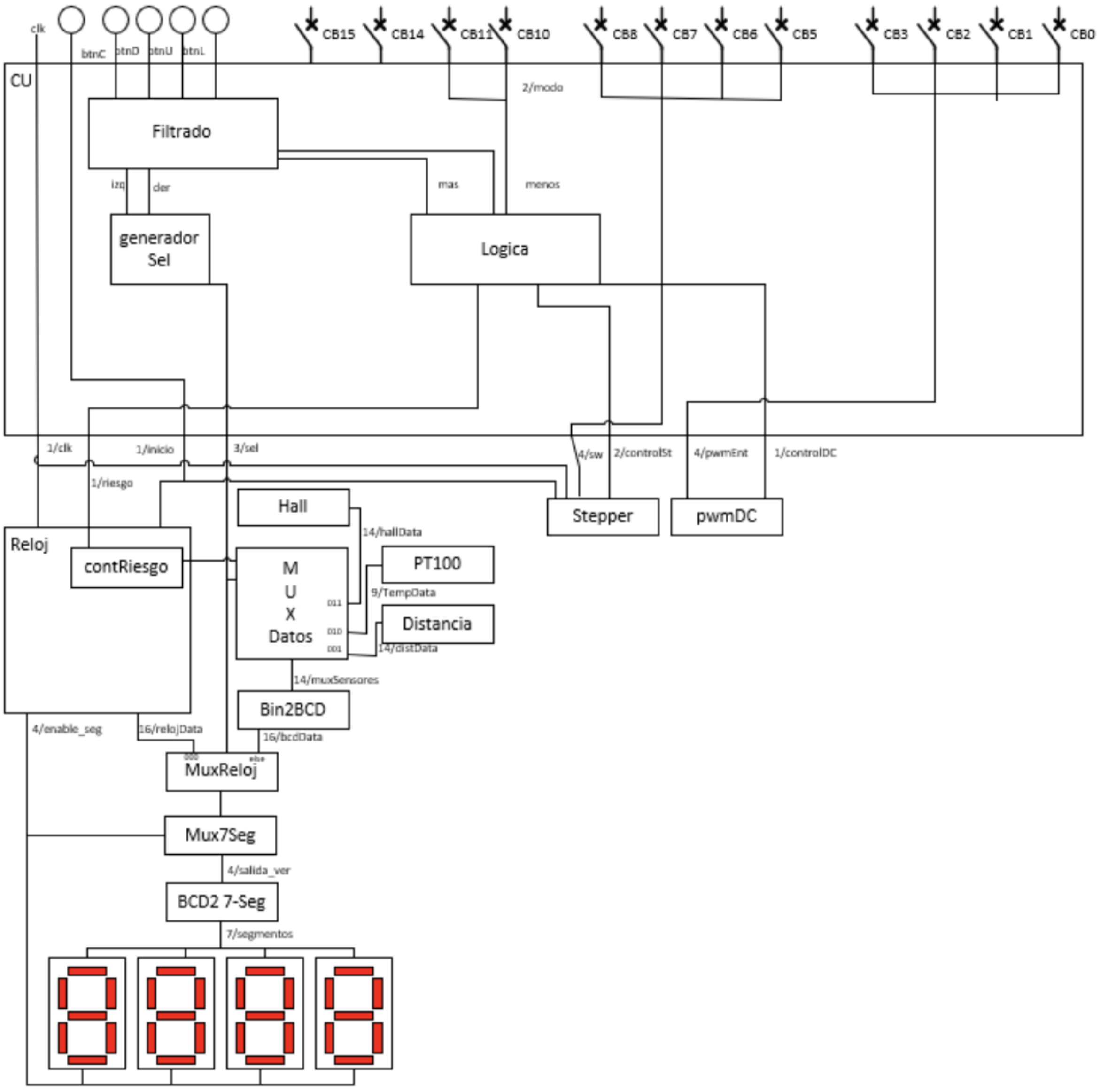
Diagrama



Código VHDL

```
VHDL Beautifier, Formatter

Only highlight, don't format
Keyword case: UPPERCASE | lowercase | Default
e.g. begin, case, when
Type name case: UPPERCASE | lowercase | Default
e.g. boolean, natural, string
Show More Settings ▼ Output Settings
Word wrap
```

```
IBRARY IEEE;
JSE IEEE.std_logic_1164.ALL;
JSE IEEE.numeric_std.ALL;
JSE IEEE.std_logic_unsigned.ALL;

NTITY todo IS
  PORT (
           clk : IN STD_LOGIC;
           inicio : IN STD_LOGIC;
           enableSw : IN STD_LOGIC;
           enableSwS : IN STD_LOGIC;
           enable_seg : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
           segmentos : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
           modo : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
           alarma : OUT STD_LOGIC;
           paro : IN STD_LOGIC;

           btnU : IN STD_LOGIC;
           btnD : IN STD_LOGIC;
           btnL : IN STD_LOGIC;
           btnR : IN STD_LOGIC;

           cale : OUT STD_LOGIC;

           dcEnc : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
           pwmEnt : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
           pwmPos : OUT STD_LOGIC;
           pwmNeg : OUT STD_LOGIC;

           enableSteper : OUT STD_LOGIC;
           fc2Tarj : IN STD_LOGIC;
           fc1Calle : IN STD_LOGIC;
           led : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
           sw : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
           dir : OUT STD_LOGIC;
           step : OUT STD_LOGIC;
           echo : IN STD_LOGIC;
           trigger : OUT STD_LOGIC;

           crc_en : OUT STD_LOGIC;
           -- data_out_lsb: out std_logic_vector (7 downto 0);
           --data_out_msb: out std_logic_vector (3 downto 0);
           ds_data_bus : INOUT STD_LOGIC
  );
ND todo;

RCHITECTURE Behavioral OF todo IS
  ------------------------------signals para control
  SIGNAL rDist : INTEGER RANGE 0 TO 1023;
  SIGNAL rTemp : INTEGER RANGE 0 TO 1023;
  SIGNAL rHall : INTEGER RANGE 0 TO 1023;
  SIGNAL riesgo : STD_LOGIC;

  ------------------------------signals para filtrado de btn
  --Up
  SIGNAL estadoU : STD_LOGIC_VECTOR (1 DOWNTO 0);
  --Down
  SIGNAL estadoD : STD_LOGIC_VECTOR (1 DOWNTO 0);
  --Left
  SIGNAL estadoL : STD_LOGIC_VECTOR (1 DOWNTO 0);
  --R
  SIGNAL estadoR : STD_LOGIC_VECTOR (1 DOWNTO 0);
  -------------------------------signals para pwmDC
  -- mot pwm
  SIGNAL estadoPWM : STD_LOGIC_VECTOR(2 DOWNTO 0);
  SIGNAL contPWM : INTEGER RANGE 0 TO 500000;
  SIGNAL tope : INTEGER RANGE 0 TO 500000;
  SIGNAL pwm : STD_LOGIC;
  SIGNAL sentido : STD_LOGIC;
  --PID
```

```vhdl
SIGNAL error : INTEGER RANGE -50000 TO 50000;
-------------------------------signals para hallDC
SIGNAL cont_baseDC : INTEGER RANGE 0 TO 100000000;
SIGNAL contador_ticks : INTEGER RANGE 0 TO 100000; --contador de microsegundos ticks de reloj
SIGNAL contador_ticks_2 : INTEGER RANGE 0 TO 100000; --contador de microsegundos ticks de reloj
SIGNAL rpm : INTEGER RANGE 0 TO 9999;
SIGNAL aux_rpm : INTEGER RANGE 0 TO 9999;
SIGNAL cont_microsDC : INTEGER RANGE 0 TO 100000;
SIGNAL estado_hall : STD_LOGIC_VECTOR (2 DOWNTO 0);
SIGNAL sentidoGiro : STD_LOGIC;
-- conexion del sensor hall
SIGNAL hall : STD_LOGIC_VECTOR (1 DOWNTO 0);
SIGNAL cont_hall : INTEGER RANGE 0 TO 1000;

--------------------------------signals para distancia
SIGNAL cont_micros : INTEGER RANGE 0 TO 60000;
SIGNAL cont_baseDist : INTEGER RANGE 0 TO 100;
SIGNAL cont_echo : INTEGER RANGE 0 TO 60000;
SIGNAL distancia_entero : INTEGER RANGE 0 TO 1023;

--------------------------------signals para stepper
SIGNAL dirAux : STD_LOGIC;
SIGNAL reloj_paso_paso : INTEGER RANGE 0 TO 10000000; --para generar la frecuencia de 50 Hz
SIGNAL paso_paso_aux : STD_LOGIC; --para generar la salida
SIGNAL tope_frecuencia_paso_paso : INTEGER RANGE 0 TO 1000000000; --para controlar la velocidad
SIGNAL frecuencia_paso_paso_entero : INTEGER RANGE 0 TO 10000;

--------------------------------signal para PT100
TYPE STATE_TYPE IS (WAIT_800ms, RESET, PRESENCE, SEND, WRITE_BYTE, WRITE_LOW, WRITE_HIGH, GET_DATA, READ_BIT);
SIGNAL state : STATE_TYPE;
SIGNAL data : STD_LOGIC_VECTOR(71 DOWNTO 0);
SIGNAL S_reset : STD_LOGIC;
SIGNAL i : INTEGER RANGE 0 TO 799999;
SIGNAL write_command : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL presence_signal : STD_LOGIC;
SIGNAL WRITE_BYTE_CNT : INTEGER RANGE 0 TO 8 := 0; -- citac pro odesilany bajt
SIGNAL write_low_flag : INTEGER RANGE 0 TO 2 := 0; -- priznak pozice ve stavu WRITE_LOW
SIGNAL write_high_flag : INTEGER RANGE 0 TO 2 := 0; -- priznak pozice ve stavu WRITE_HIGH
SIGNAL read_bit_flag : INTEGER RANGE 0 TO 3 := 0; -- priznak pozice ve stavu READ_BIT
SIGNAL GET_DATA_CNT : INTEGER RANGE 0 TO 72 := 0; -- citac pro pocet prectenych bitu
SIGNAL CONT_AUX : INTEGER RANGE 0 TO 100; --125; --chez a propochet ringa rangu
SIGNAL dataOut : STD_LOGIC_VECTOR(71 DOWNTO 0);

---------------------------------signals para bin2bcd
SIGNAL vector_aux : STD_LOGIC_VECTOR (29 DOWNTO 0);
SIGNAL estado : STD_LOGIC_VECTOR (1 DOWNTO 0);
SIGNAL cont_bits : INTEGER RANGE 0 TO 13;
SIGNAL fin : STD_LOGIC;

SIGNAL sal_mux : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL unidades : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL decenas : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL centenas : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL millares : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL enable_aux : STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL cont_base_enable : INTEGER RANGE 0 TO 100000;

SIGNAL cont : INTEGER RANGE 0 TO 100000000;

---------------------------------signals para intercomunicar Cajas
SIGNAL relojData : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL TempData : STD_LOGIC_VECTOR(8 DOWNTO 0);
SIGNAL distData : STD_LOGIC_VECTOR(13 DOWNTO 0);
SIGNAL to7seg : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL muxSensores : STD_LOGIC_VECTOR(13 DOWNTO 0);
SIGNAL bcdData : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL selLSB : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL hallData : STD_LOGIC_VECTOR(13 DOWNTO 0);
SIGNAL sel : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL rDistData : STD_LOGIC_VECTOR(14 DOWNTO 0);

SIGNAL der : STD_LOGIC;
```

```
  SIGNAL izq : STD_LOGIC;
  SIGNAL menos : STD_LOGIC;
  SIGNAL mas : STD_LOGIC;

  SIGNAL controlSt : STD_LOGIC_VECTOR(1 DOWNTO 0);
  SIGNAL controlDC : STD_LOGIC;

  -------------------------------------------------signal para 7seg
  SIGNAL salida_ver : STD_LOGIC_VECTOR (3 DOWNTO 0);

  -------------------------------------------------signals para Reloj
  SIGNAL cont_seg_uni : STD_LOGIC_VECTOR (3 DOWNTO 0);
  SIGNAL cont_seg_dec : STD_LOGIC_VECTOR (3 DOWNTO 0);
  SIGNAL cont_min_uni : STD_LOGIC_VECTOR (3 DOWNTO 0);
  SIGNAL cont_min_dec : STD_LOGIC_VECTOR (3 DOWNTO 0);

  SIGNAL cont_riesgo : INTEGER RANGE 0 TO 9999;
  SIGNAL cont_base : INTEGER RANGE 0 TO 100000000;
  SIGNAL cont_enable_aux : INTEGER RANGE 0 TO 100000;
  SIGNAL enable_seg_aux : STD_LOGIC_VECTOR (3 DOWNTO 0);

BEGIN

  led <= sel & "0110111101111";
  selLSB <= sel(1 DOWNTO 0);
  hall <= dcEnc;

  --------------------------------------------filtrado de pulsadores
  --U
  PROCESS (inicio, clk)
  BEGIN
          IF inicio = '1' THEN
                  estadoU <= "00";
          ELSIF rising_edge(clk) THEN
                  CASE estadoU IS
                          WHEN "00" =>
                                  IF btnU = '0' THEN
                                          estadoU <= "00";
                                  ELSE
                                          estadoU <= "01";
                                  END IF;
                          WHEN "01" =>
                                  IF btnU = '0' THEN
                                          estadoU <= "10";
                                  ELSE
                                          estadoU <= "01";
                                  END IF;
                          WHEN "10" =>
                                  estadoU <= "00";
                          WHEN OTHERS =>
                                  estadoU <= "00";
                  END CASE;
          END IF;
  END PROCESS;

  PROCESS (estadoU)
  BEGIN
          CASE estadoU IS
                  WHEN "00" => mas <= '0';
                  WHEN "01" => mas <= '0';
                  WHEN "10" => mas <= '1';
                  WHEN OTHERS => mas <= '0';
          END CASE;
  END PROCESS;

  --D
  PROCESS (inicio, clk)
  BEGIN
          IF inicio = '1' THEN
                  estadoD <= "00";
          ELSIF rising_edge(clk) THEN
                  CASE estadoD IS
```

```vhdl
                                WHEN "00" =>
                                        IF btnD = '0' THEN
                                                estadoD <= "00";
                                        ELSE
                                                estadoD <= "01";
                                        END IF;
                                WHEN "01" =>
                                        IF btnD = '0' THEN
                                                estadoD <= "10";
                                        ELSE
                                                estadoD <= "01";
                                        END IF;
                                WHEN "10" =>
                                        estadoD <= "00";
                                WHEN OTHERS =>
                                        estadoD <= "00";
                        END CASE;
                END IF;
        END PROCESS;

        PROCESS (estadoD)
        BEGIN
                CASE estadoD IS
                        WHEN "00" => menos <= '0';
                        WHEN "01" => menos <= '0';
                        WHEN "10" => menos <= '1';
                        WHEN OTHERS => menos <= '0';
                END CASE;
        END PROCESS;

        --L
        PROCESS (inicio, clk)
        BEGIN
                IF inicio = '1' THEN
                        estadoL <= "00";
                ELSIF rising_edge(clk) THEN
                        CASE estadoL IS
                                WHEN "00" =>
                                        IF btnL = '0' THEN
                                                estadoL <= "00";
                                        ELSE
                                                estadoL <= "01";
                                        END IF;
                                WHEN "01" =>
                                        IF btnL = '0' THEN
                                                estadoL <= "10";
                                        ELSE
                                                estadoL <= "01";
                                        END IF;
                                WHEN "10" =>
                                        estadoL <= "00";
                                WHEN OTHERS =>
                                        estadoL <= "00";
                        END CASE;
                END IF;
        END PROCESS;

        PROCESS (estadoL)
        BEGIN
                CASE estadoL IS
                        WHEN "00" => izq <= '0';
                        WHEN "01" => izq <= '0';
                        WHEN "10" => izq <= '1';
                        WHEN OTHERS => izq <= '0';
                END CASE;
        END PROCESS;

        --R
        PROCESS (inicio, clk)
        BEGIN
                IF inicio = '1' THEN
                        estadoR <= "00";
```

```
		ELSIF rising_edge(clk) THEN
			CASE estadoR IS
				WHEN "00" =>
					IF btnR = '0' THEN
						estadoR <= "00";
					ELSE
						estadoR <= "01";
					END IF;
				WHEN "01" =>
					IF btnR = '0' THEN
						estadoR <= "10";
					ELSE
						estadoR <= "01";
					END IF;
				WHEN "10" =>
					estadoR <= "00";
				WHEN OTHERS =>
					estadoR <= "00";
			END CASE;
		END IF;
	END PROCESS;

	PROCESS (estadoR)
	BEGIN
		CASE estadoR IS
			WHEN "00" => der <= '0';
			WHEN "01" => der <= '0';
			WHEN "10" => der <= '1';
			WHEN OTHERS => der <= '0';
		END CASE;
	END PROCESS;
	-----------------------------------------------------------------------------
	----------------------------CONTROL------------------------------------------
	-----------------------------------------------------------------------------

	--generacion de lineas de control
	PROCESS (inicio, clk, modo)
	BEGIN
		IF inicio = '1'THEN
			controlSt <= "00";
		ELSIF rising_edge(clk) THEN
			CASE modo IS
				WHEN "00" => --modo ida y vuelta continuo
					cale <= '0';
					controlSt <= "11";
					controlDC <= '0';
				WHEN "01" => --ida y vuelta hasta temperatura de riesgo
					controlDC <= '0';
					IF riesgo = '1' THEN
						controlSt <= "00";
						cale <= '0';
					ELSE
						controlSt <= "11";
						cale <= '1';
					END IF;
				WHEN "10" => --poner stepper a distancia especificada
					controlDC <= '0';
					cale <= '0';
					IF (rDist > distData) THEN
						controlSt <= "01";
					ELSIF rDist < distData THEN
						controlSt <= "10";
					ELSE controlSt <= "00";
					END IF;
				WHEN "11" =>
					cale <= '0';
					controlSt <= "00";
					controlDC <= '1';
				WHEN OTHERS => controlSt <= "00";
			END CASE;
		END IF;
	END PROCESS;
```

```vhdl
--insercion de datos por btns
PROCESS (clk, inicio, sel)
BEGIN
         IF inicio = '1' THEN
                  --    rDist <= 0;
                  --    rHall <= 0;
                  --    rTemp <= 0;
                  riesgo <= '0';
         ELSIF rising_edge(clk) THEN
                  CASE sel IS
                           WHEN "101" =>
                                    IF mas = '1' THEN
                                             rTemp <= rTemp + 1;
                                    ELSIF menos = '1'THEN
                                             rTemp <= rTemp - 1;
                                    END IF;
                           WHEN "110" =>
                                    IF mas = '1' THEN
                                             rDist <= rDist + 1;
                                    ELSIF menos = '1'THEN
                                             rDist <= rDist - 1;
                                    END IF;
                           WHEN "111" =>
                                    IF paro = '0'THEN
                                             IF mas = '1' THEN
                                                      rHall <= rHall + 1;
                                             ELSIF menos = '1'THEN
                                                      rHall <= rHall - 1;
                                             END IF;
                                    END IF;
                           WHEN OTHERS =>
                  END CASE;
         END IF;
         IF (TempData > rTemp) THEN
                  riesgo <= '1';
         ELSE
                  riesgo <= '0';
         END IF;
END PROCESS;
--generacion de sel
PROCESS (clk, inicio)
BEGIN
         IF inicio = '1' THEN
                  sel <= "000";
         ELSIF rising_edge(clk) THEN
                  IF (der = '1') THEN
                           sel <= sel + 1;
                  ELSIF (izq = '1') THEN
                           sel <= sel - 1;
                  END IF;
         END IF;
END PROCESS;

---------------------------------------------process para pwmDC
PROCESS (controlDC, pwmEnt, clk, inicio)
BEGIN
         IF controlDC = '0'THEN
                  CASE pwmEnt IS
                           WHEN "0000" => tope <= 0;
                           WHEN "0001" => tope <= 50000;
                           WHEN "0010" => tope <= 100000;
                           WHEN "0011" => tope <= 150000;
                           WHEN "0100" => tope <= 200000;
                           WHEN "0101" => tope <= 250000;
                           WHEN "0110" => tope <= 300000;
                           WHEN "0111" => tope <= 350000;
                           WHEN "1000" => tope <= 400000;
                           WHEN "1001" => tope <= 450000;
                           WHEN "1010" => tope <= 500000;
                           WHEN OTHERS => tope <= 0;
                  END CASE;
```

```vhdl
          ELSE
                    --PID
                    IF inicio = '1'THEN
                              error <= 0;
                    ELSIF rising_edge(clk) THEN
                              error <= (rHall - rpm) * 10;
                              IF error > 50000 THEN
                                        tope <= 50000;

                              END IF;
                              --          if(rHall > rpm)then
                              --              tope <= tope +10;
                              --          elsif(rHall < rpm)then
                              --              tope <= tope -10;
                              --          else
                              --              tope <= tope;
                              --          end if;
                    END IF;
          END IF;

END PROCESS;
PROCESS (sentido)
BEGIN
          IF (sentido = '0') THEN
                    pwmPos <= pwm;
                    pwmNeg <= '0';
          ELSE
                    pwmPos <= '0';
                    pwmNeg <= pwm;
          END IF;
END PROCESS;

PROCESS (inicio, clk)
BEGIN
          IF inicio = '1' THEN
                    contPWM <= 0;
                    estadoPWM <= "000";
          ELSIF rising_edge(clk) THEN
                    CASE estadoPWM IS
                              WHEN "000" =>
                                        contPWM <= 0;
                                        IF (tope = 0) THEN
                                                  estadoPWM <= "001";
                                        ELSE
                                                  estadoPWM <= "010";
                                        END IF;
                              WHEN "001" =>
                                        contPWM <= 1;
                                        estadoPWM <= "011";
                              WHEN "010" =>
                                        contPWM <= 1;
                                        estadoPWM <= "100";
                              WHEN "011" =>
                                        contPWM <= contPWM + 1;
                                        IF (contPWM = 500000 AND tope = 0) THEN
                                                  estadoPWM <= "001";
                                        ELSIF (contPWM = 500000) THEN
                                                  estadoPWM <= "010";
                                        END IF;
                              WHEN "100" =>
                                        contPWM <= contPWM + 1;
                                        IF (contPWM = tope AND tope = 500000) THEN
                                                  estadoPWM <= "010";
                                        ELSIF (contPWM = tope) THEN
                                                  estadoPWM <= "011";
                                        END IF;
                              WHEN OTHERS =>
                                        contPWM <= 0;
                                        estadoPWM <= "000";
                    END CASE;
          END IF;
END PROCESS;
```

```vhdl
PROCESS (estadoPWM)
BEGIN
         CASE estadoPWM IS
                  WHEN "000" => pwm <= '0';
                  WHEN "001" => pwm <= '0';
                  WHEN "010" => pwm <= '1';
                  WHEN "011" => pwm <= '0';
                  WHEN "100" => pwm <= '1';
                  WHEN OTHERS => pwm <= '0';
         END CASE;
END PROCESS;

--------------------------------------------process para hallDC
-- cuenta las revoluciones del motor cada segundo y luego las pasa a la variable rev_per_seg
PROCESS (clk, inicio)
BEGIN
         IF rising_edge(clk) THEN
                  IF inicio = '1' THEN
                           contador_ticks <= 0;
                           cont_microsDC <= 0;
                           rpm <= 0;
                           aux_rpm <= 0;
                           estado_hall <= "000";
                           cont_hall <= 0;
                  ELSE
                           IF contador_ticks = 100 THEN
                                    CASE estado_hall IS
                                             WHEN "000" => -- estado_hall inicial

                                                      cont_microsDC <= 0;

                                                      IF hall = "00" THEN
                                                               estado_hall <= "001";
                                                      END IF;

                                             WHEN "001" =>
                                                      cont_microsDC <= cont_microsDC + 1;
                                                      rpm <= rpm;

                                                      IF cont_microsDC >= 60000 THEN
                                                               rpm <= 0;
                                                               estado_hall <= "000";
                                                      ELSIF hall = "01" THEN
                                                               sentidoGiro <= '0';
                                                               estado_hall <= "010";
                                                      ELSIF hall = "10" THEN
                                                               sentidoGiro <= '1';
                                                               estado_hall <= "011";
                                                      END IF;

                                             WHEN "010" =>
                                                      cont_microsDC <= cont_microsDC + 1;

                                                      IF cont_microsDC >= 60000 THEN
                                                               rpm <= 0;
                                                               estado_hall <= "000";
                                                      ELSIF hall = "00" THEN
                                                               estado_hall <= "100";
                                                      END IF;

                                             WHEN "011" =>
                                                      cont_microsDC <= cont_microsDC + 1;

                                                      IF cont_microsDC >= 60000 THEN
                                                               rpm <= 0;
                                                               estado_hall <= "000";
                                                      ELSIF hall = "00" THEN
                                                               estado_hall <= "100";
                                                      END IF;

                                             WHEN "100" =>
```

```
                                                aux_rpm <= (60000000 / (cont_microsDC * 8 * 120));
                                                IF cont_hall = 100 AND (aux_rpm < rpm - 2 OR aux_rpm > rpm
+ 2) THEN
                                                        rpm <= aux_rpm;
                                                        cont_hall <= 0;
                                                ELSE
                                                        cont_hall <= cont_hall + 1;
                                                END IF;
                                                estado_hall <= "000";

                                        WHEN OTHERS =>
                                                estado_hall <= "000";
                                END CASE;

                                contador_ticks <= 0;
                        ELSE
                                contador_ticks <= contador_ticks + 1;
                        END IF;
                END IF;
        END IF;
        IF paro = '0' THEN
                hallData <= STD_LOGIC_VECTOR(to_unsigned(rpm, 14));
        END IF;
END PROCESS;
---------------------------------------------process para distancia
-- Contador de micros y generacion del ciclo de 60 ms
PROCESS (clk, inicio)
BEGIN
        IF inicio = '1' THEN
                cont_baseDist <= 0;
        ELSIF rising_edge(clk) THEN
                IF cont_baseDist = 100 THEN
                        cont_baseDist <= 0;
                        IF cont_micros = 60000 THEN
                                cont_micros <= 0;
                        ELSE
                                cont_micros <= cont_micros + 1;
                        END IF;
                ELSE
                        cont_baseDist <= cont_baseDist + 1;
                END IF;
        END IF;
END PROCESS;

-- generador del trigger
PROCESS (cont_micros)
BEGIN
        IF cont_micros < 10 THEN
                trigger <= '1';
        ELSE
                trigger <= '0';
        END IF;
END PROCESS;

-- Contador de echo
PROCESS (clk, inicio)
BEGIN
        IF inicio = '1' THEN
                cont_echo <= 0;
        ELSIF rising_edge(clk) THEN
                IF cont_micros = 0 THEN
                        cont_echo <= 0;
                ELSE
                        IF cont_baseDist = 0 THEN
                                IF echo = '1' THEN
                                        cont_echo <= cont_echo + 1;
                                END IF;
                        END IF;
                END IF;
        END IF;
END PROCESS;
```

```vhdl
-- Registro y calculo de la distancia por cada ciclo
PROCESS (inicio, clk)
BEGIN
	IF inicio = '1' THEN
		distancia_entero <= 0;
	ELSIF rising_edge(clk) THEN
		IF cont_micros = 60000 THEN
			distancia_entero <= cont_echo/58;
		END IF;
	END IF;
END PROCESS;
distData <= "0000" & STD_LOGIC_VECTOR(to_unsigned(distancia_entero, 10));
---------------------------------------------process para stepper
frecuencia_paso_paso_entero <= to_integer(unsigned(sw & "00000"));
tope_frecuencia_paso_paso <= (100000000/frecuencia_paso_paso_entero)/2;

--control de Stepper
PROCESS (clk, inicio, controlST)
BEGIN
	IF inicio = '1' THEN
		dirAux <= '0';
		enableSteper <= enableSwS;
	ELSIF rising_edge(clk) THEN
		CASE controlSt IS
			WHEN "00" => --parado
				enableSteper <= '0';
			WHEN "01" => --en direccion calle
				dirAux <= '1';
				IF (fc1Calle = '1') THEN
					enableSteper <= '0';
				ELSE
					enableSteper <= enableSwS;
				END IF;
			WHEN "10" => --en direccion placa
				dirAux <= '0';
				IF (fc2Tarj = '1') THEN
					enableSteper <= '0';
				ELSE
					enableSteper <= enableSwS;
				END IF;
			WHEN "11" => --en las 2 direcciones
				enableSteper <= enableSwS;

				--        if(fc2Tarj = '0' and fc1Calle = '0')then
				--          dirAux <= dirAux;
				--         else
				--             if(fc2Tarj = '1')then
				--             dirAux <= '1';
				--             else
				--               dirAux <= '0';
				--             end if;
				--         end if;

				IF (fc2Tarj = '0' AND fc1Calle = '0') THEN
					dirAux <= dirAux;
				ELSE
					IF (fc2Tarj = '1') THEN
						dirAux <= '1';
					ELSE
						dirAux <= '0';
					END IF;
				END IF;
			WHEN OTHERS =>
				dirAux <= '0';
				enableSteper <= enableSwS;
		END CASE;
	END IF;
END PROCESS;

PROCESS (clk)
BEGIN
	IF inicio = '1' THEN
```

```
                  reloj_paso_paso <= 0;
          ELSIF rising_edge(clk) THEN
                  IF reloj_paso_paso = tope_frecuencia_paso_paso THEN --0 then
                          reloj_paso_paso <= 0;
                  ELSE
                          reloj_paso_paso <= reloj_paso_paso + 1;
                  END IF;
          END IF;
  END PROCESS;

  PROCESS (clk)
  BEGIN
          IF clk = '1' AND clk'event THEN
                  IF reloj_paso_paso = 0 THEN -- con el A
                          paso_paso_aux <= NOT(paso_paso_aux);
                  END IF;
          END IF;
  END PROCESS;
  dir <= dirAux;

  PROCESS (clk, enableSw)
  BEGIN
          IF (rising_edge(clk)) THEN
                  IF (enableSw = '1') THEN
                          step <= paso_paso_aux;
                  ELSE
                          step <= '0';
                  END IF;
          END IF;
  END PROCESS;

  ----------------------------------------------process para PT100
  PROCESS (clk)
          CONSTANT PRESENCE_ERROR_DATA : STD_LOGIC_VECTOR(71 DOWNTO 0) :=
'111111111111111111111111111111111111111111111111111111111111111111111111";

          VARIABLE bit_cnt : INTEGER RANGE 0 TO 71; -- prave cteny bit
          VARIABLE flag : INTEGER RANGE 0 TO 5; -- priznak pro odesilany prikaz

  BEGIN
          IF rising_edge(clk) AND cont_aux = 0 THEN
                  CASE state IS
                          WHEN RESET => -- stav pro reset senzoru
                                  S_reset <= '0'; -- reset citace
                                  IF (i = 0) THEN
                                          ds_data_bus <= '0'; -- zacatek resetovaciho pulzu
                                  ELSIF (i = 485) THEN
                                          ds_data_bus <= 'Z'; --aqui -- uvolneni sbernice
                                  ELSIF (i = 550) THEN
                                          presence_signal <= ds_data_bus; -- odebrani vzorku pro zjisteni
ritomnosti senzoru
                                  ELSIF (i = 1000) THEN
                                          state <= PRESENCE; -- prechod do stavu PRESENCE
                                  END IF;

                          WHEN PRESENCE => -- stav pro zjisteni pritomnosti senzoru na sbernici
                                  -- detekce senzoru na sbernici
                                  IF (presence_signal = '0' AND ds_data_bus = '1') THEN -- senzor byl detkovan
                                          S_reset <= '1'; -- reset citace
                                          state <= SEND; -- inicializace dokoncena, prechod do stavu SEND
                                  ELSE -- senzor nebyl detekovan
                                          S_reset <= '1'; -- reset citace
                                          dataOut <= PRESENCE_ERROR_DATA; -- nastaveni dat indikujicich chybu
a vystup
                                          crc_en <= '1'; -- zahajeni vypoctu CRC
                                          state <= WAIT_800ms; -- prechod do stavu WAIT_800ms
                                  END IF;

                          WHEN SEND => -- stav pro odesilani prikazu pro senzor
                                  -- sekvence odesilanych prikazu rizena priznakem flag
                                  IF (flag = 0) THEN -- prvni prikaz
                                          flag := 1;
```

```
                    write_command <= "11001100"; -- prikaz CCh - SKIP ROM
                    state <= WRITE_BYTE; -- prechod do stavu WRITE BYTE
                ELSIF (flag = 1) THEN -- druhy prikaz
                    flag := 2;
                    write_command <= "01000100"; -- prikaz 44h - CONVERT TEMPERATURE
                    state <= WRITE_BYTE; -- prechod do stavu WRITE BYTE
                ELSIF (flag = 2) THEN -- treti prikaz
                    flag := 3;
                    state <= WAIT_800ms; -- prechod do stavu WAIT_800ms, cekani na
konceni prikazu 44h
                ELSIF (flag = 3) THEN -- treti prikaz
                    flag := 4;
                    write_command <= "11001100"; -- prikaz CCh - SKIP ROM
                    state <= WRITE_BYTE; -- prechod do stavu WRITE BYTE
                ELSIF (flag = 4) THEN -- ctvrty prikaz
                    flag := 5;
                    write_command <= "10111110"; -- prikaz BEh - READ SCRATCHPAD
                    state <= WRITE_BYTE; -- prechod do stavu WRITE BYTE
                ELSIF (flag = 5) THEN -- ukonceni vysilani prikazu
                    flag := 0; -- reset priznaku pro odesilany prikaz
                    state <= GET_DATA; -- prechod do stavu GET_DATA
                END IF;

            WHEN WAIT_800ms => -- stav cekani po dobu 800 ms
                CRC_en <= '0'; -- reset priznaku pro zahajeni vypoctu CRC
                S_reset <= '0'; -- spusteni citace
                IF (i = 799) THEN -- konec periody citace
                    S_reset <= '1'; -- resetovani citace
                    state <= RESET; -- navrat do stavu RESET
                END IF;

            WHEN GET_DATA => -- stav pro precteni pameti scratchpad
                CASE GET_DATA_CNT IS -- pozice ve stavu GET_DATA
                    WHEN 0 TO 71 => -- cteni jednotlivych bitu pameti scratchpad
                        ds_data_bus <= '0'; -- zahajeni cteni na sbernici
                        GET_DATA_CNT <= GET_DATA_CNT + 1; -- inkrementace citace
ro prave cteny bit
                        state <= READ_BIT; -- prechod do stavu READ_BIT
                    WHEN 72 => -- pamet prectena (72 bitu)
                        bit_cnt := 0; -- reset citace pro prave cteny bit
                        GET_DATA_CNT <= 0; -- reset citace prectenych bitu
                        dataOut <= data(71 DOWNTO 0); -- odeslani prectenych dat
a vystupni port
                        CRC_en <= '1'; -- spusteni vypoctu CRC prectenych dat
                        state <= WAIT_800ms; -- navrat do stavu WAIT_800ms
                    WHEN OTHERS => -- chyba ve stavu GET_DATA
                        read_bit_flag <= 0; -- reset pozice ve stavu READ_BIT
                        GET_DATA_CNT <= 0; -- reset citace pro pocet prectenych
itu
                END CASE;

            WHEN READ_BIT => -- stav pro cteni bitu
                -- sekvence cteni bitu rizena priznakem read_bit_flag
                CASE read_bit_flag IS -- pozice ve stavu READ_BIT
                    WHEN 0 => -- vyslani zacatku casoveho slotu pro cteni
                        read_bit_flag <= 1;
                    WHEN 1 =>
                        ds_data_bus <= 'Z';--aqui -- uvolneni sbernice pro prijem
itu ze senzoru
                        S_reset <= '0'; -- zapnuti citace
                        IF (i = 13) THEN -- cekani 14 us
                            S_reset <= '1'; -- reset citace
                            read_bit_flag <= 2;
                        END IF;
                    WHEN 2 => -- odebrani vzorku dat ze sbernice
                        data(bit_cnt) <= ds_data_bus; -- ulozeni vzorku dat do
egistru
                        bit_cnt := bit_cnt + 1; -- zvyseni citace pro prave cteny
it
                        read_bit_flag <= 3;
                    WHEN 3 => -- dokonceni casoveho slotu
                        S_reset <= '0'; -- zapnuti citace
```

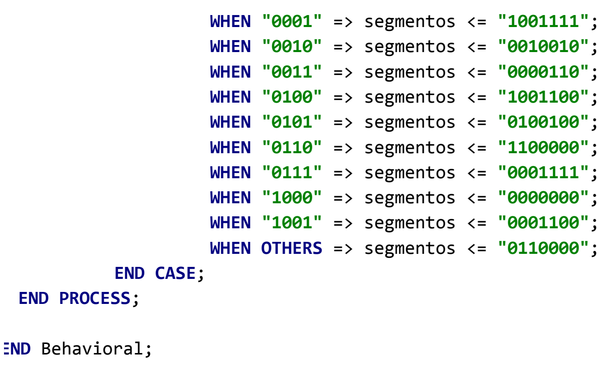
```
                                                IF (i = 63) THEN -- cekani 62 us
                                                        S_reset <= '1'; -- reset citace
                                                        read_bit_flag <= 0; -- reset pozice ve stavu
READ_BIT

                                                        state <= GET_DATA; -- navrat do stavu GET_DATA
                                                END IF;
                                        WHEN OTHERS => -- chyba ve stavu READ_BIT
                                                read_bit_flag <= 0; -- reset pozice ve stavu READ_BIT
                                                bit_cnt := 0; -- reset citace pro prave cteny bit
                                                GET_DATA_CNT <= 0; -- reset citace prectenych bitu
                                                state <= RESET; -- reset senzoru
                                END CASE;

                        WHEN WRITE_BYTE => -- stav pro zapis bajtu dat na sbernici
                                -- sekvence zapisu bajtu dat rizena citacem WRITE_BYTE_CNT
                                CASE WRITE_BYTE_CNT IS -- pozice ve stavu WRITE_BYTE
                                        WHEN 0 TO 7 => -- odesilani bitu 0-7
                                                IF (write_command(WRITE_BYTE_CNT) = '0') THEN -- odesilany
bit ma hodnotu log. 0

                                                        state <= WRITE_LOW; -- prechod do stavu
WRITE_LOW

                                                ELSE -- odesilany bit ma hodnotu log. 1
                                                        state <= WRITE_HIGH; -- prechod do stavu

WRITE_HIGH

                                                END IF;
                                                WRITE_BYTE_CNT <= WRITE_BYTE_CNT + 1; -- inkrementace

citace odesilaneho bitu

                                        WHEN 8 => -- odesilani bajtu dokonceno
                                                WRITE_BYTE_CNT <= 0; -- reset citace odesilaneho bitu
                                                state <= SEND; -- navrat do stavu SEND
                                        WHEN OTHERS => -- chyba ve stavu WRITE_BYTE
                                                WRITE_BYTE_CNT <= 0; -- reset citace odesilaneho bitu
                                                write_low_flag <= 0; -- reset pozice ve stavu WRITE_LOW
                                                write_high_flag <= 0; -- reset pozice ve stavu WRITE_HIGH
                                                state <= RESET; -- reset senzoru
                                END CASE;

                        WHEN WRITE_LOW => -- stav pro zapis log. 0 na sbernici
                                -- casovy slot pro zapis log 0 rizeny priznakem write_low_flag
                                CASE write_low_flag IS -- pozice ve stavu WRITE_LOW
                                        WHEN 0 => -- vyslani zacatku casoveho slotu pro zapis log. 0
                                                ds_data_bus <= '0'; -- zacatek casoveho slotu
                                                S_reset <= '0'; -- zapnuti citace
                                                IF (i = 59) THEN -- cekani 60 us
                                                        S_reset <= '1'; -- reset citace
                                                        write_low_flag <= 1;
                                                END IF;
                                        WHEN 1 => -- uvolneni sbernice pro ukonceni casoveho slotu
                                                ds_data_bus <= 'Z';--aqui -- uvolneni sbernice
                                                S_reset <= '0'; -- zapnuti citace
                                                IF (i = 3) THEN -- cekani 4 us na ustaleni sbernice
                                                        S_reset <= '1'; -- reset citace
                                                        write_low_flag <= 2;
                                                END IF;
                                        WHEN 2 => -- konec zapisu log. 0
                                                write_low_flag <= 0; -- reset pozice ve stavu WRITE_LOW
                                                state <= WRITE_BYTE; -- navrat do stavu WRITE_BYTE
                                        WHEN OTHERS => -- chyba zapisu log. 0
                                                WRITE_BYTE_CNT <= 0; -- reset citace odesilaneho bitu
                                                write_low_flag <= 0; -- reset pozice ve stavu WRITE_LOW
                                                state <= RESET; -- reset senzoru
                                END CASE;

                        WHEN WRITE_HIGH => -- stav pro zapis log. 1 na sbernici
                                -- casovy slot pro zapis log 1 rizeny priznakem write_high_flag
                                CASE write_high_flag IS -- pozice ve stavu WRITE_HIGH
                                        WHEN 0 => -- vyslani zacatku casoveho slotu pro zapis log. 1
                                                ds_data_bus <= '0'; -- zacatek casoveho slotu
                                                S_reset <= '0'; -- zapnuti citace
                                                IF (i = 9) THEN -- cekani 10 us
                                                        S_reset <= '1'; -- reset citace
                                                        write_high_flag <= 1;
```

```vhdl
                                                        END IF;
                                                WHEN 1 => -- uvolneni sbernice pro ukonceni casoveho slotu
                                                        ds_data_bus <= 'Z'; -- uvolneni sbernice--aqui
                                                        S_reset <= '0'; -- zapnuti citace
                                                        IF (i = 53) THEN -- cekani 54 us
                                                                S_reset <= '1'; -- reset citace
                                                                write_high_flag <= 2;
                                                        END IF;
                                                WHEN 2 => -- konec zapisu log. 1
                                                        write_high_flag <= 0; -- reset pozice ve stavu WRITE_HIGH
                                                        state <= WRITE_BYTE; -- navrat do stavu WRITE BYTE
                                                WHEN OTHERS => -- chyba zapisu log. 1
                                                        WRITE_BYTE_CNT <= 0; -- reset citace odesilaneho bitu
                                                        write_high_flag <= 0; -- reset pozice ve stavu WRITE_HIGH
                                                        state <= RESET; -- reset senzoru
                                        END CASE;

                                WHEN OTHERS => -- chyba FSM
                                        state <= RESET; -- reset senzoru

                        END CASE;
                END IF;
        END PROCESS;

        -- Proces citace se synchronnim resetem
        PROCESS (clk, S_reset)

        BEGIN
                IF (rising_edge(clk)) THEN
                        IF cont_aux = 100 THEN --=125 then
                                cont_aux <= 0;
                                IF (S_reset = '1') THEN -- reset citace
                                        i <= 0; -- vynulovani citace
                                ELSE
                                        i <= i + 1; -- inkrementace citace
                                END IF;
                        ELSE
                                cont_aux <= cont_aux + 1;
                        END IF;
                END IF;

        END PROCESS;

        --data_out_lsb<=dataOut(7 downto 0);
        --data_out_msb<=dataOut(11 downto 8);

        TempData <= dataOut(12 DOWNTO 4);

        -----------------------------------------------process para Reloj
        PROCESS (inicio, clk, riesgo)
        BEGIN
                IF inicio = '1' OR NOT(modo = "01") THEN
                        cont_riesgo <= 0;
                        alarma <= '0';
                ELSIF rising_edge(clk) THEN
                        IF (riesgo = '1' AND cont_base = 100000000) THEN
                                alarma <= '1';
                                IF (cont_riesgo = 9999) THEN
                                        cont_riesgo <= 0;
                                ELSE
                                        cont_riesgo <= cont_riesgo + 1;
                                END IF;
                        ELSE
                                cont_riesgo <= 0;
                        END IF;
                END IF;
        END PROCESS;

        PROCESS (inicio, clk)
        BEGIN
                IF inicio = '1' THEN
                        cont_base <= 0;
```

```vhdl
		ELSIF rising_edge(clk) THEN
			IF cont_base = 100000000 THEN
				cont_base <= 0;
			ELSE
				cont_base <= cont_base + 1;
			END IF;
		END IF;
	END PROCESS;

	--descripcion del contador BCD
	PROCESS (inicio, clk)
	BEGIN
		IF inicio = '1' THEN
			cont_seg_uni <= "0000";
		ELSIF rising_edge(clk) THEN
			IF cont_base = 100000000 THEN
				IF cont_seg_uni = "1001" THEN
					cont_seg_uni <= "0000";
				ELSE
					cont_seg_uni <= cont_seg_uni + 1;
				END IF;
			END IF;
		END IF;
	END PROCESS;

	PROCESS (inicio, clk)
	BEGIN
		IF inicio = '1' THEN
			cont_seg_dec <= "0000";
		ELSIF rising_edge(clk) THEN
			IF cont_base = 100000000 AND cont_seg_uni = 9 THEN
				IF cont_seg_dec = 5 THEN
					cont_seg_dec <= "0000";
				ELSE
					cont_seg_dec <= cont_seg_dec + 1;
				END IF;
			END IF;
		END IF;
	END PROCESS;

	PROCESS (inicio, clk)
	BEGIN
		IF inicio = '1' THEN
			cont_min_uni <= "0000";
		ELSIF rising_edge(clk) THEN
			IF cont_base = 100000000 AND cont_seg_uni = 9 AND cont_seg_dec = 5 THEN
				IF cont_min_uni = "1001" THEN
					cont_min_uni <= "0000";
				ELSE
					cont_min_uni <= cont_min_uni + 1;
				END IF;
			END IF;
		END IF;
	END PROCESS;

	PROCESS (inicio, clk)
	BEGIN
		IF inicio = '1' THEN
			cont_min_dec <= "0000";
		ELSIF rising_edge(clk) THEN
			IF cont_base = 100000000 AND cont_seg_uni = 9 AND cont_seg_dec = 5 AND cont_min_uni = 9 THEN
				IF cont_min_dec = 5 THEN
					cont_min_dec <= "0000";
				ELSE
					cont_min_dec <= cont_min_dec + 1;
				END IF;
			END IF;
		END IF;
	END PROCESS;

	PROCESS (clk, inicio)
	BEGIN
```

```vhdl
		IF inicio = '1' THEN
			enable_seg_aux <= "1110";
		ELSIF rising_edge(clk) THEN
			IF cont_enable_aux = 100000 THEN--100000 then
				enable_seg_aux <= enable_seg_aux(2 DOWNTO 0) & enable_seg_aux(3);
			END IF;
		END IF;
	END PROCESS;

	PROCESS (inicio, clk)
	BEGIN
		IF inicio = '1' THEN
			cont_enable_aux <= 0;
		ELSIF rising_edge(clk) THEN
			IF cont_enable_aux = 100000 THEN--100000 then
				cont_enable_aux <= 0;
			ELSE
				cont_enable_aux <= cont_enable_aux + 1;
			END IF;
		END IF;
	END PROCESS;

	relojData <= cont_min_dec & cont_min_uni & cont_seg_dec & cont_seg_uni;

	-----------------------------------------------mux datos
	PROCESS (inicio, clk, sel)
	BEGIN
		CASE(sel) IS

			WHEN "001" => muxSensores <= "00000" & TempData;
			WHEN "010" => muxSensores <= distData;
			WHEN "011" => muxSensores <= hallData;--hallData;

			WHEN "100" => muxSensores <= STD_LOGIC_VECTOR(to_unsigned(cont_riesgo, 14));
			WHEN "101" => muxSensores <= STD_LOGIC_VECTOR(to_unsigned(rTemp, 14));
			WHEN "110" => muxSensores <= STD_LOGIC_VECTOR(to_unsigned(rDist, 14));
			WHEN "111" => muxSensores <= STD_LOGIC_VECTOR(to_unsigned(rHall, 14));
			WHEN OTHERS => muxSensores <= "00000000000000";
		END CASE;
	END PROCESS;
	---------------------------------------------- conv bin2bcd
	PROCESS (clk, inicio)
	BEGIN
		IF inicio = '1' THEN
			cont <= 0;
		ELSIF rising_edge(clk) THEN
			IF cont = 100000000 THEN
				cont <= 0;
			ELSE
				cont <= cont + 1;
			END IF;
		END IF;
	END PROCESS;

	PROCESS (clk, inicio)
	BEGIN
		IF inicio = '1' THEN
			vector_aux <= "0000000000000000" & muxSensores;
			estado <= "00";
			cont_bits <= 0;
			unidades <= "0011";
			decenas <= "0011";
			centenas <= "0011";
			millares <= "0011";
			fin <= '0';
		ELSIF rising_edge(clk) THEN
			--if cont=0 then
			CASE estado IS
				WHEN "00" => cont_bits <= 0;
					fin <= '0';
					vector_aux <= "0000000000000000" & muxSensores;
					IF enableSw = '1' THEN
```

```vhdl
                                                estado <= "01";
                                        END IF;
                                WHEN "01" => cont_bits <= cont_bits + 1;
                                        fin <= '0';
                                        vector_aux <= vector_aux(28 DOWNTO 0) & '0';
                                        IF cont_bits < 13 THEN
                                                estado <= "10";
                                        ELSE
                                                estado <= "11";
                                        END IF;
                                WHEN "10" => cont_bits <= cont_bits;
                                        fin <= '0';
                                        IF vector_aux(21 DOWNTO 18) > 4 THEN
                                                vector_aux(21 DOWNTO 18) <= vector_aux(21 DOWNTO 18) + "0011";
                                        END IF;
                                        IF vector_aux(17 DOWNTO 14) > 4 THEN
                                                vector_aux(17 DOWNTO 14) <= vector_aux(17 DOWNTO 14) + "0011";
                                        END IF;
                                        IF vector_aux(25 DOWNTO 22) > 4 THEN
                                                vector_aux(25 DOWNTO 22) <= vector_aux(25 DOWNTO 22) + "0011";
                                        END IF;
                                        IF vector_aux(29 DOWNTO 26) > 4 THEN
                                                vector_aux(29 DOWNTO 26) <= vector_aux(29 DOWNTO 26) + "0011";
                                        END IF;
                                        estado <= "01";
                                WHEN "11" => cont_bits <= 0;
                                        fin <= '1';
                                        decenas <= vector_aux(21 DOWNTO 18);
                                        unidades <= vector_aux(17 DOWNTO 14);
                                        centenas <= vector_aux(25 DOWNTO 22);
                                        millares <= vector_aux(29 DOWNTO 26);
                                        estado <= "00";
                                WHEN OTHERS => cont_bits <= 0;
                                        fin <= '0';
                                        estado <= "00";
                        END CASE;
                END IF;
                --end if;
END PROCESS;--final de decodificador BCD-7seg

bcdData <= millares & centenas & decenas & unidades;

----------------------------------------------mux reloj
PROCESS (sel)
BEGIN
        IF (sel = "000") THEN
                to7seg <= relojData;
        ELSE
                to7seg <= bcdData;
        END IF;
END PROCESS;

----------------------------------------------process para 7seg
--mux 7seg
PROCESS (enable_seg_aux)
BEGIN
        CASE enable_seg_aux IS
                WHEN "1110" => salida_ver <= to7seg(3 DOWNTO 0);
                WHEN "1101" => salida_ver <= to7seg(7 DOWNTO 4);
                WHEN "1011" => salida_ver <= to7seg(11 DOWNTO 8);
                WHEN "0111" => salida_ver <= to7seg(15 DOWNTO 12);
                WHEN OTHERS => salida_ver <= "0000";
        END CASE;
END PROCESS;

enable_seg <= enable_seg_aux;

--bcd2 7seg
PROCESS (salida_ver)
BEGIN
        CASE salida_ver IS
                WHEN "0000" => segmentos <= "0000001";
```


```
                    WHEN "0001" => segmentos <= "1001111";
                    WHEN "0010" => segmentos <= "0010010";
                    WHEN "0011" => segmentos <= "0000110";
                    WHEN "0100" => segmentos <= "1001100";
                    WHEN "0101" => segmentos <= "0100100";
                    WHEN "0110" => segmentos <= "1100000";
                    WHEN "0111" => segmentos <= "0001111";
                    WHEN "1000" => segmentos <= "0000000";
                    WHEN "1001" => segmentos <= "0001100";
                    WHEN OTHERS => segmentos <= "0110000";
          END CASE;
 END PROCESS;

END Behavioral;
```

Constraints

```
----------------------------CONSTRAINS------------------------
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to
the top level signal names in the project

## Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports clk]

 set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets paro_IBUF]

### Switches
set_property PACKAGE_PIN V17 [get_ports {pwmEnt[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {pwmEnt[0]}]
set_property PACKAGE_PIN V16 [get_ports {pwmEnt[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {pwmEnt[1]}]
set_property PACKAGE_PIN W16 [get_ports {pwmEnt[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {pwmEnt[2]}]
set_property PACKAGE_PIN W17 [get_ports {pwmEnt[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {pwmEnt[3]}]
#set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN W14 [get_ports {sw[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W13 [get_ports {sw[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN V2 [get_ports {sw[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
#set_property PACKAGE_PIN T3 [get_ports {sel[1]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {sel[1]}]
set_property PACKAGE_PIN T2 [get_ports {modo[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {modo[0]}]
set_property PACKAGE_PIN R3 [get_ports {modo[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {modo[1]}]
#set_property PACKAGE_PIN W2 [get_ports {modo[1]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {modo[1]}]
set_property PACKAGE_PIN U1 [get_ports {paro}]
set_property IOSTANDARD LVCMOS33 [get_ports {paro}]
set_property PACKAGE_PIN T1 [get_ports {enableSwS}]
set_property IOSTANDARD LVCMOS33 [get_ports {enableSwS}]
set_property PACKAGE_PIN R2 [get_ports {enableSw}]
set_property IOSTANDARD LVCMOS33 [get_ports {enableSw}]


## LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
set_property PACKAGE_PIN V3 [get_ports {led[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[9]}]
set_property PACKAGE_PIN W3 [get_ports {led[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[10]}]
set_property PACKAGE_PIN U3 [get_ports {led[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[11]}]
set_property PACKAGE_PIN P3 [get_ports {led[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[12]}]
set_property PACKAGE_PIN N3 [get_ports {led[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[13]}]
set_property PACKAGE_PIN P1 [get_ports {led[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[14]}]
set_property PACKAGE_PIN L1 [get_ports {led[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[15]}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {segmentos[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segmentos[6]}]
set_property PACKAGE_PIN W6 [get_ports {segmentos[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segmentos[5]}]
set_property PACKAGE_PIN U8 [get_ports {segmentos[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segmentos[4]}]
set_property PACKAGE_PIN V8 [get_ports {segmentos[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segmentos[3]}]
set_property PACKAGE_PIN U5 [get_ports {segmentos[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segmentos[2]}]
set_property PACKAGE_PIN V5 [get_ports {segmentos[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segmentos[1]}]
set_property PACKAGE_PIN U7 [get_ports {segmentos[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segmentos[0]}]

#set_property PACKAGE_PIN V7 [get_ports dp]
#set_property IOSTANDARD LVCMOS33 [get_ports dp]

##Pmod Header JB
##Sch name = JB1
set_property PACKAGE_PIN A14 [get_ports {alarma}]
set_property IOSTANDARD LVCMOS33 [get_ports {alarma}]
##Sch name = JB2
set_property PACKAGE_PIN A16 [get_ports {cale}]
set_property IOSTANDARD LVCMOS33 [get_ports {cale}]
##Sch name = JB3
set_property PACKAGE_PIN B15 [get_ports {dir}]
set_property IOSTANDARD LVCMOS33 [get_ports {dir}]
##Sch name = JB4
set_property PACKAGE_PIN B16 [get_ports {pwmPos}]
set_property IOSTANDARD LVCMOS33 [get_ports {pwmPos}]
##Sch name = JB7
#set_property PACKAGE_PIN A15 [get_ports {JB[4]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {JB[4]}]
##Sch name = JB8
set_property PACKAGE_PIN A17 [get_ports {step}]
set_property IOSTANDARD LVCMOS33 [get_ports {step}]
```

```
##Sch name = JB9
set_property PACKAGE_PIN C15 [get_ports {enableSteper}]
set_property IOSTANDARD LVCMOS33 [get_ports {enableSteper}]
##Sch name = JB10
set_property PACKAGE_PIN C16 [get_ports {pwmNeg}]
set_property IOSTANDARD LVCMOS33 [get_ports {pwmNeg}]

##Pmod Header JC
##Sch name = JC1
set_property PACKAGE_PIN K17 [get_ports {dcEnc[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dcEnc[0]}]
##Sch name = JC2
set_property PACKAGE_PIN M18 [get_ports {echo}]
set_property IOSTANDARD LVCMOS33 [get_ports {echo}]
##Sch name = JC3
set_property PACKAGE_PIN N17 [get_ports {fc1Calle}]
set_property IOSTANDARD LVCMOS33 [get_ports {fc1Calle}]
##Sch name = JC4
set_property PACKAGE_PIN P18 [get_ports {crc_en}]
set_property IOSTANDARD LVCMOS33 [get_ports {crc_en}]
##Sch name = JC7
set_property PACKAGE_PIN L17 [get_ports {dcEnc[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dcEnc[1]}]
##Sch name = JC8
set_property PACKAGE_PIN M19 [get_ports {trigger}]
set_property IOSTANDARD LVCMOS33 [get_ports {trigger}]
##Sch name = JC9
set_property PACKAGE_PIN P17 [get_ports {fc2Tarj}]
set_property IOSTANDARD LVCMOS33 [get_ports {fc2Tarj}]
##Sch name = JC10
set_property PACKAGE_PIN R18 [get_ports {ds_data_bus}]
set_property IOSTANDARD LVCMOS33 [get_ports {ds_data_bus}]


set_property PACKAGE_PIN U2 [get_ports {enable_seg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {enable_seg[0]}]
set_property PACKAGE_PIN U4 [get_ports {enable_seg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {enable_seg[1]}]
set_property PACKAGE_PIN V4 [get_ports {enable_seg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {enable_seg[2]}]
set_property PACKAGE_PIN W4 [get_ports {enable_seg[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {enable_seg[3]}]


###Buttons
set_property PACKAGE_PIN U18 [get_ports inicio]
set_property IOSTANDARD LVCMOS33 [get_ports inicio]
set_property PACKAGE_PIN T18 [get_ports btnU]
set_property IOSTANDARD LVCMOS33 [get_ports btnU]
set_property PACKAGE_PIN W19 [get_ports btnL]
set_property IOSTANDARD LVCMOS33 [get_ports btnL]
set_property PACKAGE_PIN T17 [get_ports btnR]
set_property IOSTANDARD LVCMOS33 [get_ports btnR]
set_property PACKAGE_PIN U17 [get_ports btnD]
set_property IOSTANDARD LVCMOS33 [get_ports btnD]
```