ESCUELA SUPERIOR POLITECNICA DEL LITORAL

MATERIA

LABORATORIO DE SISTEMAS DIGITALES II

INTEGRANTES

MARIO PABÓN

RICARDO BRAVO

PROYECTO

CAR RACING

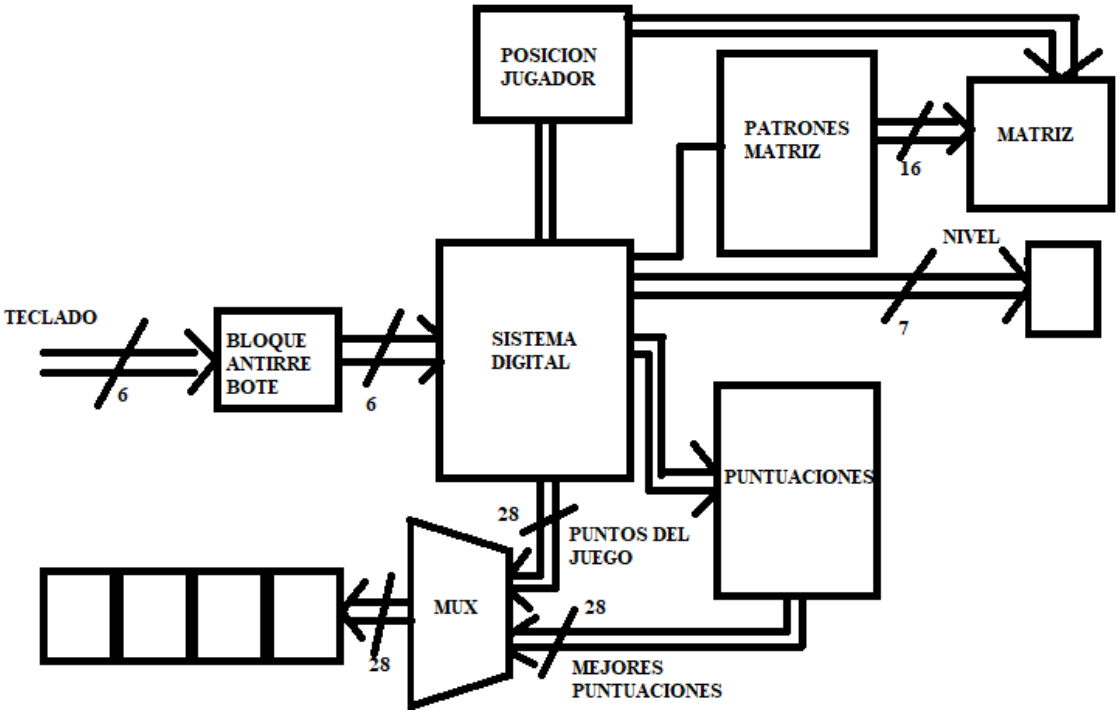
PROFESOR

LEONARDO MUÑOZ

PARALELO 106

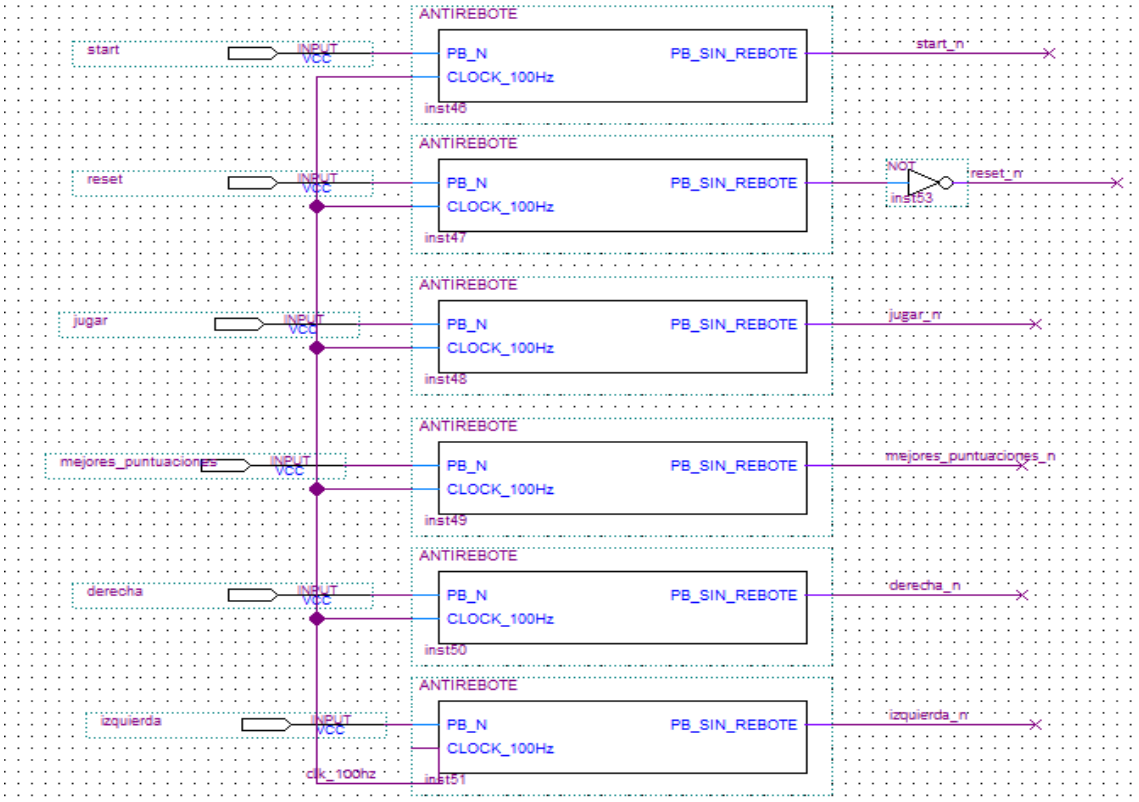
2° TERMINO

I. DIAGRAMA DE BLOQUES

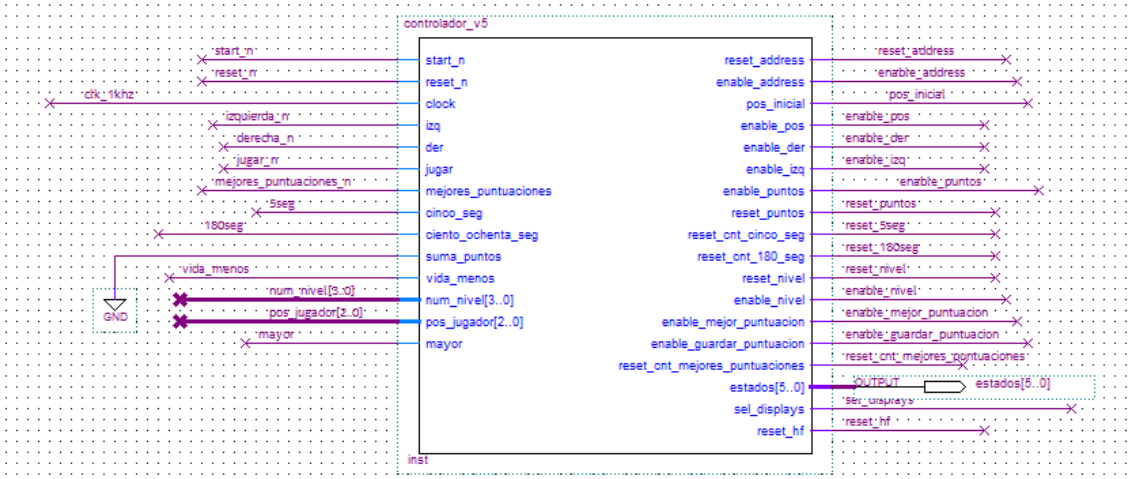


Cuando el usuario usa el teclado, estos pasan a través de un bloque anti-rebote antes de entrar al sistema digital. En el sistema digital, dependiendo de lo que se reciba en la entrada se obtendrá una salida. Cuando se presione el botón START se habilitará el sistema y cuando se presione JUGAR se habilitará el bloque PATRONES MATRIZ, el cual es una RAM que contiene los obstáculos a esquivar, también se habilitará el bloque POSICION JUGADOR el cual es un contador que permite mover nuestro carro de izquierda a derecha sin desbordarse de la pantalla y el otro bloque que se habilita es PUNTUACIONES, el cual es una RAM donde se guardan las mejores puntuaciones del jugador. El bloque de nivel es un display que proyectará el nivel en el que nos encontramos y posee otros 4 displays junto con un multiplexor el cual nos permitirá visualizar los puntos mientras jugamos o cuando se selecciona la opción de MEJORES PUNTUACIONES.

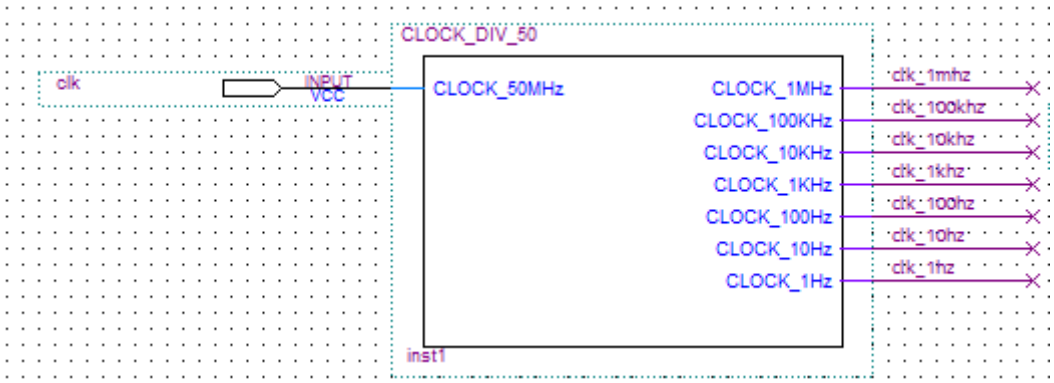
II. PARTICION FUNCIONAL



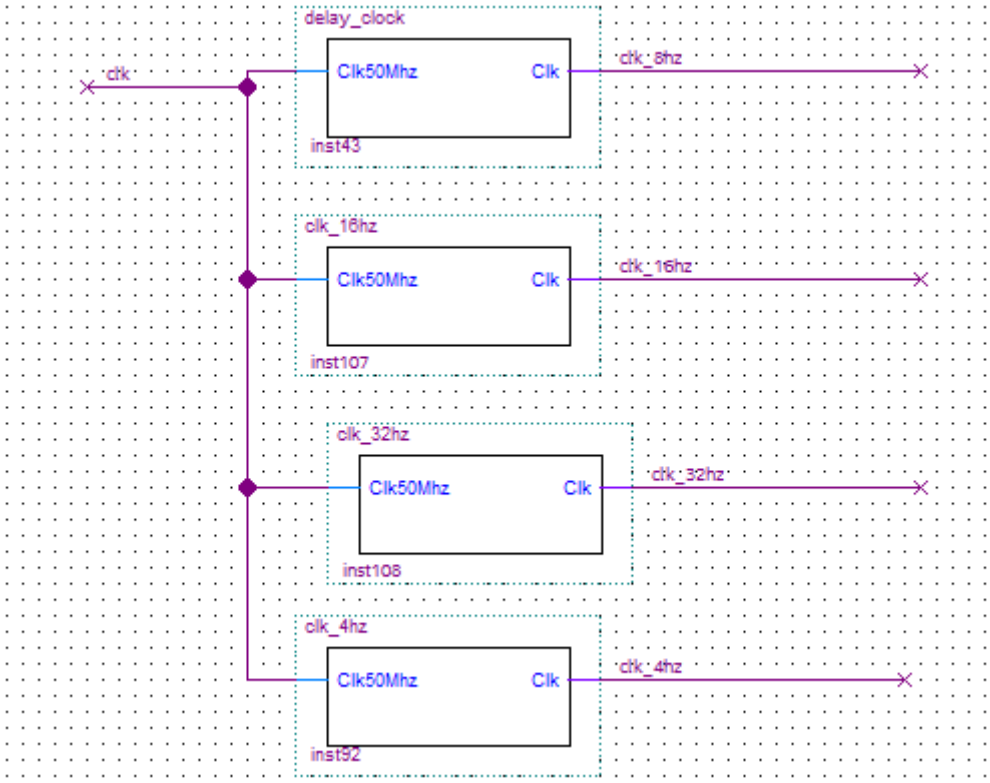
Entradas de botoneras con su respectivo bloque de ANTIREBOTE.



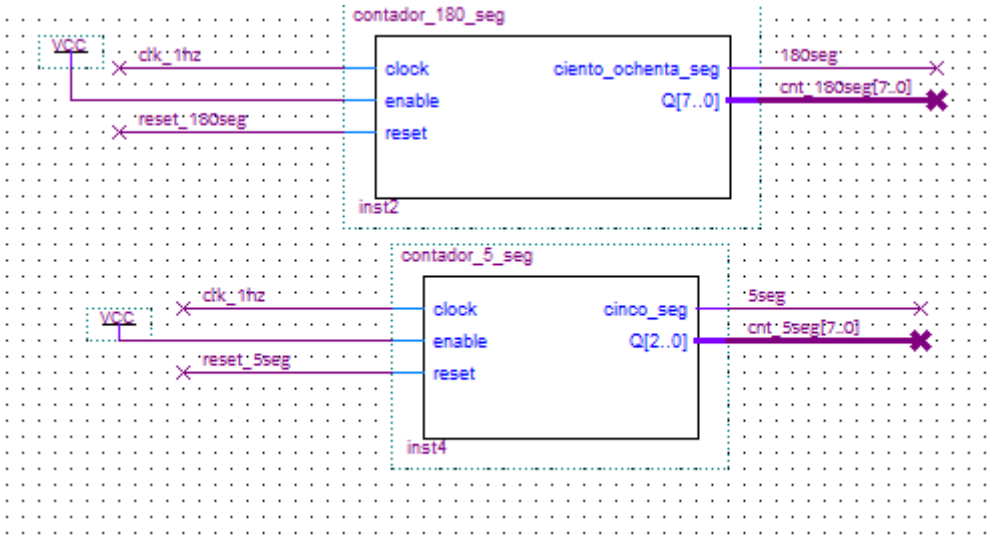
Bloque controlador del sistema con sus respectivas entradas y salidas.



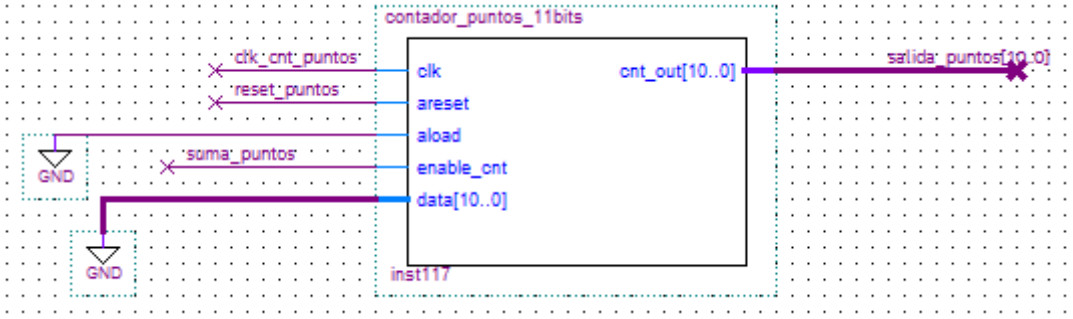
Bloque divisor de clock con una entrada de 50MHz para generar frecuencias de salida de 1MHz, 100KHz, 10KHz, 1KHz, 100Hz, 10Hz y 1Hz.



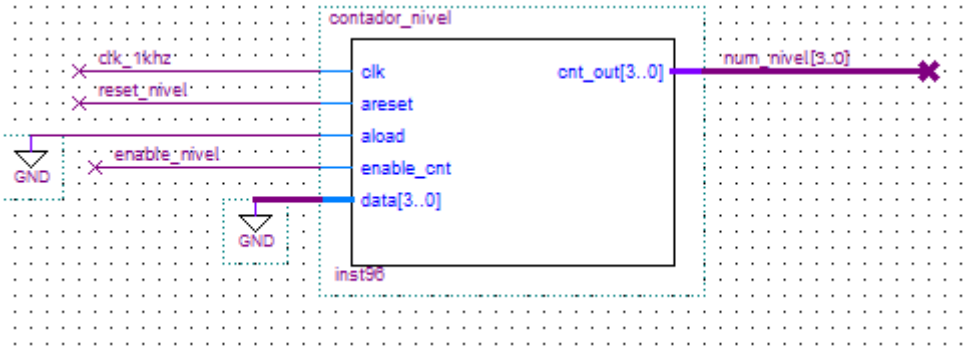
Bloques divisores de frecuencia con entradas de 50 MHz para obtener frecuencias específicas de 4Hz, 8Hz, 16Hz, 32Hz.



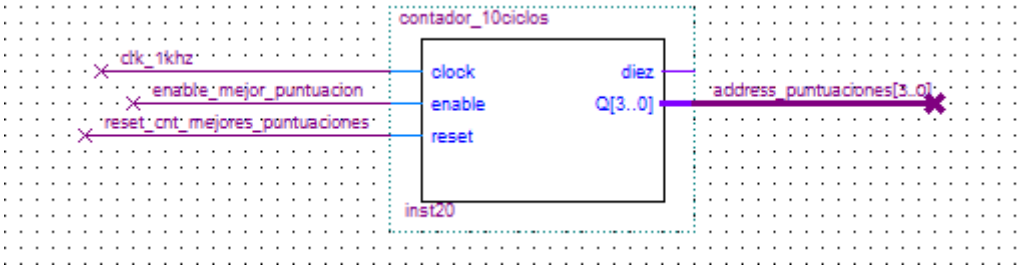
Contadores de 5 segundos y 180 segundos (3 minutos), para conteo de tiempo.



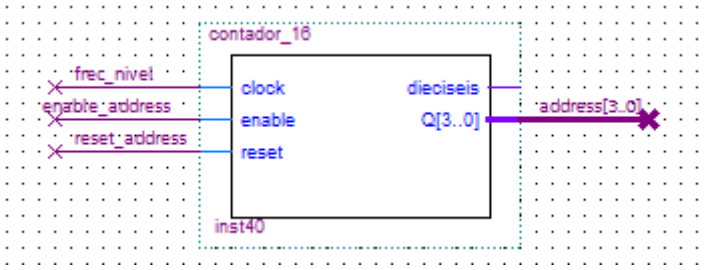
Contador de puntos de 11 bits, con clock multiplexado dependiendo del nivel.



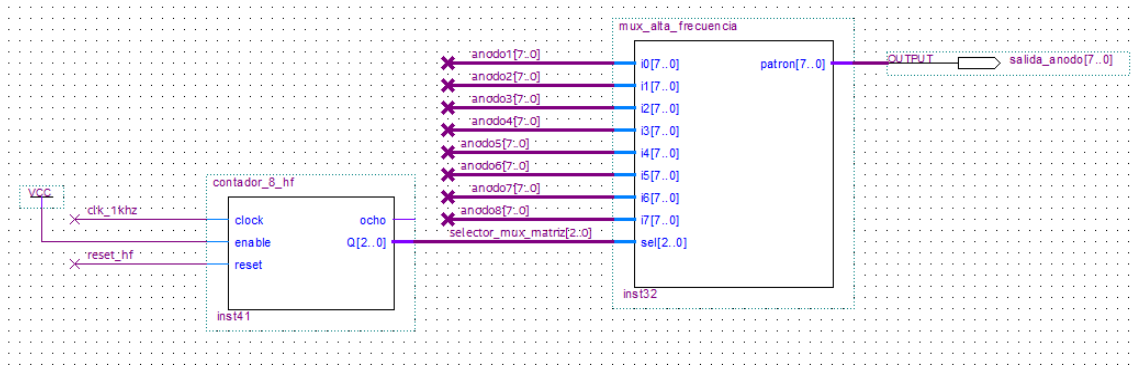
Contador de nivel con clock de 1 KHz, habilitado por el controlador.



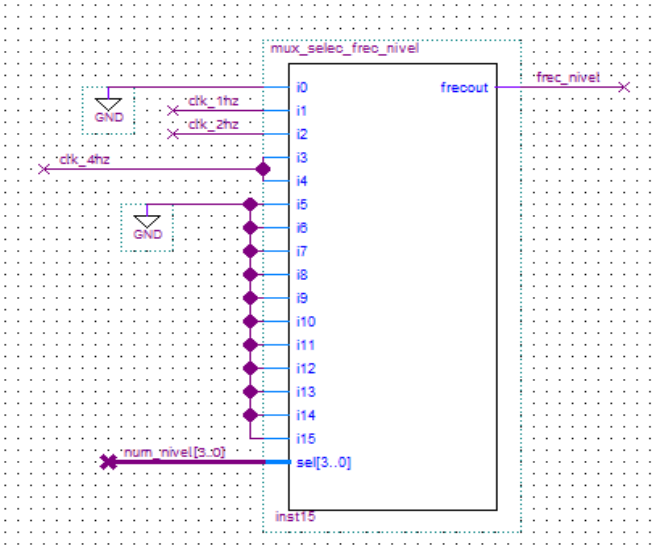
Contador de 10 ciclos para mostrar las 10 mejores puntuaciones



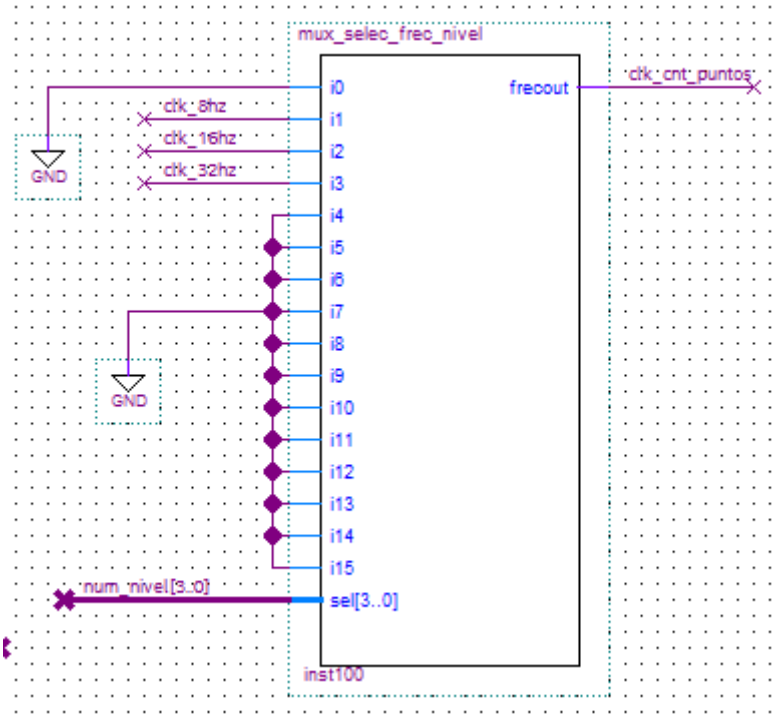
Contador de 16 ciclos para recorrer las 16 direcciones de las memorias RAM de patrones.



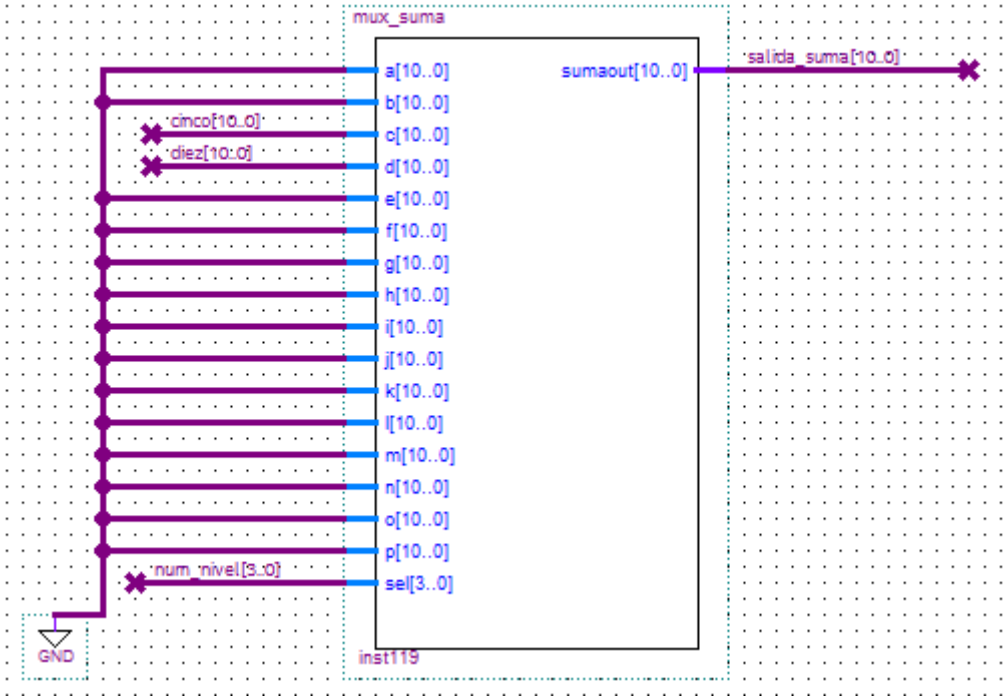
Contador de 8 ciclos a frecuencia de 1KHz para una multiplicación de alta velocidad y poder mostrar los obstáculos y el jugador como si estuvieran prendiendo al mismo tiempo.



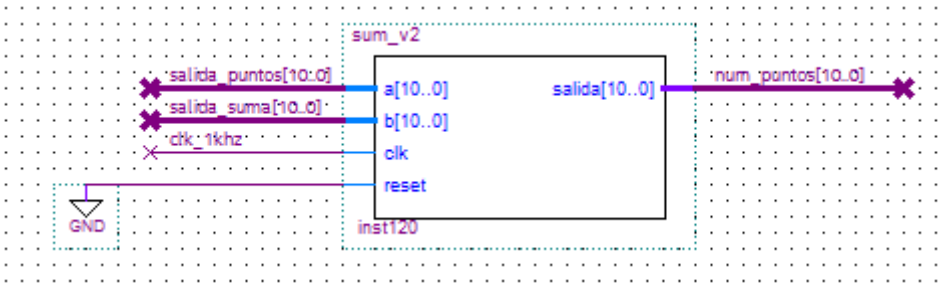
Multiplexor de 16 entradas para multiplexar la frecuencia del nivel.



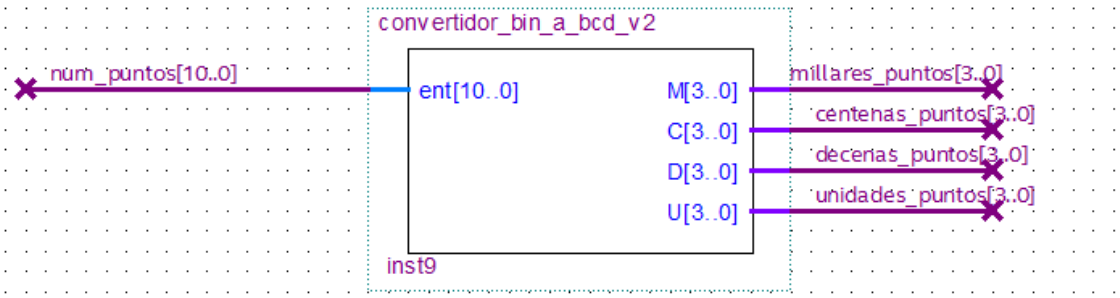
Multiplexor para multiplexar la frecuencia del contador de puntos dependiendo del nivel.



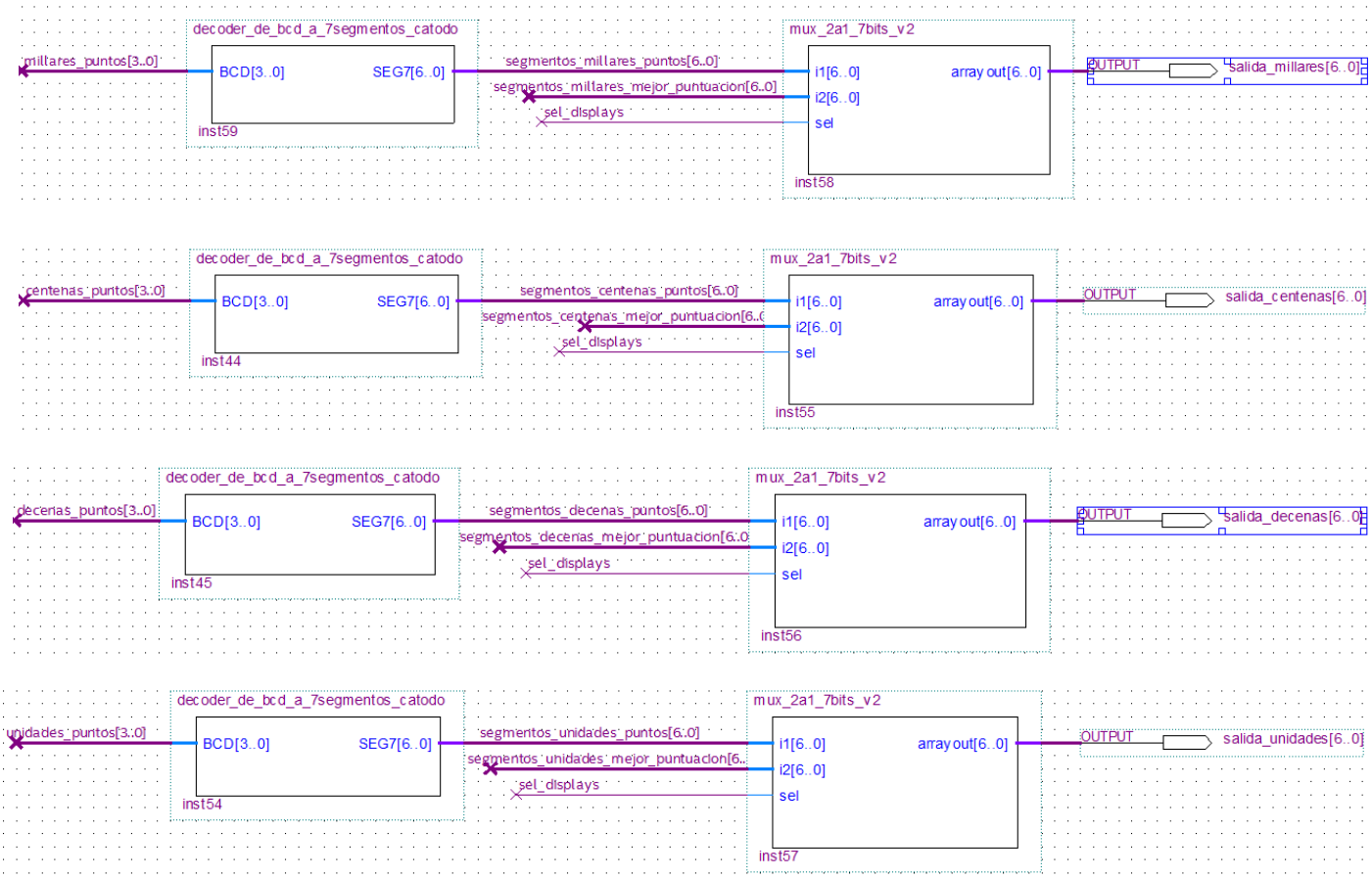
Multiplexor para multiplexar la suma cuando se avanza de nivel.



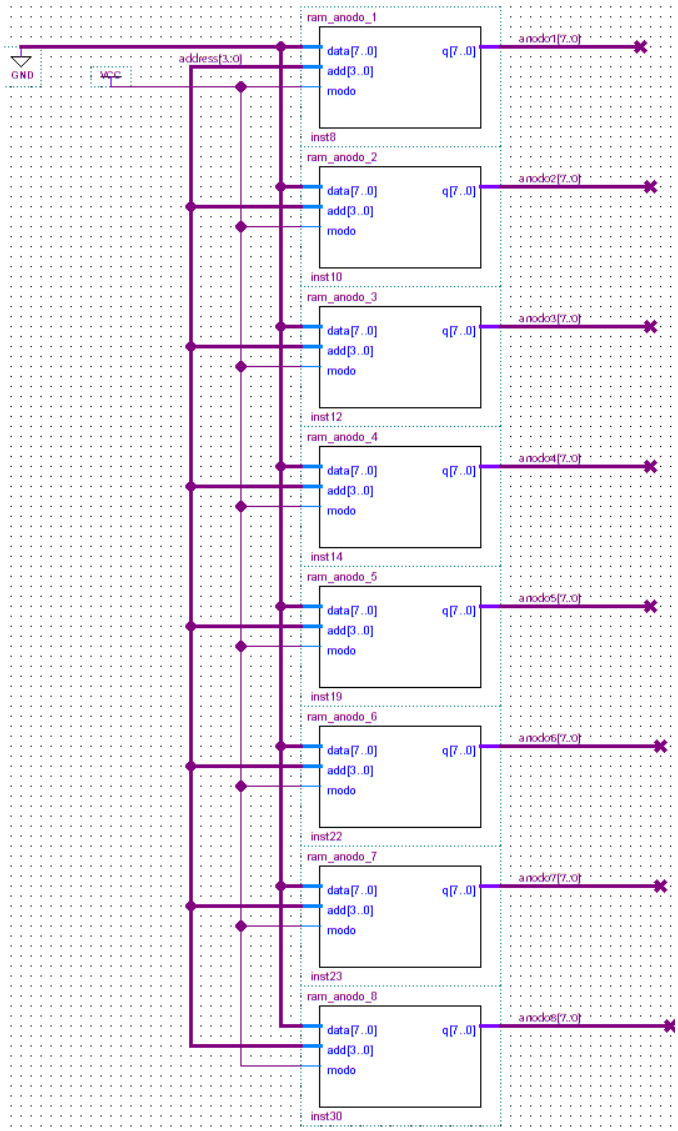
Sumador de 11 bits para sumar los 5 puntos que se obtienen por avanzar de nivel.



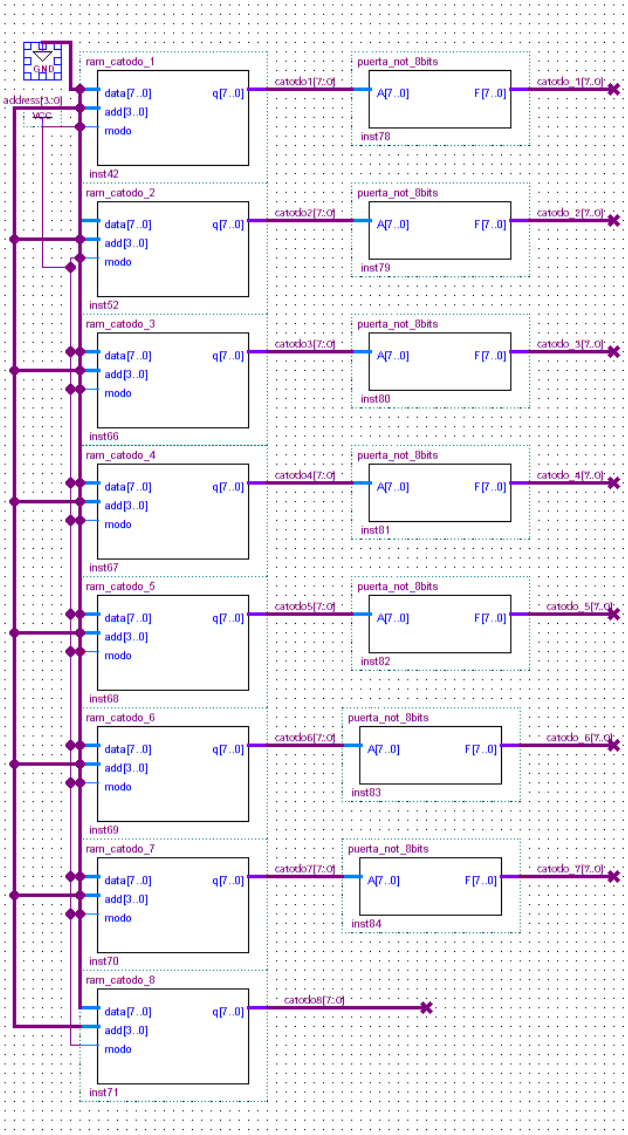
Bloque convertidor de binario a 4 cifras BCD con una entrada de 11 bits.



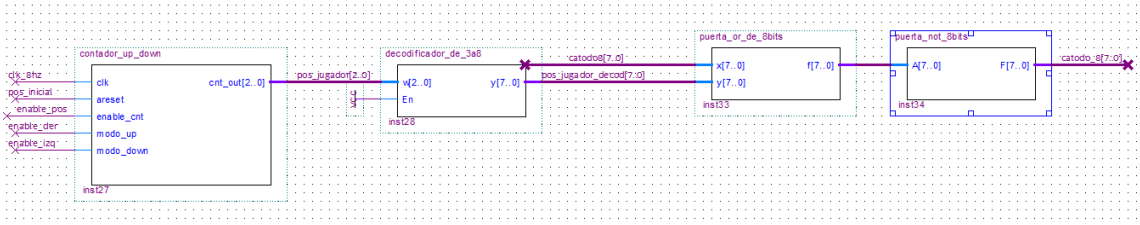
Convertidores BCD a 7 segmentos para displays cátodo común cuya salida será multiplexada para solo ser mostrada durante el juego.



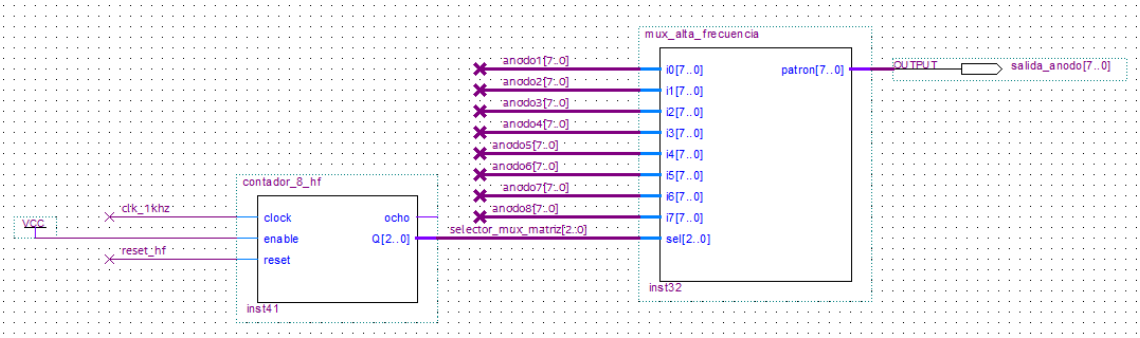
Arreglo de Memorias RAM para mandar nivel alto de voltaje a los ánodos de la matriz.



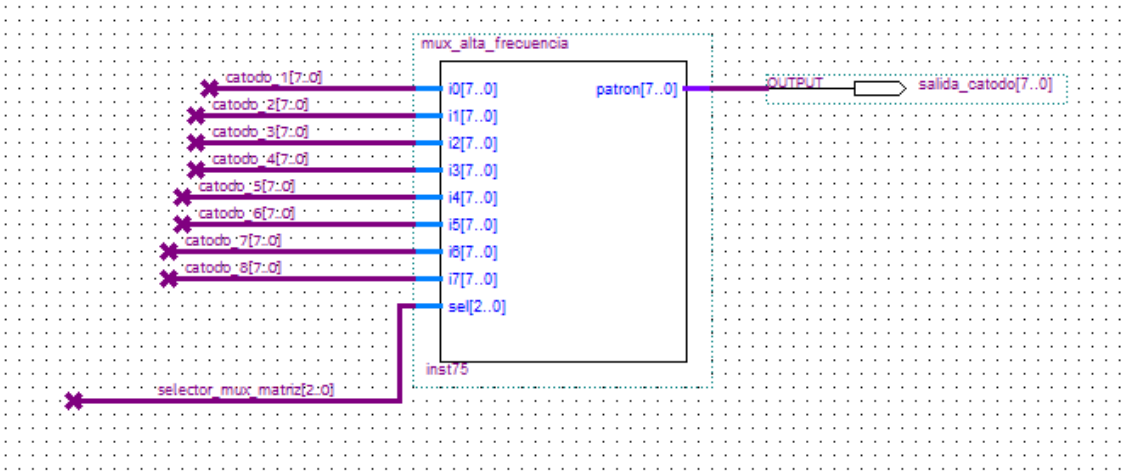
Arreglo de memorias RAM para generar los patrones de cátodo.



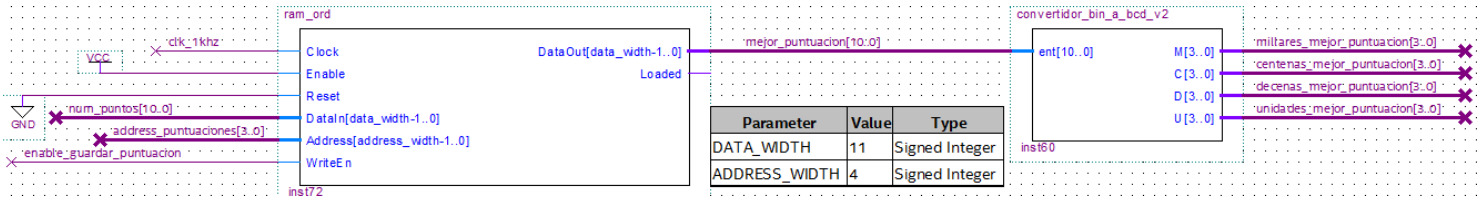
Contador UP/DOWN para contar la posición del jugador, en serie a un decoder de 3 a 8, el cual irá a una puerta or de 8 bits junto con el cátodo 8 y finalmente una puerta not de 8 bits para validar el encendido del jugador como del obstáculo en la última fila.



Contador de 0 a 7 con frecuencia de 1KHz, para multiplexar a alta frecuencia y simular que todos los obstáculos y el jugador están encendidos al mismo tiempo.



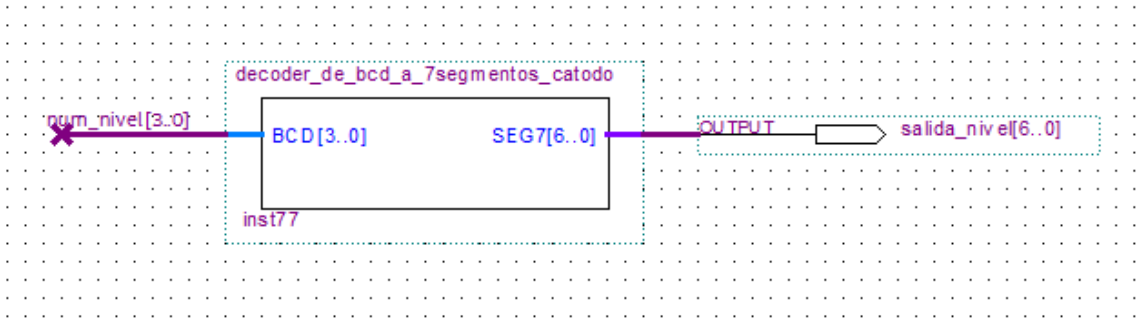
Multiplexor para los cátodos cuyo selector es el número salido del contador de alta frecuencia, para mandar nivel bajo de voltaje a alta velocidad.



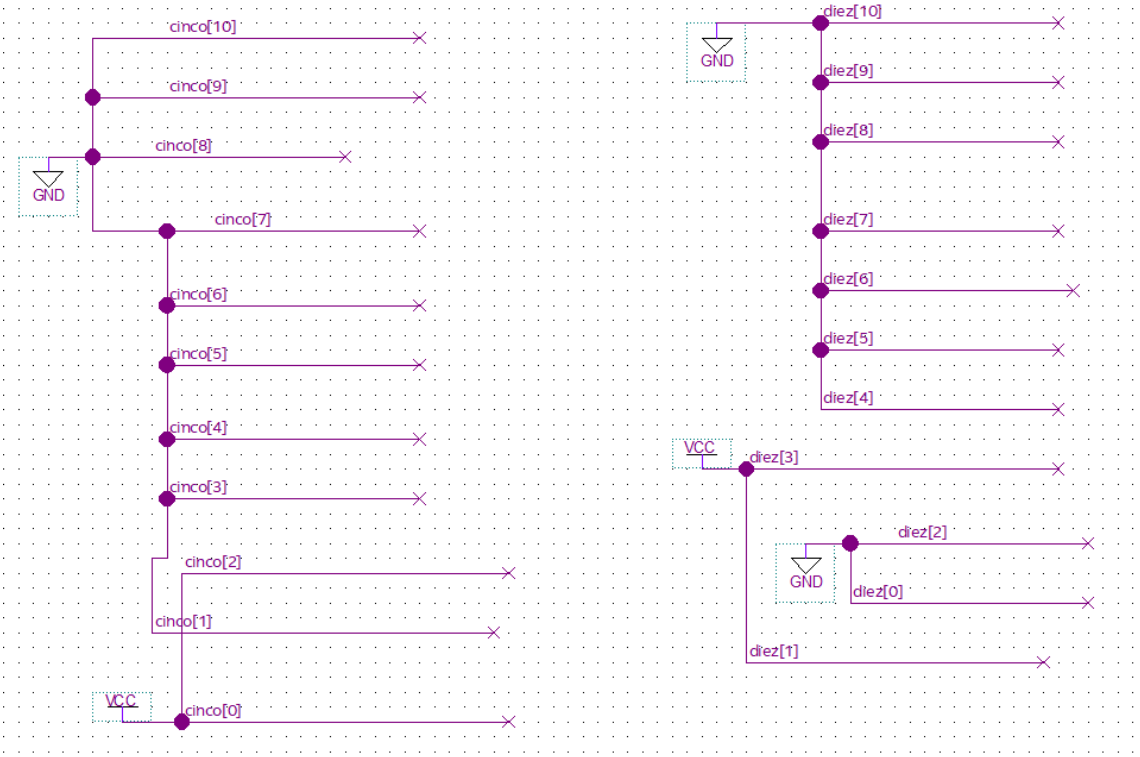
Parameter	Value	Type
DATA_WIDTH	11	Signed Integer
ADDRESS_WIDTH	4	Signed Integer

Memoria RAM para guardar las mejores puntuaciones, en serie al convertidor de binario a BCD.

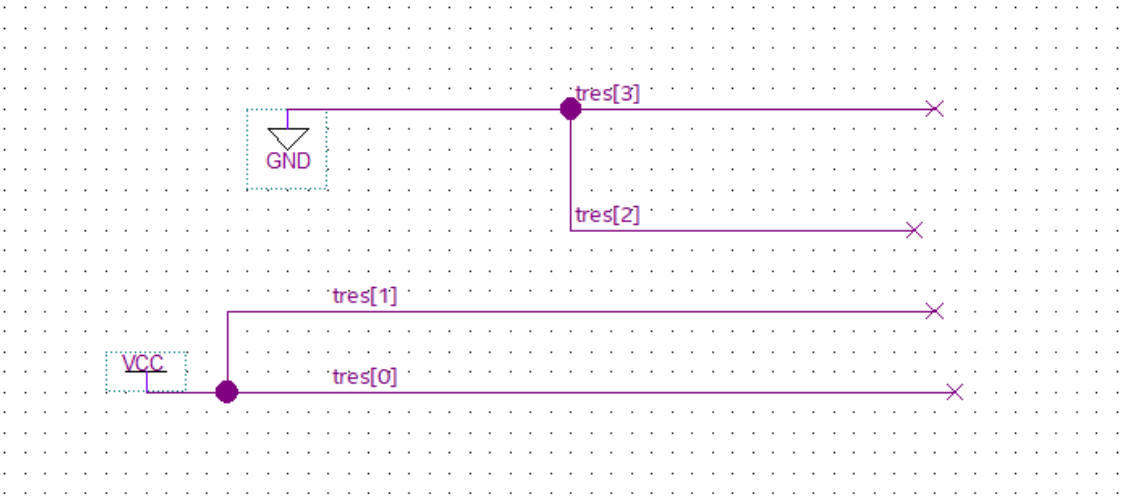
Bloques convertidores de BCD a 7 segmentos para generar las salidas de mejores puntuaciones las cuales serán multiplexadas para ser mandadas a los display de 7 segmentos cuando sea requerido.



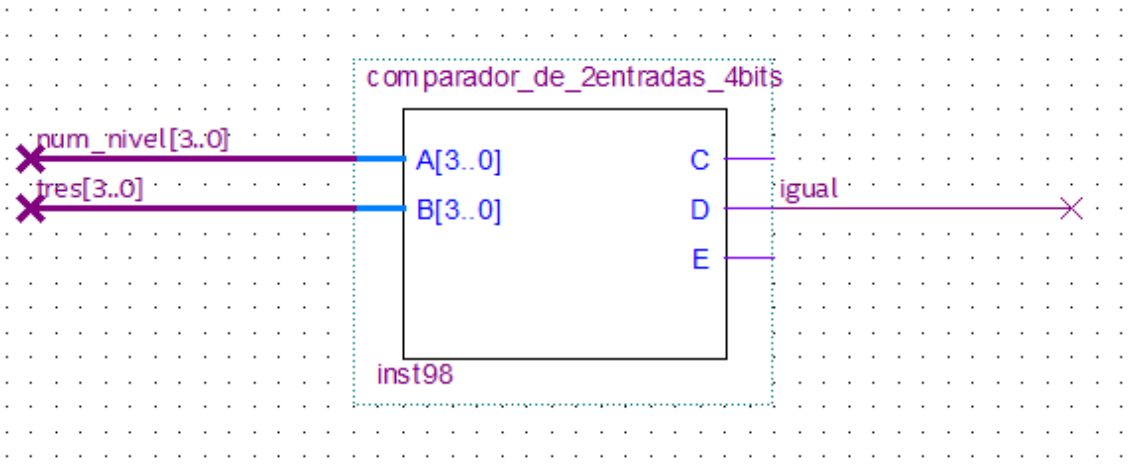
Decodificador de BCD a 7 segmentos para el número de nivel, cuya salida irá directo al display.



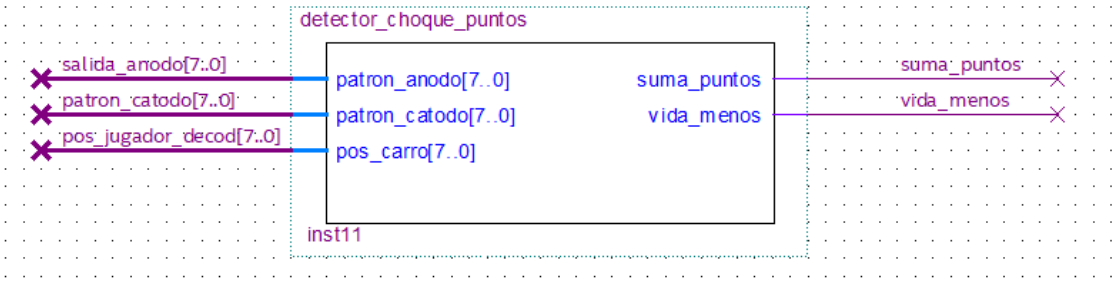
Cables conectados a GND y VCC para generar los números 5 y 10 en binario de 11 bits para sumarlos cuando se suba de nivel en el juego



Cables conectados a GND y VCC para generar el número 3 en binario de 4 bits.

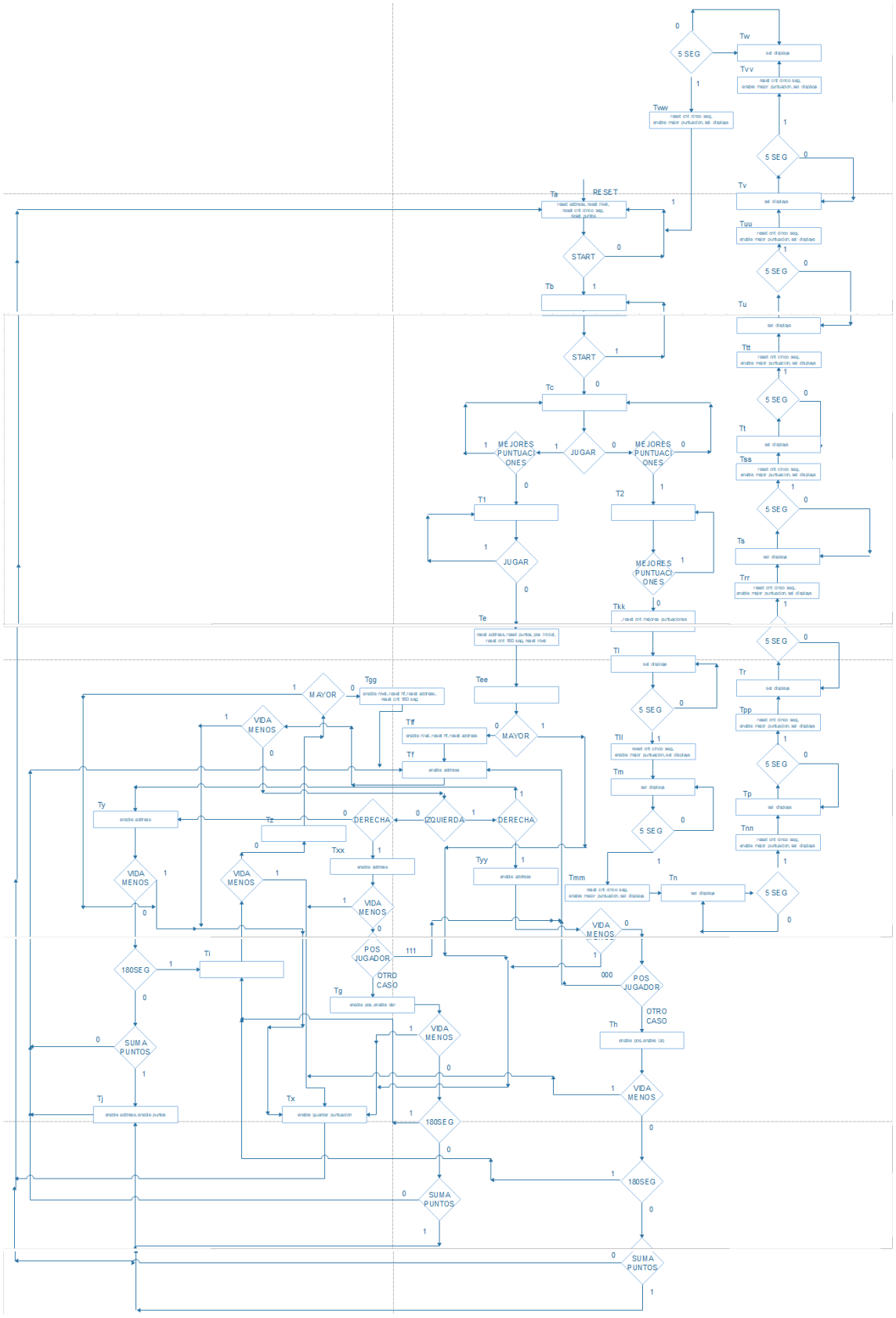


Comparador de 2 números de 4 bits donde se desea la señal igual, se compara el número de nivel con el número 3, en caso de cumplirse se envía la señal al controlador.



Bloque detector de choque y puntos, encargado de generar las señales para habilitar la suma de puntaje al rebasar los obstáculos y también encargado de enviar al controlador la señal para terminar el juego cuando el vehículo ha chocado con algún obstáculo.

III. DIAGRAMA ASM



SEÑALES EXTERNAS DE ENTRADA A LA MSS

Reset: Señal de 1 bit proveniente del usuario para reiniciar el sistema.

Start: Señal de 1 bit proveniente del usuario para dar inicio al sistema.

Jugar: Señal de 1 bit proveniente del usuario para comenzar a jugar.

Mejores_Puntuaciones: Señal de 1 bit proveniente del usuario para visualizar las 10 mejores puntuaciones.

Izq: Señal de 1 bit proveniente del usuario para mover el carro a la izquierda.

Der: Señal de 1 bit proveniente del usuario para mover el carro a la derecha.

SEÑALES INTERNAS DE ENTRADA A LA MSS

Clock: Reloj automático de la MSS.

Cinco_seg: Señal de 1 bit proveniente del contador de 5 segundos para proyectar cada puntuación almacenada durante 5 segundos

Ciento_ochenta_seg: Señal de 1 bit proveniente del contador de 180 segundos para avisar al sistema que ya pasaron 3 minutos y avanzar al siguiente nivel.

Suma_puntos: Señal de 1 bit proveniente del detector de choque para indicar al sistema que se ha superado un obstáculo y se debe sumar puntos.

Vida_menos: Señal de 1 bit proveniente del detector de choque para indicar al sistema que se ha estrellado con un obstáculo y se debe perder vida.

Num_nivel: Señal de 4 bits proveniente del contador de nivel para indicar al sistema el nivel en el que se encuentra para avanzar al siguiente nivel o terminar el juego.

Pos_jugador: Señal de 3 bits proveniente del contador up-down para indicar a la MSS la posición del carro.

Mayor: Señal de 1 bit proveniente del comparador que avisa a la MSS cuando el nivel es mayor a 3.

SEÑALES INTERNAS DE SALIDA DE LA MSS

Reset_address: Señal de 1 bit proveniente de la MSS para reiniciar el contador de direcciones de la RAM que contiene los patrones que se proyectan en la matriz.

Enable_address: Señal de 1 bit proveniente de la MSS para habilitar el contador de direcciones de la RAM que contiene los patrones que se proyectan en la matriz.

Pos_inicial: Señal de 1 bit proveniente de la MSS para reiniciar el contador up-down de movimiento del carro.

Enable_pos: Señal de 1 bit proveniente de la MSS para habilitar el contador up-down de movimiento del carro.

Enable_der: Señal de 1 bit proveniente de la MSS para habilitar el movimiento a la derecha del contador up-down del carro.

Enable_izq: Señal de 1 bit proveniente de la MSS para habilitar el movimiento a la izquierda del contador up-down del carro.

Enable_puntos: Señal de 1 bit proveniente de la MSS para habilitar el contador de puntos.

Reset_puntos: Señal de 1 bit proveniente de la MSS para reiniciar el contador de puntos.

Reset_cnt_cinco_seg: Señal de 1 bit proveniente de la MSS para reiniciar el contador de 5 segundos.

Reset_nivel: Señal de 1 bit proveniente de la MSS para reiniciar el contador de nivel.

Enable_nivel: Señal de 1 bit proveniente de la MSS para habilitar el contador de nivel.

Enable_mejor_puntuacion: Señal de 1 bit proveniente de la MSS para habilitar el contador de la RAM para visualizar las mejores puntuaciones.

Enable_guardar_puntuacion: Señal de 1 bit proveniente de la MSS para habilitar el modo de escritura de la RAM donde se guardan las puntuaciones.

Reset_cnt_mejores_puntuaciones: Señal de 1 bit proveniente de la MSS para habilitar reiniciar el contador de la RAM donde se guardan las puntuaciones.

Sel_displays: Señal de 1 bit proveniente de la MSS que se usa como selector del multiplexor que mostrará las puntuaciones actuales al jugar o las puntuaciones almacenadas en la RAM.

Reset_hf: Señal de 1 bit proveniente de la MSS que reinicia el contador para el selector de alta frecuencia.

Estados: Señal de 6 bits proveniente de la MSS que muestra los cambios de estado del sistema.

IV. CODIGO VHDL DE LA MSS

```
library ieee;
use ieee.std_logic_1164.all;
entity controlador_v5 is
    port( start_n, reset_n, clock, izq, der, jugar, mejores_puntuaciones: in std_logic;
          cinco_seg, ciento_ochenta_seg, suma_puntos, vida_menos: in std_logic;
          num_nivel: in std_logic_vector(3 downto 0);
          pos_jugador: in std_logic_vector(2 downto 0);
          mayor: in std_logic;
          reset_address, enable_address, pos_inicial, enable_pos, enable_der, enable_izq: out std_logic;
          enable_puntos, reset_puntos, reset_cnt_cinco_seg, reset_cnt_180_seg, reset_nivel, enable_nivel, enable_mejor_puntuacion,enable_guardar_puntuacion, reset_cnt_mejores_puntuaciones: out std_logic;
          estados: out std_logic_vector (5 downto 0);
          sel_displays, reset_hf: out std_logic);
end controlador_v5;

architecture mss of controlador_v5 is
    type estado is (Ta, Tb, T1, T2, Tc, Te, Tee, Tf, Tff, Tg, Tgg, Th, Ti, Tj, Tkk, Tl, Tll, Tm, Tmm, Tn, Tnn, Tp, Tpp, Tr, Trr, Ts, Tss, Tt, Ttt, Tu, Tuu, Tv, Tvv, Tw, Tww, Tx, Txx, Ty, Tyy, Tz);
    signal y: estado;
begin
    process(reset_n, clock, vida_menos)
    begin
    if reset_n = '1' then y <= Ta;
    elsif (clock'event and clock = '1') then
        case y is
            when Ta => if start_n = '1' then y <= Tb; else y <= Ta; end if;
            when Tb => if start_n = '0' then y <= Tc; else y <= Tb; end if;
            when Tc => if jugar = '0' and mejores_puntuaciones = '0' then y <= Tc;
                       elsif jugar = '0' and mejores_puntuaciones = '1' then y <= T2;
                       elsif jugar = '1' and mejores_puntuaciones = '0' then y <= T1;
                       else y <= Tc; end if;
            when T1 => if jugar = '0' then y <= Te; else y <= T1; end if;
            when T2 => if mejores_puntuaciones = '0' then y <= Tkk; else y <= T2; end if;
            when Te => y <= Tee;
            when Tee => if mayor = '1' then y <= Tx;
                        else y <= Tff; end if;
            when Tff => if vida_menos = '1' then y <= Tx;
                        elsif izq = '0' and der = '0' then y <= Ty;
                        elsif izq = '0' and der = '1' then y <= Txx;
                        elsif izq = '1' and der = '0' then y <= Tyy;
                        else y <= Ty; end if;
            when Tff => y <= Tf;
            when Tg => if vida_menos = '1' then y <= Tx;
                       elsif vida_menos = '0' and ciento_ochenta_seg = '0' and suma_puntos = '0' then y <= Tf;
                       elsif vida_menos = '0' and ciento_ochenta_seg = '0' and suma_puntos = '1' then y <= Tf;
                       elsif vida_menos = '0' and ciento_ochenta_seg = '1' and suma_puntos = '0' then y <= T1;
                       elsif vida_menos = '0' and ciento_ochenta_seg = '1' and suma_puntos = '1' then y <= T1;
                       else y <= Tx; end if;
            when Tgg => y <= Tf;
            when Th => if vida_menos = '1' then y <= Tx;
                       elsif vida_menos = '0' and ciento_ochenta_seg = '0' and suma_puntos = '0' then y <= Tf;
                       elsif vida_menos = '0' and ciento_ochenta_seg = '0' and suma_puntos = '1' then y <= Tf;
                       else y <= Tx; end if;
            when Ti => if vida_menos = '1' then y <= Tx;
                       else y <= Tz; end if;
            when Tj => y <= Tf;
            when Tkk => y <= Tl;
            when Tl => if cinco_seg = '1' then y <= Tll; else y <= Tl; end if;
            when Tll => y <= Tm;
            when Tm => if cinco_seg = '1' then y <= Tmm; else y <= Tm; end if;
            when Tmm => y <= Tn;
            when Tn => if cinco_seg = '1' then y <= Tnn; else y <= Tn; end if;
            when Tnn=> y <= Tp;
            when Tp => if cinco_seg = '1' then y <= Tpp; else y <= Tp; end if;
            when Tpp => y <= Tr;
            when Tr => if cinco_seg = '1' then y <= Trr; else y <= Tr; end if;
            when Trr => y <= Ts;
            when Ts => if cinco_seg = '1' then y <= Tss; else y <= Ts; end if;
            when Tss => y <= Tt;
            when Tt => if cinco_seg = '1' then y <= Ttt; else y <= Tt; end if;
            when Ttt => y <= Tu;
            when Tu => if cinco_seg = '1' then y <= Tuu; else y <= Tu; end if;
            when Tuu => y <= Tv;
            when Tv => if cinco_seg = '1' then y <= Tvv; else y <= Tv; end if;
            when Tvv => y <= Tw;
            when Tw => if cinco_seg = '1' then y <= Tww; else y <= Tw; end if;
            when Tww => y <= Ta;
            when Tx => y <= Ta;
            when Ty => if vida_menos = '1' then y <= Tx;
                       elsif vida_menos = '0' and ciento_ochenta_seg = '0' and suma_puntos = '0' then y <= Tf;
                       elsif vida_menos = '0' and ciento_ochenta_seg = '0' and suma_puntos = '1' then y <= Tf;
                       elsif vida_menos = '0' and ciento_ochenta_seg = '1' and suma_puntos = '0' then y <= Ti;
                       elsif vida_menos = '0' and ciento_ochenta_seg = '1' and suma_puntos = '1' then y <= Ti;
                       else y <= Tx; end if;
            when Txx => if vida_menos = '1' then y <= Tx;
                        elsif pos_jugador /= "111" then y <= Tg;
                        else y <= Tf; end if;
            when Tyy => if vida_menos = '1' then y <= Tx;
                        elsif pos_jugador /= "000" then y <= Th;
                        else y <= Tf; end if;
            when Tz => if mayor = '1' then y <= Tx;
                       else y <= Tgg; end if;
        end case;
    end if;
        when Ta => estados <= "000000";
        when Tb => estados <= "000000";
        when Tc => estados <= "000010";
        when T1 => estados <= "000000";
        when T2 => estados <= "000000";
        when Te => estados <= "000100";
                   reset_address <= '1';
                   reset_puntos <= '1';
                   pos_inicial <= '1';
                   reset_cnt_180_seg <= '1';
                   reset_nivel <= '1';
        when Tee => estados <= "000000";
        when Tf => estados <= "000101"; enable_address <= '1';
        when Tff => enable_nivel <= '1'; reset_hf <= '1'; reset_address <= '1';
        when Tg => estados <= "000110"; enable_pos <= '1'; enable_der <= '1';
                   if suma_puntos = '1' then enable_puntos <= '1'; end if;
        when Tgg => enable_nivel <= '1'; reset_hf <= '1'; reset_address <= '1'; reset_cnt_180_seg <= '1';
        when Th => estados <= "000111"; enable_pos <= '1'; enable_izq <= '1';
                   if suma_puntos = '1' then enable_puntos <= '1'; end if;
        when Ti => estados <= "001000";
        when Tj => estados <= "001001"; enable_puntos <= '1'; enable_address <= '1';
        when Tkk => estados <= "001011"; reset_cnt_mejores_puntuaciones <= '1';
        when Tl => estados <= "001100"; sel_displays <= '1';
        when Tll => estados <= "001101"; reset_cnt_cinco_seg <= '1'; enable_mejor_puntuacion <= '1'; sel_displays <= '1';
        when Tm => estados <= "001110"; sel_displays <= '1';
        when Tmm => estados <= "001111"; reset_cnt_cinco_seg <= '1'; enable_mejor_puntuacion <= '1'; sel_displays <= '1';
        when Tn => estados <= "010000"; sel_displays <= '1';
        when Tnn=> estados <= "010001"; reset_cnt_cinco_seg <= '1'; enable_mejor_puntuacion <= '1'; sel_displays <= '1';
        when Tp => estados <= "010010"; sel_displays <= '1';
        when Tpp => estados <= "010011"; reset_cnt_cinco_seg <= '1'; enable_mejor_puntuacion <= '1'; sel_displays <= '1';
        when Tr => estados <= "010100"; sel_displays <= '1';
        when Trr => estados <= "010101"; reset_cnt_cinco_seg <= '1'; enable_mejor_puntuacion <= '1'; sel_displays <= '1';
        when Ts => estados<= "010110"; sel_displays <= '1';
        when Tss => estados <= "010111"; reset_cnt_cinco_seg <= '1'; enable_mejor_puntuacion <= '1'; sel_displays <= '1';
        when Tt => estados <= "011000"; sel_displays <= '1';
        when Ttt => estados <= "011001"; reset_cnt_cinco_seg <= '1'; enable_mejor_puntuacion <= '1'; sel_displays <= '1';
        when Tu => estados <= "011010"; sel_displays <= '1';
        when Tuu => estados <= "011011"; reset_cnt_cinco_seg <= '1'; enable_mejor_puntuacion <= '1'; sel_displays <= '1';
        when Tv => estados <= "011100"; sel_displays <= '1';
        when Tvv => estados <= "011101"; reset_cnt_cinco_seg <= '1'; enable_mejor_puntuacion <= '1'; sel_displays <= '1';
        when Tw => estados <= "011110"; sel_displays <= '1';
        when Tww => estados <= "011111"; reset_cnt_cinco_seg <= '1'; enable_mejor_puntuacion <= '1'; sel_displays <= '1';
        when Tx => estados <= "100000"; enable_guardar_puntuacion <= '1';
        when Txx => estados <= "000000"; enable_address <= '1';
        when Ty => estados <= "100001"; enable_address <= '1';
                   if suma_puntos = '1' then enable_puntos <= '1'; end if;
        when Tyy => estados <= "000000"; enable_address <= '1';
        when Tz => estados <= "000000";
    end case;
    end process;
end mss;
```

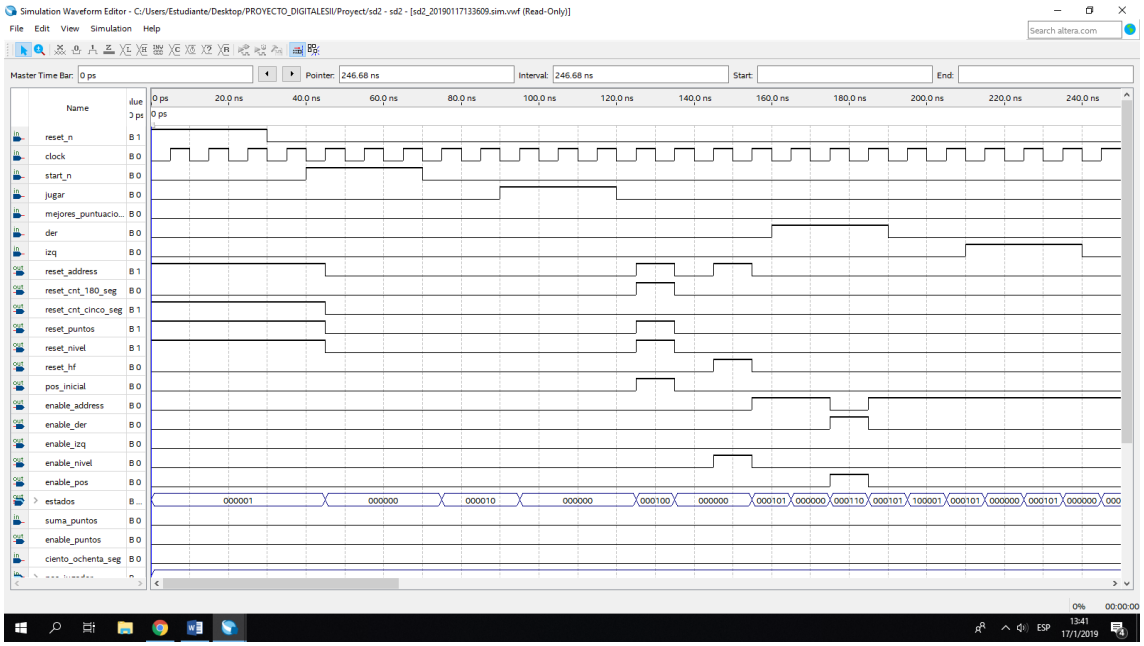
En el Process de Estados, en el estado inicial (Ta) donde se enceran contadores de direcciones y de puntos. En (Ta) se pregunta por la botonera de Start, donde si es presionada pasa a (Tb) donde debe soltarse para pasar a (Tc), caso de no soltar la botonera se queda en (Tb). En el estado (Tc) se pregunta por las siguientes botoneras: jugar y mejores puntuaciones. Donde en caso de presionar alguna de ellas me mandará a los estados que validan el hecho de soltar la botonera presionada (T1 para jugar y T2 para mejores puntuaciones). En los estados (Tb, Tc, T1, T2) solo se generan salidas de estados. En caso de haber presionado y soltado la botonera de jugar, esta mandará al estado (Te) que reinicia nuevamente los contadores de puntos, direcciones y adicionalmente reinicia el contador de 180 segundos. Del estado (Te) se pasa directamente al estado (Tee) donde se pregunta si el número de nivel es igual a 3 para finalizar el juego, en caso de no cumplirse pasa a (Tff), y si es igual a 3 pasa a Tx para terminar el juego. En el estado (Tff), se habilita el primer nivel, se resetea el contador de alta frecuencia y el contador de direcciones de los patrones de ánodo y cátodo para la matriz. De (Tff) se pasa directamente a (Tf). En el estado (Tf) caen los obstáculos y se pregunta primeramente por la señal vida_menos donde si es igual a '1' me manda al estado (Tx) para terminar el juego. En caso de que la señal vida_menos = '0', pregunta por las botoneras de izquierda y derecha, donde si estas no son presionadas o se presionan ambas a la vez el controlador me envía al estado (Ty), en caso de presionar solo derecha me envía a (Txx), y en caso de presionar solo izquierda me manda al estado (Tyy).

Al estar en el estado (Txx) me pregunta primeramente por vida_menos, en caso de no cumplirse, el controlador valida que si la pos_jugador es distinta de "111" habilita el movimiento hacia la derecha y luego mande al estado (Tg) caso contrario me envie al estado (Tf) y continuar con el juego. Al estar en el estado (Tyy) se pregunta primeramente por la señal vida_menos, en caso de no cumplirse el controlador valida que si pos_jugador es distinta de "000" habilite el movimiento hacia a la izquierda y luego mande al estado (Th) caso contrario me envie al estado (Tf) y continuar con el juego. En los estados (Ty, Tg y Th) se pregunta por las señales vida_menos, suma_puntos, ciento_ochenta_seg.

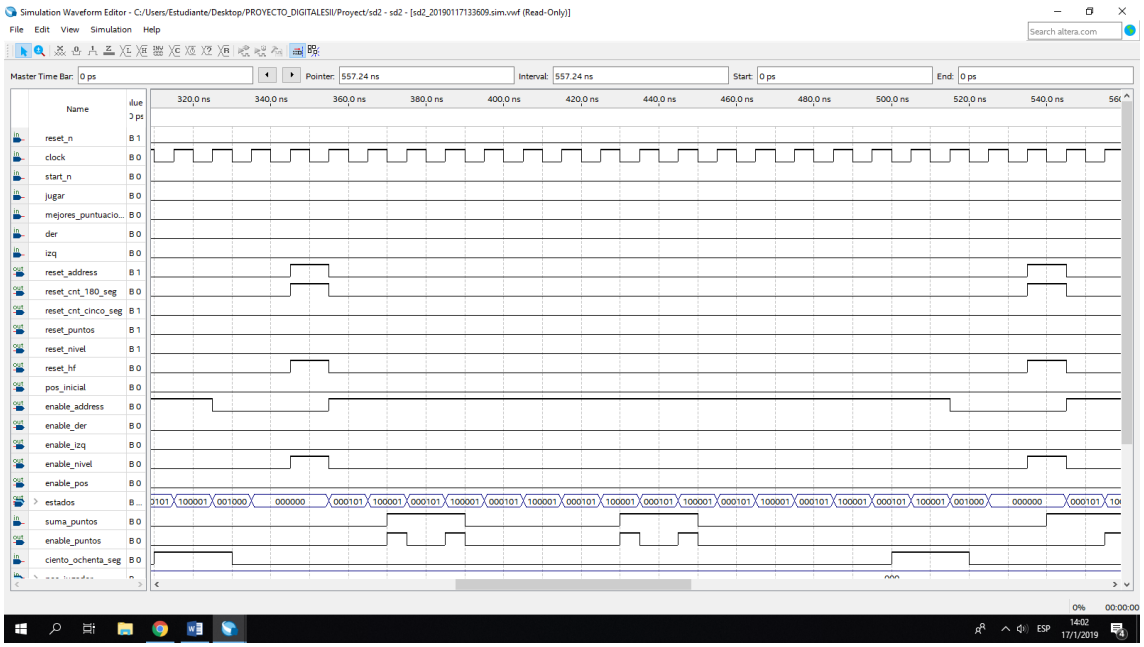
Cuando se genera la señal de ciento_ochenta segundos, pasa al estado (Ti) donde pregunta si el nivel es igual a 3, para cambiar al estado (Tz), luego pasa a (Tgg) donde se habilita el contador para avanzar de nivel, se encera el contador de alta frecuencia, se encera el contador de direcciones de la RAM de ánodos y cátodos; y se encera el contador de 180 segundos. Cuando se hayan superado los 3 niveles del juego, este mandará al estado (Tx) donde se habilita el guardado de puntuación en la memoria RAM y me manda al estado inicial.

En caso de haber presionado y soltado mejores puntuaciones, la MSS me enviará al estado (Tkk) donde se resetea el contador de la RAM de puntuaciones, luego pasará al estado (Tl) donde se cambia el valor de sel_displays a '1' para mostrar las mejores puntuaciones de la RAM de puntuaciones en los displays. Donde cada vez que se muestra un puntaje, se espera la señal de 5 segundos para cambiar la dirección y resetear el contador de 5 segundos. Esto sucede en los estados (Tl, Tll, Tm, Tmm, Tn, Tnn, Tp, Tpp, Tr, Trr, Ts, Tss, Tt, Ttt, Tu, Tuu, Tv, Tvv, Tw y Tww).

V. DIAGRAMA DE TIEMPO (WVF) DE LA MSS

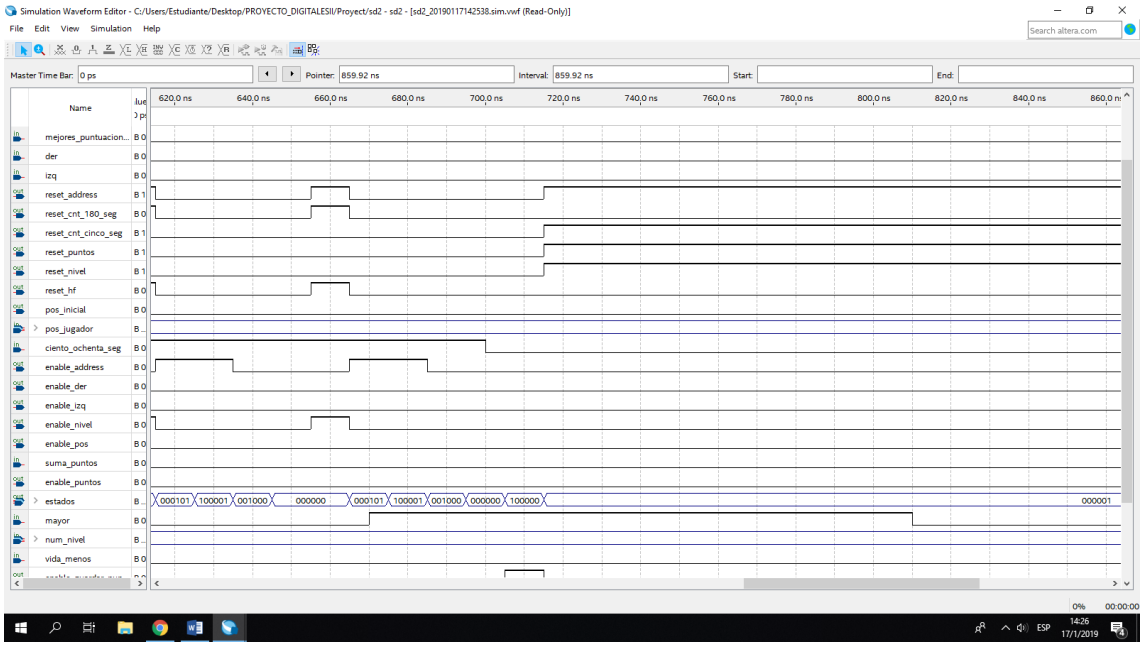


Primero se procedió a resetear la MSS para comenzar en el estado inicial. Luego se presiono y solto Start, y después se presiono y soltó jugar. En el momento que se suelta jugar se habilita reset_address, reset_cnt_180_seg, reset_cnt_cinco_seg, reset_nivel, pos_incial. Luego se activa la señal enable address para habilitar la caída de los obstáculos, se aplastó la botonera derecha lo cual activo la señal enable_pos y enable_der a la vez.

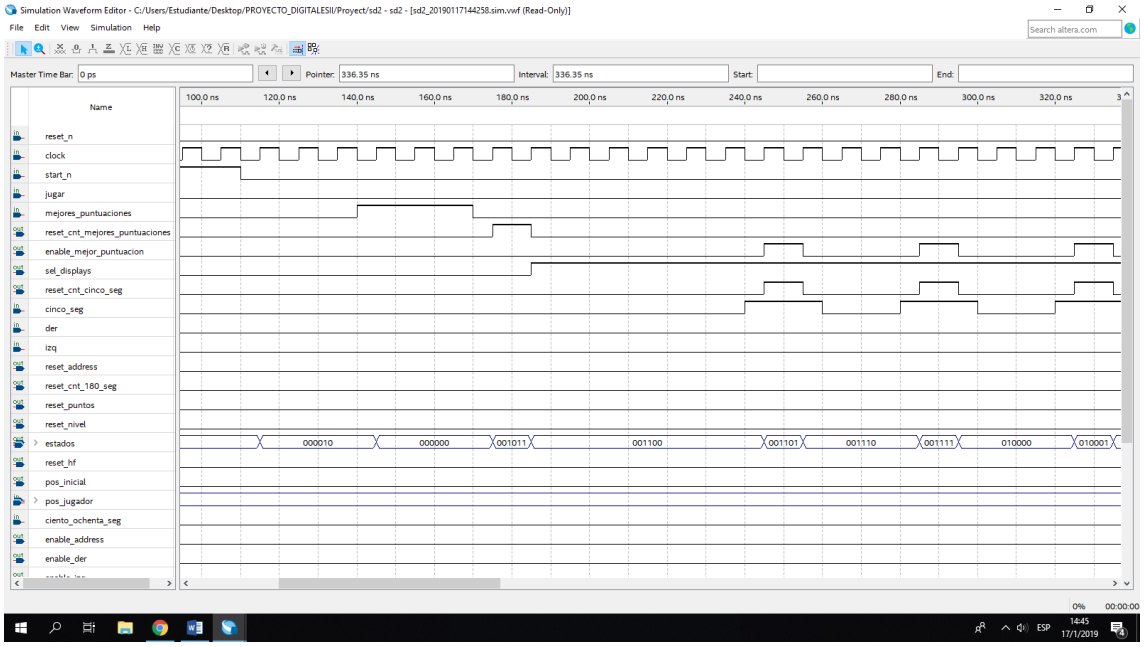


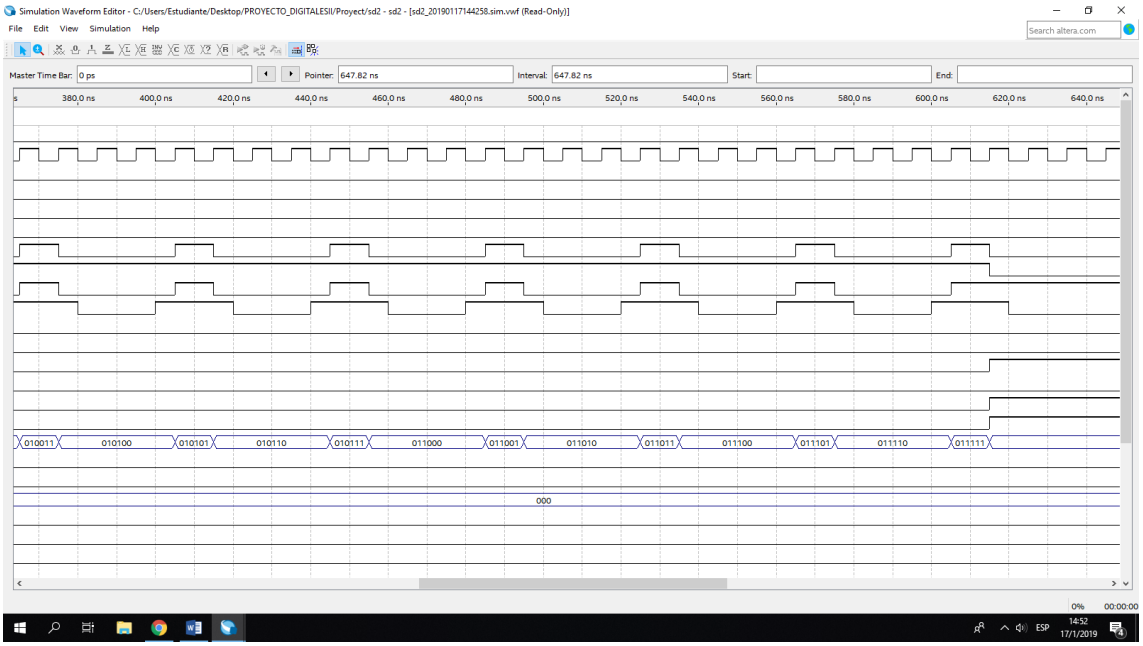
En esta parte del diagrama de tiempo, llega la señal de ciento_ochenta_seg, lo cual produce el cambio de nivel, evidenciado en la activación de la señal enable_nivel, reset_hf, reset_address y reset_cnt_180_seg. Luego se mandó la señal suma_puntos la cual cada flanco de reloj activa la señal enable_puntos que será enviada de la MSS al habilitador del contador de puntos. Luego se volvió a activar la señal ciento_ochenta_seg, produciendo el reseteo de los contadores de las

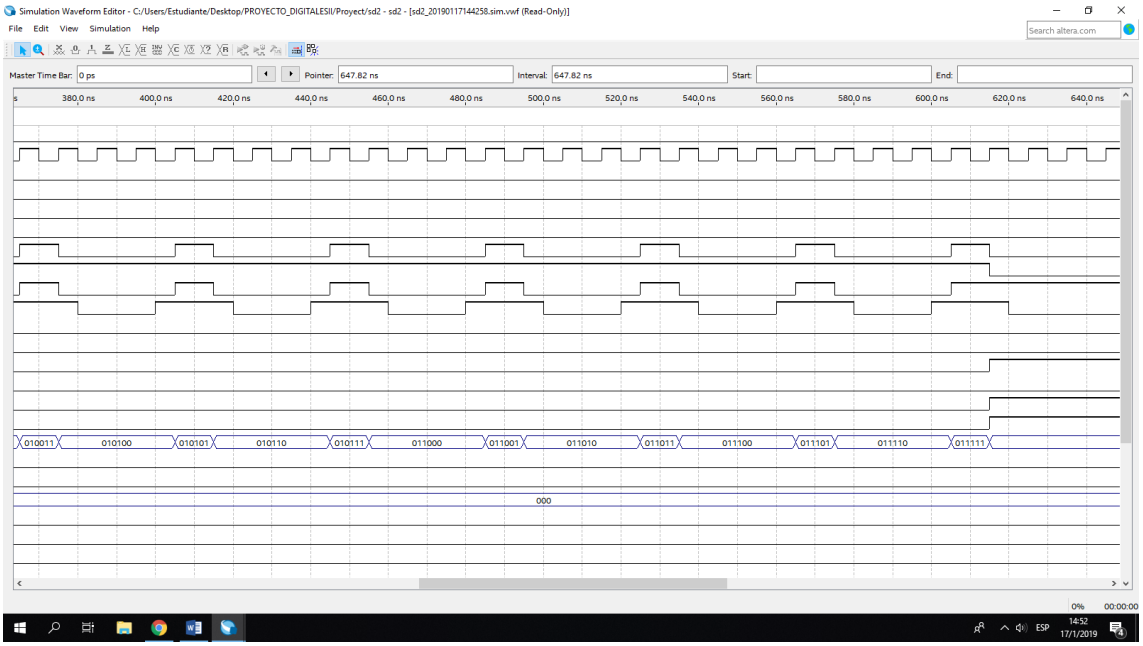
direcciones de las memorias de ánodo y cátodo, del contador de alta frecuencia y el contador de 180 segundos.



En esta parte del diagrama de tiempo, se ve que cuando se activa ciento_ochenta_seg y la señal mayor, manda al estado inicial de activación debido a que el nivel es igual a 3 despues de haber jugado el tercer nivel.







En esta parte del diagrama de tiempo se presiona y se suelta mejores_puntuaciones, se resetea el contador de las direcciones de la memoria de puntuaciones. Se activa el sel_displays en ‘1’ lo cual determina que los displays mostrarán las mejores puntuaciones, mas no la puntuación del jugador. Luego se muestra las 10 mejores puntuaciones, cada una por 5 segundos, cuando se cumplen los 5 segundos, se habilita el contador para cambiar la dirección de la RAM de puntuaciones. Al terminar se regresa al estado inicial.

VI. Código VHDL de los bloques utilizados

- Bloque Divisor de Clock: Entrada de 50MHz, Salidas de 1MHz, 100KHz, 10KHz, 1KHz, 100Hz, 10Hz, 1Hz.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_ARITH.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY CLOCK_DIV_50 IS
    PORT
    (   CLOCK_50Mhz     :IN   STD_LOGIC;
        CLOCK_1Mhz      :OUT  STD_LOGIC;
        CLOCK_100Khz    :OUT  STD_LOGIC;
        CLOCK_10Khz     :OUT  STD_LOGIC;
        CLOCK_1Khz      :OUT  STD_LOGIC;
        CLOCK_100hz     :OUT  STD_LOGIC;
        CLOCK_10hz      :OUT  STD_LOGIC;
        CLOCK_1hz       :OUT  STD_LOGIC);
END CLOCK_DIV_50;

ARCHITECTURE a OF CLOCK_DIV_50 IS
    SIGNAL   count_1Mhz: STD_LOGIC_VECTOR(5 DOWNTO 0);
    SIGNAL   count_100Khz, count_10Khz, count_1Khz: STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL   count_100hz, count_10hz, count_1hz: STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL   clock_1Mhz_int, clock_100khz_int, clock_10khz_int, clock_1khz_int: STD_LOGIC;
    SIGNAL   clock_100hz_int, clock_10hz_int, clock_1hz_int: STD_LOGIC;

BEGIN
    PROCESS
    BEGIN
    -- Divide by 50
        WAIT UNTIL clock_50Mhz'EVENT and clock_50Mhz = '1';   -- 24 Mhz
            IF count_1Mhz < 49 THEN
                count_1Mhz <= count_1Mhz + 1;
            ELSE
                count_1Mhz <= "000000";
            END IF;
            IF count_1Mhz < 4 THEN
                clock_1Mhz_int <= '0';
            ELSE
                clock_1Mhz_int <= '1';
            END IF;
            -- Ripple clocks are used in this code to save prescalar hardware
            -- Sync all clock prescalar outputs back to master clock signal
            clock_1Mhz <= clock_1Mhz_int;
            clock_100khz <= clock_100khz_int;
            clock_10khz <= clock_10khz_int;
            clock_1khz <= clock_1khz_int;
            clock_100hz <= clock_100hz_int;
            clock_10hz <= clock_10hz_int;
            clock_1hz <= clock_1hz_int;
    END PROCESS;
        ---Divide by 10
    PROCESS
    BEGIN
        WAIT UNTIL clock_1Mhz_int'EVENT and clock_1Mhz_int = '1';
            ELSE
                count_100khz <= "000";
                clock_100khz_int <= NOT clock_100khz_int;
            END IF;
    END PROCESS;
        ---Divide by 10
    PROCESS
    BEGIN
        WAIT UNTIL clock_100Khz_int'EVENT and clock_100Khz_int = '1';
            IF count_10khz /= 4 THEN
                count_10khz <= count_10khz + 1;
            ELSE
                count_10khz <= "000";
                clock_10khz_int <= NOT clock_10khz_int;
            END IF;
    END PROCESS;
        ---Divide by 10
    PROCESS
    BEGIN
        WAIT UNTIL clock_10Khz_int'EVENT and clock_10Khz_int = '1';
            IF count_1khz /= 4 THEN
                count_1khz <= count_1khz + 1;
            ELSE
                count_1khz <= "000";
                clock_1khz_int <= NOT clock_1khz_int;
            END IF;
    END PROCESS;
        ---Divide by 10
    PROCESS
    BEGIN
        WAIT UNTIL clock_1khz_int'EVENT and clock_1khz_int = '1';
            IF count_100hz /= 4 THEN
                count_100hz <= count_100hz + 1;
            ELSE
                count_100hz <= "000";
                clock_100hz_int <= NOT clock_100hz_int;
            END IF;
    END PROCESS;
        ---Divide by 10
    PROCESS
    BEGIN
        WAIT UNTIL clock_100hz_int'EVENT and clock_100hz_int = '1';
            IF count_10hz /= 4 THEN
                count_10hz <= count_10hz + 1;
            ELSE
                count_10hz <= "000";
                clock_10hz_int <= NOT clock_10hz_int;
            END IF;
    END PROCESS;
        ---Divide by 10
    PROCESS
    BEGIN
```

- Contador de Nivel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity contador_nivel is
    PORT (
        clk    : IN  STD_LOGIC;
        areset : IN  STD_LOGIC;
        aload  : IN  STD_LOGIC;
        enable_cnt: IN STD_LOGIC;
        data   : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
        cnt_out: OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
end contador_nivel;

architecture Behavioral of contador_nivel is
    -- Señal temporal para el contador.
    signal cnt_tmp: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
begin
    proceso_contador: process (aload, areset, clk, data, enable_cnt) begin
        if areset = '1' then
            cnt_tmp <= "0000";
        elsif aload = '1' then
            cnt_tmp <= data;
        elsif rising_edge(clk) and (enable_cnt = '1') then
            cnt_tmp <= cnt_tmp + 1;
        end if;
    end process;

    cnt_out <= cnt_tmp;
end Behavioral;
```

Este bloque se utiliza para contar el nivel en el que se encuentra el jugador durante el transcurso del juego. Este bloque se resetea en el estado inicial. Cuando se aplasta y suelta jugar, se habilita el contador de nivel y suma ‘1’, para mostrar el primer nivel, luego cuando se activa la señal de ciento_ochenta_seg, se habilita el contador para sumar ‘1’ lo cual llevaría a los siguientes niveles.

- Contador de puntos

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity contador_puntos_11bits is
    PORT (
        clk    : IN  STD_LOGIC;
        areset : IN  STD_LOGIC;
        aload  : IN  STD_LOGIC;
        enable_cnt: IN STD_LOGIC;
        data   : IN  STD_LOGIC_VECTOR(10 DOWNTO 0);
        cnt_out: OUT STD_LOGIC_VECTOR(10 DOWNTO 0)
    );
end contador_puntos_11bits;

architecture Behavioral of contador_puntos_11bits is
    -- Señal temporal para el contador.
    signal cnt_tmp: STD_LOGIC_VECTOR(10 DOWNTO 0) := "00000000000";
begin
    proceso_contador: process (aload, areset, clk, data, enable_cnt) begin
        if areset = '1' then
            cnt_tmp <= "00000000000";
        elsif aload = '1' then
            cnt_tmp <= data;
        elsif rising_edge(clk) and enable_cnt = '1' then
            cnt_tmp <= cnt_tmp + "00000000011";
        end if;
    end process;

    cnt_out <= cnt_tmp;
end Behavioral;
```

Este bloque se utiliza para contar los puntos que acumula el jugador durante el juego. La entrada de ‘clk’ es multiplexada dependiendo del nivel en que se encuentre el jugador. Este bloque suma ‘3’ puntos cada vez que el jugador pasa un obstáculo.

- Contador de 180 segundos

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY contador_180_seg IS
   PORT(clock,enable,reset : IN STD_LOGIC;
        ciento_ochenta_seg : OUT STD_LOGIC;
        Q : Buffer STD_LOGIC_VECTOR (7 downto 0));
END contador_180_seg;

ARCHITECTURE sol OF contador_180_seg IS
BEGIN

   PROCESS(clock,reset)
   BEGIN
      if reset='1' then Q<="00000000";ciento_ochenta_seg<='0';
      elsif (clock'event and clock='1') then
         if enable='1' then
            if (Q="10110100") then Q<="00000000";ciento_ochenta_seg<='1';
            else Q<=Q+1;ciento_ochenta_seg<='0';
            end if;
         end if;
      end if;
   END PROCESS;
END sol;
```

Este bloque se utiliza para contar 3 minutos cada vez que inicia un nivel del juego. Cuando llega la señal de que el contador ha llegado a 180, la MSS se encarga de mandar la señal habilitadora para aumentar en 1 el contador de nivel.

- Contador de 5 segundos

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY contador_5_seg IS
   PORT(clock,enable,reset : IN STD_LOGIC;
        cinco_seg : OUT STD_LOGIC;
        Q : Buffer STD_LOGIC_VECTOR (2 downto 0));
END contador_5_seg;

ARCHITECTURE sol OF contador_5_seg IS
BEGIN

   PROCESS(clock,reset)
   BEGIN
      if reset='1' then Q<="000";cinco_seg<='0';
      elsif (clock'event and clock='1') then
         if enable='1' then
            if (Q="101") then Q<="000";cinco_seg<='1';
            else Q<=Q+"001";cinco_seg<='0';
            end if;
         end if;
      end if;
   END PROCESS;
END sol;
```

Este bloque se utiliza para contar 5 segundos cada vez que se muestra una de las mejores puntuaciones. Cuando llega la señal de que el contador ha llegado a 5, la MSS se encarga de mandar la señal habilitadora para aumentar en 1 el contador de mejores puntuaciones.

- Bloque Antirrebote

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

   -- Debounce Pushbutton: Filters out mechanical switch bounce for around 40Ms.
ENTITY ANTIREBOTE IS
   PORT(PB_N, CLOCK_100Hz  : IN  STD_LOGIC;
        PB_SIN_REBOTE     : OUT STD_LOGIC);
END ANTIREBOTE;

ARCHITECTURE a OF ANTIREBOTE IS
   SIGNAL SHFT_PB        : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN

   -- Debounce clock should be approximately 10ms or 100Hz
   PROCESS
   BEGIN
      WAIT UNTIL (clock_100Hz'EVENT) AND (clock_100Hz = '1');
      -- use a shift register to filter switch contact bounce
      SHIFT_PB(2 DOWNTO 0) <= SHIFT_PB(3 DOWNTO 1);
      SHIFT_PB(3) <= NOT PB_N;
      IF SHIFT_PB(3 DOWNTO 0)="0000" THEN
            PB_SIN_REBOTE <= '0';
      ELSE
            PB_SIN_REBOTE <= '1';
      END IF;
   END PROCESS;
END a;
```

Este bloque nos permite eliminar el ruido que se produce en las señales provenientes de botoneras.

- Mux selector de frecuencia de nivel y Mux para el clock de puntos

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_selec_frec_nivel is
port (i0 : in std_logic;
      i1 : in std_logic;
      i2 : in std_logic;
      i3 : in std_logic;
      i4 : in std_logic;
      i5 : in std_logic;
      i6 : in std_logic;
      i7 : in std_logic;
      i8 : in std_logic;
      i9 : in std_logic;
      i10 : in std_logic;
      i11 : in std_logic;
      i12 : in std_logic;
      i13 : in std_logic;
      i14 : in std_logic;
      i15 : in std_logic;
     sel : in std_logic_vector(3 downto 0);
      frecout : out std_logic);
end mux_selec_frec_nivel;
architecture Behavioral of mux_selec_frec_nivel is
begin
    with sel select
       frecout <= i0 when "0000",
                  i1 when "0001",
                  i2 when "0010",
                  i3 when "0011",
                  i4 when "0100",
                  i5 when "0101",
                  i6 when "0110",
                  i7 when "0111",
                  i8 when "1000",
                  i9 when "1001",
                  i10 when "1010",
                  i11 when "1011",
                  i12 when "1100",
                  i13 when "1101",
                  i14 when "1110",
                  i15 when "1111";
end Behavioral;
```

Este mux de 16 entradas tiene como selector el número de nivel, donde en las entradas i0, i1, 12 tiene las frecuencias correspondientes a los niveles 1, 2 y 3 del juego, y en el resto de las entradas se conectan a GND debido a que son DON'T CARE. Para el contador de puntos se utiliza el mismo bloque solo que las entradas son las frecuencias a las que se aumenta puntos correspondientes a cada nivel.

- Mux para sumar puntos por aumento de nivel

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_suma is
port ( a: in std_logic_vector(10 DOWNTO 0);
       b: in std_logic_vector(10 DOWNTO 0);
       c: in std_logic_vector(10 DOWNTO 0);
       d: in std_logic_vector(10 DOWNTO 0);
       e: in std_logic_vector(10 DOWNTO 0);
       f: in std_logic_vector(10 DOWNTO 0);
       g: in std_logic_vector(10 DOWNTO 0);
       h: in std_logic_vector(10 DOWNTO 0);
       i: in std_logic_vector(10 DOWNTO 0);
       j: in std_logic_vector(10 DOWNTO 0);
       k: in std_logic_vector(10 DOWNTO 0);
       l: in std_logic_vector(10 DOWNTO 0);
       m: in std_logic_vector(10 DOWNTO 0);
       n: in std_logic_vector(10 DOWNTO 0);
       o: in std_logic_vector(10 DOWNTO 0);
       p: in std_logic_vector(10 DOWNTO 0);
       sel : in std_logic_vector(3 downto 0);
       sumaout : out std_logic_vector(10 downto 0));
end mux_suma;
architecture Behavioral of mux_suma is
begin
    with sel select
        sumaout <= a when "0000",
                   b when "0001",
                   c when "0010",
                   d when "0011",
                   e when "0100",
                   f when "0101",
                   g when "0110",
                   h when "0111",
                   i when "1000",
                   j when "1001",
                   k when "1010",
                   l when "1011",
                   m when "1100",
                   n when "1101",
                   o when "1110",
                   p when "1111";
end Behavioral;
```

Este multiplexor sirve para sumar los puntos por aumento de nivel, 0 puntos en el nivel 1, 5 puntos en el nivel 2 y 10 puntos en el nivel 3.

- Generador de Clock

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY delay_clock IS
PORT (
        Clk50Mhz: IN STD_LOGIC;
        Clk: OUT STD_LOGIC
        );
END delay_clock;

ARCHITECTURE rtl OF delay_clock IS
    CONSTANT max: INTEGER := 6250000;
    CONSTANT half: INTEGER := max/2;
    SIGNAL count: INTEGER RANGE 0 TO max;

BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clk50Mhz'EVENT and Clk50Mhz = '1';
        IF
            count < max THEN count <= count + 1;
            ELSE count <= 0;
        END IF;

        IF
            count < half THEN Clk <= '0';
            ELSE Clk <= '1';
        END IF;
    END PROCESS;
END rtl;
```

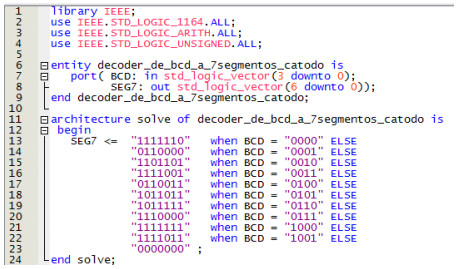
Con este bloque se puede generar cualquier clock a partir de una entrada de 50MHz, cambiando la variable constant max.

- Comparador de 2 entradas de 4 bits

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

entity comparador_de_2entradas_4bits is
   port ( A,B: in std_logic_vector(3 downto 0);
          C,D,E: out std_logic);
end comparador_de_2entradas_4bits;

architecture sol of comparador_de_2entradas_4bits is
begin
    C <= '1' when (A>B) else '0';
    D <= '1' when (A=B) else '0';
    E <= '1' when (A<B) else '0';
end sol;
```

Bloque que dependiendo de los valores de las entradas, arroja en alto o bajo nivel de voltaje las señales: mayor, igual, menor. En el juego utilizamos comparando el número de nivel con el número 3, y nos interesa la salida en que A=B.

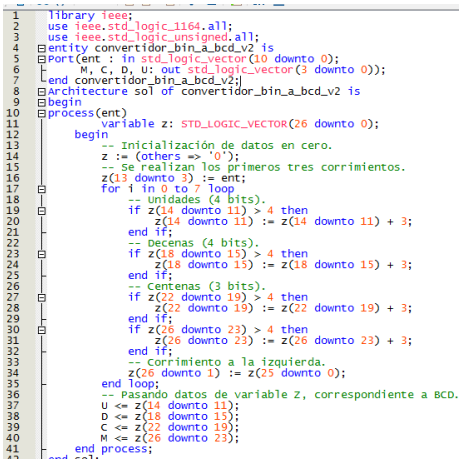
- Bloque Decodificador BCD a 7 segmentos (cátodo común)



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

entity decoder_de_bcd_a_7segmentos_catodo is
    port( BCD: in std_logic_vector(3 downto 0);
          SEG7: out std_logic_vector(6 downto 0));
end decoder_de_bcd_a_7segmentos_catodo;

architecture solve of decoder_de_bcd_a_7segmentos_catodo is
 begin
    SEG7 <=  "1111110"   when BCD = "0000" ELSE
             "0110000"   when BCD = "0001" ELSE
             "1101101"   when BCD = "0010" ELSE
             "1111001"   when BCD = "0011" ELSE
             "0110011"   when BCD = "0100" ELSE
             "1011011"   when BCD = "0101" ELSE
             "1011111"   when BCD = "0110" ELSE
             "1110000"   when BCD = "0111" ELSE
             "1111111"   when BCD = "1000" ELSE
             "1111011"   when BCD = "1001" ELSE
             "0000000" ;
end solve;
```

Bloque decodificador de 7 segmentos, convierte una cifra BCD en 7 segmentos configuración cátodo común.

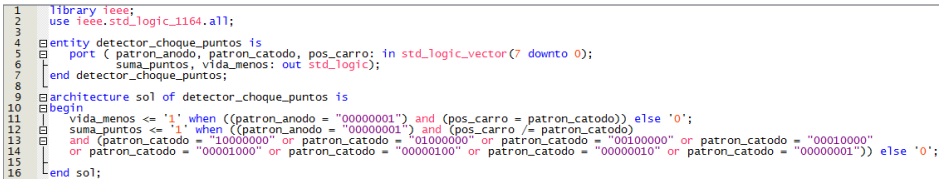
- Bloque convertidor de binario a 4 cifras BCD



```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity convertidor_bin_a_bcd_v2 is
Port(ent : in std_logic_vector(10 downto 0);
     M, C, D, U: out std_logic_vector(3 downto 0));
end convertidor_bin_a_bcd_v2;
Architecture sol of convertidor_bin_a_bcd_v2 is
begin
process(ent)
        variable z: STD_LOGIC_VECTOR(26 downto 0);
    begin
        -- Inicialización de datos en cero.
        z := (others => '0');
        -- Se realizan los primeros tres corrimientos.
        z(13 downto 3) := ent;
        for i in 0 to 7 loop
            -- Unidades (4 bits).
            if z(14 downto 11) > 4 then
                z(14 downto 11) := z(14 downto 11) + 3;
            end if;
            -- Decenas (4 bits).
            if z(18 downto 15) > 4 then
                z(18 downto 15) := z(18 downto 15) + 3;
            end if;
            -- Centenas (3 bits).
            if z(22 downto 19) > 4 then
                z(22 downto 19) := z(22 downto 19) + 3;
            end if;
            if z(26 downto 23) > 4 then
                z(26 downto 23) := z(26 downto 23) + 3;
            end if;
            -- Corrimiento a la izquierda.
            z(26 downto 1) := z(25 downto 0);
        end loop;
        -- Pasando datos de variable Z, correspondiente a BCD.
        U <= z(14 downto 11);
        D <= z(18 downto 15);
        C <= z(22 downto 19);
        M <= z(26 downto 23);
    end process;
end sol;
```

Genera 4 cifras de BCD con la entrada de un número binario de 11 bits. Para mostrar en los displays hasta un número de 2047 puntos.

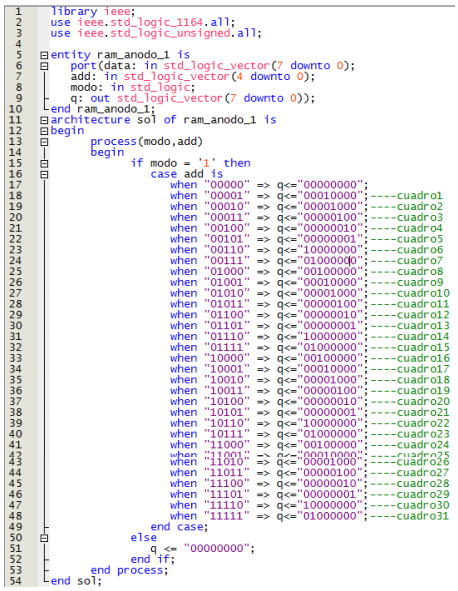
- Bloque detector de choque y suma de puntos



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

entity detector_choque_puntos is
    port ( patron_anodo, patron_catodo, pos_carro: in std_logic_vector(7 downto 0);
           suma_puntos, vida_menos: out std_logic);
end detector_choque_puntos;

architecture sol of detector_choque_puntos is
begin
    vida_menos <= '1' when ((patron_anodo = "00000001") and (pos_carro = patron_catodo)) else '0';
    suma_puntos <= '1' when ((patron_anodo = "00000001") and (pos_carro /= patron_catodo)
    and (patron_catodo = "10000000" or patron_catodo = "01000000" or patron_catodo = "00100000" or patron_catodo = "00010000"
    or patron_catodo = "00001000" or patron_catodo = "00000100" or patron_catodo = "00000010" or patron_catodo = "00000001")) else '0';

end sol;
```

Este bloque realiza comparaciones internas y verifica si el jugador choca con un obstáculo o si el jugador rebasa un obstáculo y cumplidas las condiciones se activarán las señales vida_menos o suma_puntos.

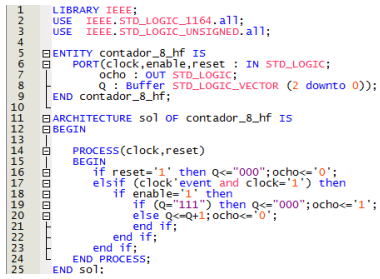
- Bloque RAM para la generación de los obstáculos



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

entity ram_anodo_1 is
    port(data: in std_logic_vector(7 downto 0);
    add: in std_logic_vector(4 downto 0);
    modo: in std_logic;
    q: out std_logic_vector(7 downto 0));
end ram_anodo_1;
architecture sol of ram_anodo_1 is
begin
        process(modo,add)
        begin
                if modo = '1' then
                   case add is
                      when "00000" => q<="00000000";
                      when "00001" => q<="00010000";----cuadro1
                      when "00010" => q<="00001000";----cuadro2
                      when "00011" => q<="00000100";----cuadro3
                      when "00100" => q<="00000010";----cuadro4
                      when "00101" => q<="00000001";----cuadro5
                      when "00110" => q<="10000000";----cuadro6
                      when "00111" => q<="01000000";----cuadro7
                      when "01000" => q<="00100000";----cuadro8
                      when "01001" => q<="00010000";----cuadro9
                      when "01010" => q<="00001000";----cuadro10
                      when "01011" => q<="00000100";----cuadro11
                      when "01100" => q<="00000010";----cuadro12
                      when "01101" => q<="00000001";----cuadro13
                      when "01110" => q<="10000000";----cuadro14
                      when "01111" => q<="01000000";----cuadro15
                      when "10000" => q<="00100000";----cuadro16
                      when "10001" => q<="00010000";----cuadro17
                      when "10010" => q<="00001000";----cuadro18
                      when "10011" => q<="00000100";----cuadro19
                      when "10100" => q<="00000010";----cuadro20
                      when "10101" => q<="00000001";----cuadro21
                      when "10110" => q<="10000000";----cuadro22
                      when "10111" => q<="01000000";----cuadro23
                      when "11000" => q<="00100000";----cuadro24
                      when "11001" => q<="00010000";----cuadro25
                      when "11010" => q<="00001000";----cuadro26
                      when "11011" => q<="00000100";----cuadro27
                      when "11100" => q<="00000010";----cuadro28
                      when "11101" => q<="00000001";----cuadro29
                      when "11110" => q<="10000000";----cuadro30
                      when "11111" => q<="01000000";----cuadro31
                end case;
            else
                q <= "00000000";
            end if;
        end process;
end sol;
```

Bloque de memoria RAM utilizado para mandar los bits correspondientes a los ánodos y cátodos de la matriz. Se utilizaron 8 memorias RAM para ánodo y 8 para cátodo.

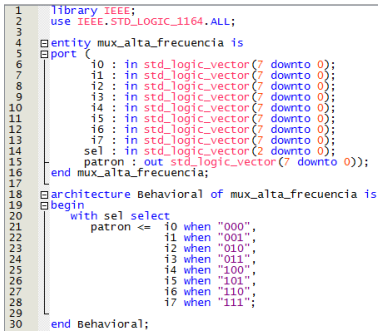
- Contador de Alta Frecuencia



```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY contador_8_hf IS
    PORT(clock,enable,reset : IN STD_LOGIC;
         ocho : OUT STD_LOGIC;
         Q : Buffer STD_LOGIC_VECTOR (2 downto 0));
END contador_8_hf;

ARCHITECTURE sol OF contador_8_hf IS
BEGIN

    PROCESS(clock,reset)
    BEGIN
        if reset='1' then Q<="000";ocho<='0';
        elsif (clock'event and clock='1') then
            if enable='1' then
                if (Q="111") then Q<="000";ocho<='1';
                else Q<=Q+1;ocho<='0';
                end if;
            end if;
        end if;
    END PROCESS;
END sol;
```

Contador que va del 0 al 7 a muy alta frecuencia para multiplexar el mux para los patrones.

- Mux de patrones



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

entity mux_alta_frecuencia is
port (
        i0 : in std_logic_vector(7 downto 0);
        i1 : in std_logic_vector(7 downto 0);
        i2 : in std_logic_vector(7 downto 0);
        i3 : in std_logic_vector(7 downto 0);
        i4 : in std_logic_vector(7 downto 0);
        i5 : in std_logic_vector(7 downto 0);
        i6 : in std_logic_vector(7 downto 0);
        i7 : in std_logic_vector(7 downto 0);
      sel : in std_logic_vector(2 downto 0);
    patron : out std_logic_vector(7 downto 0));
end mux_alta_frecuencia;

architecture Behavioral of mux_alta_frecuencia is
begin
    with sel select
        patron <=  i0 when "000",
                   i1 when "001",
                   i2 when "010",
                   i3 when "011",
                   i4 when "100",
                   i5 when "101",
                   i6 when "110",
                   i7 when "111";

end Behavioral;
```

Multiplexor para encender todos los patrones multiples veces por segundo cuyo selector es la salida del contador de alta frecuencia.

- Sumador de 11 bits

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY sum_v2 IS
   PORT (a : IN std_logic_vector(10 DOWNTO 0);
         b : IN std_logic_vector(10 DOWNTO 0);
         clk: IN std_logic;
         reset: IN std_logic;
         salida : OUT std_logic_vector(10 DOWNTO 0));
    END sum_v2;

ARCHITECTURE synth OF sum_v2 IS
BEGIN
   PROCESS (clk, reset, a) IS
   BEGIN
      if reset = '1' then
         salida <= "00000000000";
      elsif (clk'event and clk = '1') then
         salida <= a + b ;
      end if;
   END PROCESS;
END synth;
```

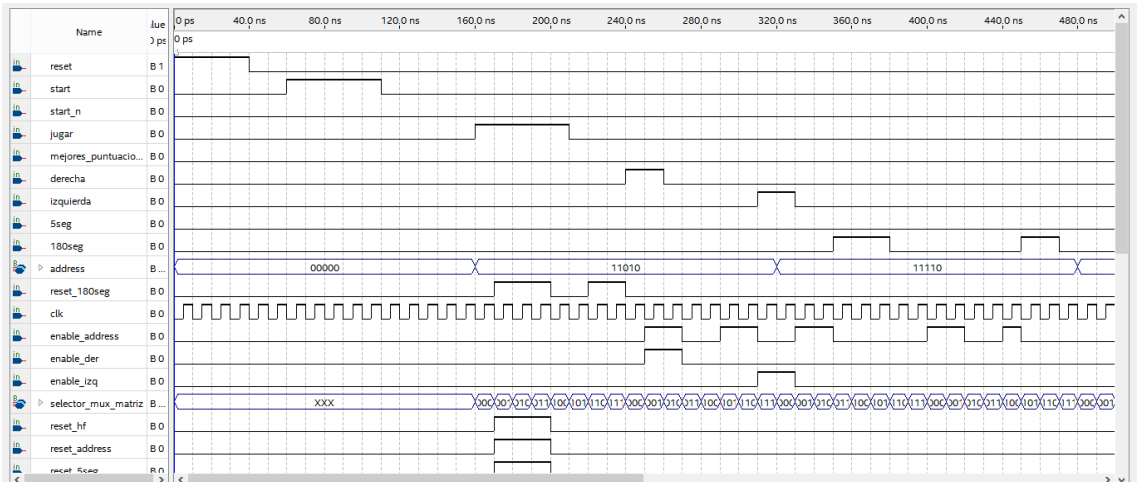
Este bloque sumador recibe el número de puntaje y los puntos por avanzar nivel, y se encargará de sumar 5 puntos en el segundo nivel y 10 puntos en el tercer nivel.

- Memoria RAM que ordena su contenido de forma descendente

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
entity ram_ord is
   Generic (DATA_WIDTH     : integer := 11;
            ADDRESS_WIDTH  : integer := 4);
   Port ( Clock,Enable  : in  STD_LOGIC;
          Reset    : in  STD_LOGIC;
          DataIn   : in  STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
          Address  : in  STD_LOGIC_VECTOR (ADDRESS_WIDTH - 1 downto 0);
          WriteEn  : in  STD_LOGIC;
          DataOut  : out STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
          Loaded   : out STD_LOGIC
   );
end ram_ord;
architecture Behavioral of ram_ord is
   type Memory_Array is array (0 to (2 ** ADDRESS_WIDTH) - 1) of STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
   signal Memory : Memory_Array;
begin
   -- Read process
   process (Clock)
   begin
      if rising_edge(Clock) then
         if Reset = '1' then
            -- Clear DataOut on Reset
            DataOut <= (others => '0');
         elsif Enable = '1' then
            DataOut <= Memory(to_integer(unsigned(Address)));
         end if;
      end if;
   end process;
   -- Write process
   process (Clock)
   begin
      if rising_edge(Clock) then
         loaded <='0';
         if Reset = '1' then
            -- Clear Memory on Reset
            for i in Memory'Range loop
               Memory(i) <= (others => '0');
            end loop;
         elsif  WriteEn = '1' then
            for i in Memory'Range loop
               if(DataIn > Memory(i)) then
                  for j in ((2 ** ADDRESS_WIDTH) - 1) downto 1 loop
                     exit when j=i;
                     Memory(j) <= Memory(j-1);
                  end loop;
                  Memory(i) <= DataIn;
                  loaded <='1';exit when TRUE;
               elsif(DataIn = Memory(i)) then
                  loaded <='1';exit when TRUE;
               end if;
            end loop;
         end if;
      end if;
   end process;
end Behavioral;
```

Este bloque de memoria RAM guarda el dato y luego lo ordena dentro de la memoria. De esta forma siempre se mostrarán los datos de mayor a menor.

DIAGRAMA COMPLETO DEL SISTEMA



En el diagrama de tiempo del sistema completo, se inicia con un reset. Luego se presiona y suelta las botonera de start y jugar, con lo cual se genera un reseteo de los contadores de 180 segundos, 5 segundos, contadores de direcciones y de alta frecuencia. Luego empieza el conteo de alta frecuencia lo cual mostraría todos los obstáculos en la matriz y el jugador. Al presionar derecha e izquierda se activan los correspondientes habilitadores de movimiento. Al activar la señal de 180 segundos, cambia de nivel, vuelve a resetear los contadores de tiempo, alta frecuencia y las direcciones, lo cual indica el correcto funcionamiento del sistema.

IX. DIAGRAMA ESQUEMATICO EN ALTIUM

ANEXOS

Link de Github con todos los archivos:

- **https://github.com/rianbrav/Car-Racing-VHDL**

Link del video de Youtube:

- **https://www.youtube.com/watch?v=m_fObZF8zH0**