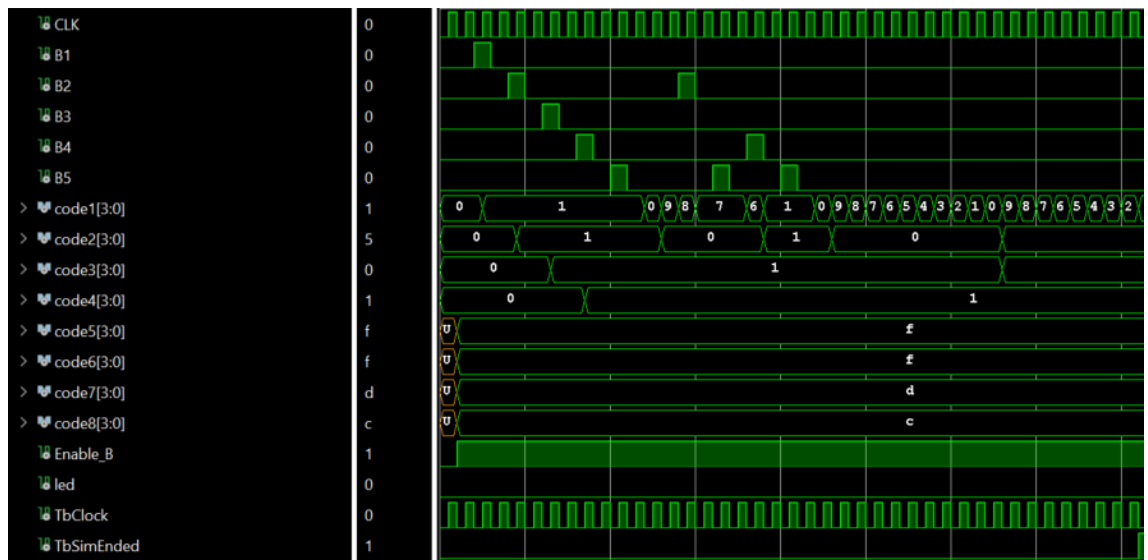
Mostraremos únicamente los resultados de las simulaciones y comentaremos sus resultados, ya que la descripción del testbench es prácticamente análoga en todos y repetitiva.

Máquina de estados



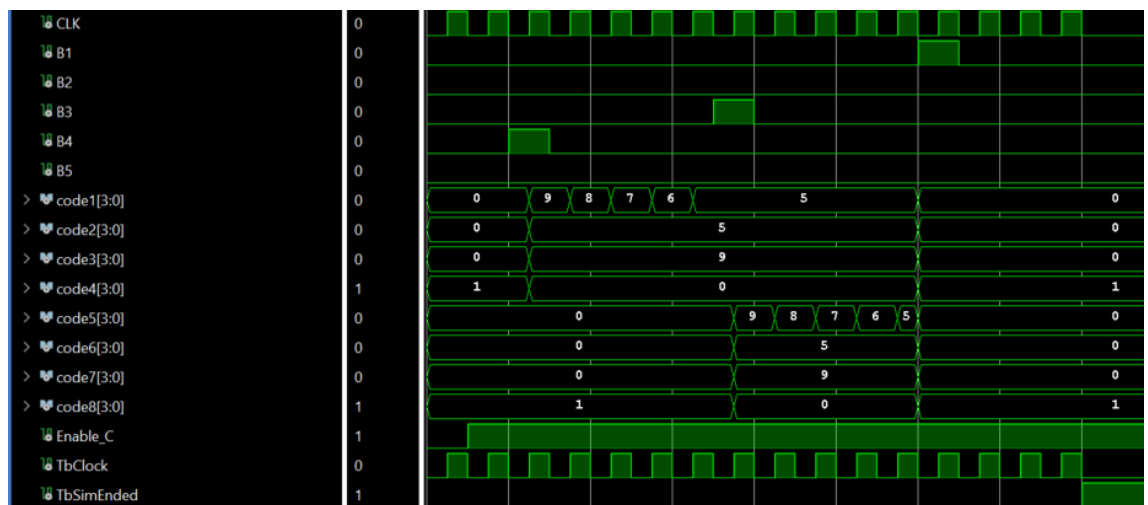
Como se observa cambia el valor de la salida a los códigos correspondientes para mostrar el modo al que se desea entrar y cuando se pulsa B5 se activa el Enable que corresponda, también se aprecia que cuando se pulsa Reset se vuelve al estado inicial.

Modo Temporizador



Se observa que funciona perfectamente el Set de la cuenta, así como el momento en el que se pulsa el botón B5 y se pasa a la cuenta atrás y la salida queda gobernada por la entidad cuenta atrás, al introducir pulsos en los botones asociados a la pausa y al reset vemos que también cumplen su función como era de esperar.

Modo Ajedrez



Se observa como se alterna la cuenta al pulsar un botón u otro y se reinicia de forma perfecta

8. CONSTRAINS

```
set_property -dict { PACKAGE_PIN E3      IOSTANDARD LVCMOS33 } [get_ports { CLK }];
set_property -dict { PACKAGE_PIN H17     IOSTANDARD LVCMOS33 } [get_ports { led }];
set_property -dict { PACKAGE_PIN T10     IOSTANDARD LVCMOS33 } [get_ports { salida_disp[6] }];
set_property -dict { PACKAGE_PIN R10     IOSTANDARD LVCMOS33 } [get_ports { salida_disp[5] }];
set_property -dict { PACKAGE_PIN K16     IOSTANDARD LVCMOS33 } [get_ports { salida_disp[4] }];
set_property -dict { PACKAGE_PIN K13     IOSTANDARD LVCMOS33 } [get_ports { salida_disp[3] }];
set_property -dict { PACKAGE_PIN P15     IOSTANDARD LVCMOS33 } [get_ports { salida_disp[2] }];
set_property -dict { PACKAGE_PIN T11     IOSTANDARD LVCMOS33 } [get_ports { salida_disp[1] }];
set_property -dict { PACKAGE_PIN L18     IOSTANDARD LVCMOS33 } [get_ports { salida_disp[0] }];

#set_property -dict { PACKAGE_PIN H15     IOSTANDARD LVCMOS33 } [get_ports { DP }]; #IO_L19N_T.

set_property -dict { PACKAGE_PIN J17     IOSTANDARD LVCMOS33 } [get_ports { refrescar_anodo[0] }];
set_property -dict { PACKAGE_PIN J18     IOSTANDARD LVCMOS33 } [get_ports { refrescar_anodo[1] }];
set_property -dict { PACKAGE_PIN T9      IOSTANDARD LVCMOS33 } [get_ports { refrescar_anodo[2] }];
set_property -dict { PACKAGE_PIN J14     IOSTANDARD LVCMOS33 } [get_ports { refrescar_anodo[3] }];
set_property -dict { PACKAGE_PIN P14     IOSTANDARD LVCMOS33 } [get_ports { refrescar_anodo[4] }];
set_property -dict { PACKAGE_PIN T14     IOSTANDARD LVCMOS33 } [get_ports { refrescar_anodo[5] }];
set_property -dict { PACKAGE_PIN K2      IOSTANDARD LVCMOS33 } [get_ports { refrescar_anodo[6] }];
set_property -dict { PACKAGE_PIN U13     IOSTANDARD LVCMOS33 } [get_ports { refrescar_anodo[7] }];

##Buttons

set_property -dict { PACKAGE_PIN C12     IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L3P_!

set_property -dict { PACKAGE_PIN N17     IOSTANDARD LVCMOS33 } [get_ports { B_C }]; #IO_L9P_T1_
set_property -dict { PACKAGE_PIN M18     IOSTANDARD LVCMOS33 } [get_ports { B_U }]; #IO_L4N_T0_
set_property -dict { PACKAGE_PIN P17     IOSTANDARD LVCMOS33 } [get_ports { B_L }]; #IO_L12P_T.
set_property -dict { PACKAGE_PIN M17     IOSTANDARD LVCMOS33 } [get_ports { B_R }]; #IO_L10N_T.
set_property -dict { PACKAGE_PIN P18     IOSTANDARD LVCMOS33 } [get_ports { B_D }]; #IO_L9N_T1_
```