UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENERIA Y TECNOLOGIAS AVANZADAS

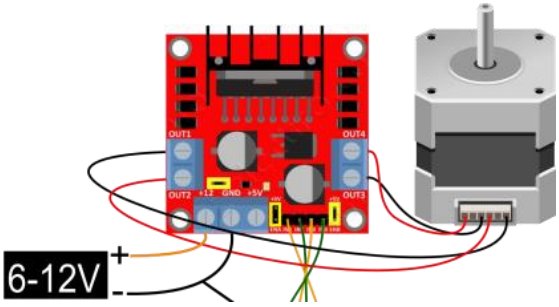
“Proyecto RS232 para control de posición y velocidad de un motor a pasos”

ALUMNO: ZARAZUA AGUILAR LUIS FERNANDO

GRUPO: 2MM9

PROFESOR: RODRÍGUEZ FUENTES MIGUEL ÁNGEL

MATERIA: DISPOSITIVOS LÓGICOS PROGRAMABLES



Planteamiento del Problema

Se requiere controlar por medio de la NEXYS 2 un motor a pasos que nos indique su posición cada vez que da un paso por medio de comunicación UART y que a este se le pueda controlar su posición por medio de comandos R y L para el sentido y un número del 0 al 20 para avanzar determinados pasos ingresados desde una interfaz con comunicación serial. Además se puede configurar la velocidad con el comando VXX donde XX representa un número del 0 a 10.

En la realización del Motor a Pasos controlado vía serial se implementaron 9 paquetes para la resolución del problema teniendo como objetivo separar en pequeños problemas que permitan procesar adecuadamente la problemática.

Código Principal que une todos los archivos

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
use work.costal.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
entity Principal is
       Port ( clk_nexys : in  STD_LOGIC;
              RX_Data : in  STD_LOGIC;
                 Data_Tx : out  STD_LOGIC;
              Sal_Motor: out  STD_LOGIC_VECTOR (3 downto 0);
              Salidas_7segc : out  STD_LOGIC_VECTOR (7 downto 0);
                 Control_Disp_7segc: out  STD_LOGIC_VECTOR (3 downto 0));

end Principal;

architecture Behavioral of Principal is
signal ready_xd,fec: std_logic;
signal rx_datac: std_logic_vector(7 downto 0);
signal Data_0 : STD_LOGIC_VECTOR (7 downto 0);
signal Data_1 : STD_LOGIC_VECTOR (7 downto 0);
signal Data_2 : STD_LOGIC_VECTOR (7 downto 0);
signal Datos_Rxc : STD_LOGIC_VECTOR (7 downto 0);
signal Display 1c,Display 2c,Display 3c,Display 4c: STD LOGIC VECTOR (7 downto 0);
signal dsalida :  STD_LOGIC;
signal auxclk1,auxclk2,auxclk3: std_logic;
signal Act_Pasosc,Giroc:  STD_LOGIC;
signal Pasosc :  STD_LOGIC_VECTOR (4 downto 0);
signal Velocidadc :  STD_LOGIC_VECTOR (3 downto 0);
signal Sal_Posicionc : STD_LOGIC_VECTOR (4 downto 0);
signal Inicioc,tdrec: std logic;
signal tx datac: std logic vector(7 downto 0);
signal Sal Motorc:  STD LOGIC VECTOR (3 downto 0);
constant Var logica: std logic vector(3 downto 0):=X"2";
begin
     Sal_Motor<=Sal_Motorc;
     Display_1c<=(auxclk1&auxclk2&auxclk3&"0"&Sal_Motorc);
      Display_2c<=Data_1;--rx_datac;
      Display_3c<="000"&Sal_Posicionc;--Data_2;--Data_0;
      Display_4c<="0000"&Velocidadc;--rx_datac;--Datos_Rxc;
      --bloque Transmision Serial
u1: uartrx port map( rxd => RX Data,--Entrada serial de 1 bit.
                     clk => clk nexys,--Reloj de entrada.
                     clr => Var_logica(0),--Cero_logico en el clr.
                     rdrf_clr => Var_logica(0),--Cero lógico en el clear de bit de recepción.
                     rdrf => ready_xd,--Señal de recepción de reloj.
                     fe => fec,--Error en el frame.
                     rx_data => rx_datac);--Salida que indica en que posición se encuentra(8 bits).
    --bloque de Memoria de 3 bytes
u2: Almc_FFD port map(clk => ready_xd,--Reloj para actualizar Flip Flop D.
                      condicional => fec,--Condicional si hay error en el dato.
                      Din => rx datac,--Dato de entrada (Byte).
                      D2 => Data 2,--Tercer Dato Recibido.
                      D1 => Data_1,--Segundo Dato Recibido.
                      D0 => Data_0,--Primer Dato Recibido.
                      dout => dsalida);

u3: Decodificador_Serial port map( clk => clk_nexys,--Reloj de la Nexys.
                                   clr => Var_logica(0),--Clear.
                                   Lectura => ready_xd,--Bit de inicio de Lectura de datos.
                                   Reg1 => Data 2,--Byte de Unidades.
                                   Reg2 => Data 1,--Byte de Decenas.
                                   Reg3 => Data 0,--Byte de Opción.
                                   Giro => Giroc,--Bit de salida que indica el giro.
                                   act_pasos => Act_Pasosc ,--Señal de guardado de pasos(Salida).
                                   Pasos=> Pasosc,--Salida de Número de pasos a avanzar (5 bits).
                                   Sal_Velocidad=> Velocidadc,--Salida de 4 bits para elegir la velocidad.
                                          Datos=> Datos_Rxc);--Datos recibidos.
    --Circuito de muestreo de Leds
u4: Leds_Display_7 port map( clkin => clk_nexys,
                             Entrada Disp 1 => Display 1c,
                             Entrada Disp 2 => Display 2c,
                             Entrada Disp 3 => Display 3c,
                             Entrada_Disp_4 => Display_4c,
                             Salidas_7seg => Salidas_7segc,
                             Control_Disp_7seg => Control_Disp_7segc);
```

```
    --bloque divisor
u5: Selector_Velocidad port map( clk => clk_nexys,--Reloj de la Nexys.
                                 Selector => Velocidadc,--Entrada de 4 bits para elegir la velocidad.
                                 clksal => auxclk1);--Reloj de salida con la frecuencia requerida.
    --bloque comparador para saber cuantos pasos queremos.
u6: Contador_Comparador port map( clk=> auxclk1,--Reloj de entrada con la frecuencia requerida.
                                  Numero=> Pasosc,--Entrada de Número de pasos a avanzar (5 bits).
                                  Actualizar=> Act Pasosc,--Señal de guardado de pasos (Entrada).
                                  clksal=>auxclk2,--Reloj de salida con la frecuencia requerida y los pasos deseados(TX1).
                                  clksal2=>auxclk3);--Reloj de salida con la frecuencia requerida y los pasos deseados (TX10).
      --bloque Driver_Motor
u7: Driver_Motor port map( clk_M =>auxclk2,--Reloj del Contador del Motor.
                           clk_P =>auxclk3,--Reloj del contador de Posición
                          --clk =>auxclk2,--Reloj de entrada con la frecuencia requerida y los pasos deseados.
                          selector =>Giroc,--Bit de entrada que indica el giro.
                          reset =>Var_logica(0),--Reset del contador en 0.
                          Salida =>Sal Motorc,--Salida al motor.
                          Cuenta =>Sal Posicionc);--Salida que indica en que posición se encuentra el motor.

      --bloque Transmision Serial
u8: Decodificador_Tx_Serial port map( clk => clk_nexys,--Reloj de la Nexys.
                                      clr => Var_logica(0),--Clear del decodificador en 0.
                                     Enviar => auxclk3,--Reloj de entrada con la frecuencia requerida y los pasos deseados.
                                     Entrada => Sal_Posicionc,--Entrada que indica en que posición se encuentra el motor.
                                     tdrel => tdrec,--Señal que indica que se puede transmitir otra vez.
                                     Registro_Tx => tx_datac,--Registro que manda el byte.
                                     Inicio => Inicioc);--Señal que indica cuando puede empezar a transmitir.
    --bloque de Memoria de 3 bytes
u9: Transmisor Serial port map( clk => clk nexys,--Señal de reloj de entrada.
                                clr => Var_logica(0),--Clear del transmisor serial.
                                tx_data  => tx_datac,--Registro de donde se toma el byte.
                                outp => Inicioc,--Señal que indica cuando puede empezar a transmitir (Activa la transmisiòn en alto).
                                tdre => tdrec,--Bit que indica que se envió el dato completo o que el dispositivo puede transmitir(1).
                                txd => Data_Tx);--Salida del transmisor serial.
end Behavioral;
```

Este bloque se encarga de conectar los circuitos diseñados para que en conjunto puedan controlar adecuadamente el motor a pasos teniendo la filosofía de dividir el problema en sub-problemas.

Código de Recepción UART

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

entity uartrx is
    Port ( rxd : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           clr : in  STD_LOGIC;
           rdrf_clr : in  STD_LOGIC;
           rdrf : out  STD_LOGIC;
           fe : out  STD_LOGIC;
           rx_data : out  STD_LOGIC_VECTOR (7 downto 0));
end uartrx;

architecture Behavioral of uartrx is
type state_type is (mark,start,delay,shift,stop);
signal state: state_type;
signal rxbuff: std_logic_vector(7 downto 0);
signal baud_count: std_logic_vector(11 downto 0);
signal bit_count: std_logic_vector(3 downto 0);
constant bit_time: std_logic_vector(11 downto 0):=X"A28";
constant half_bit_time: std_logic_vector(11 downto 0):=X"514";
signal clkdiv: std_logic_vector(0 downto 0);
signal contador: std_logic_vector(3 downto 0):="0000";
begin
    process(clk)
    begin
        if rising_edge(clk) then
            clkdiv <= clkdiv +1;
        end if;
    end process;

    process(clkdiv(0),clr,rdrf_clr)
    begin
        if clr ='1' then--Si el reset esta activo carga configuración inicial.
            state <= mark;--Regresa al estado de mark.
            rxbuff <="00000000";--Carga el buffer con 0.
            baud_count <=x"000";--Contador de clk para los bauds en 0.
            bit_count <= "0000";--Contador de bits en 0.
            rdrf <= '0';--Bit de recepción con 0.
            fe <= '0';--Bit de error de frame en 0.
        elsif rdrf clr = '1' then--Si hay un clr en rdrf clr.
            rdrf <= '0';--Resetea a rdrf.
        elsif rising_edge(clkdiv(0)) then--Elige como reloj de la máquina de estados a clkdiv.
            case state is
                when mark =>--Estado de espera de dato.
                    baud_count <=x"000";--Contador de clk para los bauds en 0.
                    bit_count <= "0000";--Contador de bits en 0.
                    rdrf <='0';
                        if rxd = '1' then--Si esta en estado de reposos.
                            state <= mark;--Continua en el estado de espera.
                        else--En caso contrario.
                            state <= start;--Pasa a estado de inicio de detección de bits.
                            fe <= '0';--Bit de error de frame en 0.
                        end if;

                when start =>--Estado de inicio.
                    if baud_count >= half_bit_time then--Si el conteo de bauds es mayor a half_bit_time lee el dato.
                        baud count <= x"000";--Contador de clk para los bauds en 0.
                        state <= delay;--Pasa al estado de delay.
                    else--En caso contrario sigue contando.
                        baud_count <= baud_count + 1;--Incrementa en 1 baud_count para seguir con el conteo.
                        state <= start;--Continua en el estado de inicio.
                    end if;

                when delay =>--Estado de delay(Tiempo de espera hasta completar la trama).
```

```
                if baud_count>= bit_time then--Si baud_count es mayor a bit time lee el bit.
                    baud_count <= x"000";--Contador de clk para los bauds en 0.
                        if bit_count < "1000" then--Si no han completado los 8 bits.
                            state<= shift;--Pasa al estado de shift.
                        else--Si se completo la cuenta de bits carga a rx_data con lo que hay en el buffer.
                            rx_data <= rxbuff;--Carga a rx_data con lo que hay en el buffer.
                            state <= stop;--Pasa al estado de stop.
                        end if;
                else
                    baud_count <= baud_count +1;--Incrementa en 1 baud_count para seguir con el conteo.
                    state <= delay;--Continua en el estado de delay.
                end if;

            when shift =>--Estado de recorrimiento.
                rxbuff(7) <= rxd;--Carga al bit 7 lo que hay en el pin rxd.
                rxbuff(6 downto 0) <= rxbuff(7 downto 1);--Realiza un recorrimiento de bits.
                bit count <= bit count + 1;--Incrementa en 1 baud count para seguir con el conteo.
                state <= delay;--Pasa al estado de delay.

            when stop =>--Estado de reconocimiento de bit de parada.
                rdrf <= '1';--Bit de recepción con 1.
                    if rxd = '0' then--Si el bit de parada en 0 hay un error en la trama.
                        fe <= '1';--Bit de error de trama con 1 indicando que hay un error en la trama.
                    else--Si el bit de parada es 1 la trama es correcta.
                        fe <= '0';--Bit de error de trama con 0 indicando que la trama es correcta.
                    end if;
                if contador<=9 then
                 contador<=contador+1;
                 state <=stop;--Pasa al estado de mark(estado de reposo o espera).
                else
                 contador<="0000";
                 state <= mark;--Pasa al estado de mark(estado de reposo o espera).
                end if;
        end case;
    end if;
  end process;
end Behavioral;
```

La función de este bloque es de las importantes del código ya que se encarga de recibir serialmente el dato a una determinada frecuencia y almacenarlo en un registro de 8 bits para su posterior lectura, además cuenta con entradas de control que facilitan la manipulación de los datos de entrada. En este programa se usó el bit "rdrf" para actualizar el registro de 3 Flip Flops D que nos permite analizar correctamente la información.

Código de Almacenamiento en el Flip Flop D

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity Almc_FFD is
    Port ( clk : in  STD_LOGIC;--Reloj para actualizar Flip Flop D
           condicional : in  STD_LOGIC;--Condicional si hay error en el dato
           Din : in  STD_LOGIC_VECTOR (7 downto 0);--Dato de entrada (Byte).
           D2 : out  STD_LOGIC_VECTOR (7 downto 0);--Tercer Dato Recibido.
           D1 : out  STD_LOGIC_VECTOR (7 downto 0);--Segundo Dato Recibido.
           D0 : out  STD_LOGIC_VECTOR (7 downto 0);--Primer Dato Recibido.
           dout : out  STD_LOGIC);--Señal que indica que todos los datos fueron recibidos.
end Almc_FFD;

architecture Behavioral of Almc_FFD is
signal permiso: STD_LOGIC;
signal Flancos: STD_LOGIC_VECTOR (3 downto 0):=x"0";
signal XD0,XD1,XD2: STD_LOGIC_VECTOR (7 downto 0):=x"00";
begin
    process(clk,condicional,Din,permiso,XD0,XD1,XD2)
     begin
           --if (rising_edge(clk) or falling_edge(clk)) then
            if (rising_edge(clk)) then
                if condicional='0' then--Si no hay error recibe datos.
                    XD0<=XD1;
                    XD1<=XD2;
                    XD2<=Din;
                    if Flancos<x"4" then
                    Flancos<=Flancos+1;
                   else
                    Flancos<=x"0";
                   end if;
                    if Flancos=x"3" then
                     permiso<='1';
                    else
                    permiso<='0';
                   end if;
                end if;
          end if;
           D0<=XD0;
           D1<=XD1;
           D2<=XD2;
           dout<=permiso and clk;
     end process;
end Behavioral;
```

Este bloque se encarga de ir recorriendo de byte en byte los datos para que posteriormente un decodificador pueda determinar si es una combinación válida.

Código de Decodificación de la entrada Serial

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity Decodificador_Serial is
    Port ( clk : in  STD_LOGIC;--Reloj de la Nexys.
           clr : in  STD_LOGIC;--Clear.
           Lectura : in  STD_LOGIC;--Bit de inicio de Lectura de datos.
           Reg1 : in  STD_LOGIC_VECTOR (7 downto 0);--Byte de Unidades.
           Reg2 : in  STD_LOGIC_VECTOR (7 downto 0);--Byte de Decenas.
           Reg3 : in  STD_LOGIC_VECTOR (7 downto 0);--Byte de Opcion.
           Giro : out  STD_LOGIC;--Bit que indica el giro.
           act_pasos : out  STD_LOGIC;--Se□de guardado de pasos.
           --act_vel : out  STD_LOGIC;--Se□de guardado de velocidad.
           Pasos : out  STD_LOGIC_VECTOR (4 downto 0);--Numero de pasos a avanzar.
           Sal_Velocidad : out  STD_LOGIC_VECTOR (3 downto 0);
              Datos: out  STD_LOGIC_VECTOR (7 downto 0));--Contenido de los registros.
end Decodificador_Serial;

architecture Behavioral of Decodificador_Serial is
type state_type is (Mark,Inicio,Giro_R,Giro_L,Velocidad,Determinar_Pasos,Cargar_Pasos,Apagar_clk,Determinar_Velocidad,Cargar_Velocidad);
signal state: state_type;
signal G:std_logic;
signal clkdiv: std_logic_vector(1 downto 0);
signal Regs: std_logic_vector(7 downto 0);
signal contador: std_logic_vector(7 downto 0);
--signal Decenas: std_logic_vector(7 downto 0);
begin
     process(clk)
      begin
        if rising_edge(clk) then
            clkdiv <= clkdiv +1;
        end if;
       end process;

     process(clkdiv(1),clr)
      begin
        if clr ='1' then
            state <= mark;
            act_pasos<='0';
        elsif rising_edge(clkdiv(1)) then
            case state is
              --Estado Mark o de Espera
               when Mark =>
                     act_pasos<='0';
                       if Lectura = '0' then
                           state<=Mark;
                           Datos<=Regs;
                       else
                           state<=Inicio;
                       end if;
               --Estado Inicio
               when Inicio =>
                 if Lectura='1' then
                    state<=Inicio;
                 else
                  if Reg3=x"52" then  --Letra R
                    state<=Giro_R;
                  elsif Reg3=x"4C" then  --Letra L
                    state<=Giro_L;
                  elsif Reg3=x"56" then --Letra V
                    state<=Velocidad;
                   else
                     state<=Mark;
                  end if;
                end if;
               --Estado Giro_R
               when Giro_R =>
                     G<='1';
                      state<=Determinar_Pasos;
               --Estado Giro L
              when Giro_L =>
                     G<='0';
                      state<=Determinar Pasos;
               --Estado Determinar_Pasos
              when Determinar_Pasos =>
                     if ((Reg1>47 and Reg1<58) and (Reg2>47 and Reg2<50)) or (Reg2=50 and Reg1=48) then-- Digitos del 00-19
                         state<=Cargar_Pasos;
                           if Reg2=48 then
                            Regs<=(Reg1-"00110000");
                           elsif Reg2=49 then
                            Regs<=((Reg1-"00110000")+x"0A");
                             elsif Reg2=50 then
                              Regs<=x"14";
                           end if;
                      else
                          state<=Mark;
                      end if;
               --Estado Cargar Pasos
               when Cargar_Pasos =>
                     Pasos<=Regs(4 downto 0);
                     Giro<=G;
                     act_pasos<='1';
                      if contador<=240 then
                         contador<=contador+1;
                         state <=Cargar_Pasos;
                      else
                         contador<="00000000";
                         state<=Apagar_clk;--Pasa al estado de Apagar_clk.
                      end if;
               --Estado Apagar clk
               when Apagar_clk =>
                     act_pasos<='0';
                     state<=Mark;
              --Estado Velocidad
```

```
            when Velocidad =>
                 state<=Determinar_Velocidad;
             --Estado Determinar_Velocidad
            when Determinar_Velocidad =>
                 if ((Reg1>47 and Reg1<58) and (Reg2=48)) or (Reg2=49 and Reg1=48) then-- Digitos del 00-10
                      state<=Cargar_Velocidad;
                        if Reg2=48 then
                      Regs<=(Reg1-"00110000");
                      elsif Reg2=49 then
                      Regs<=(Reg1-"00110000")+x"0A";
                      end if;
                 else
                      state<=Mark;
                 end if;
            --Estado Determinar Velocidad
             when Cargar_Velocidad =>
              Sal Velocidad<=Regs(3 downto 0);
                  state<=Mark;
          end case;
       end if;
    end process;
end Behavioral;
```

Este código es el que se encarga de leer los 3 bytes almacenados en los Flip Flop's D y entrega una salida decodificada según lo que se tenga y en caso que sea una combinación inválida simplemente ignora los datos y regresa a su estado de Inicio (Lectura), esta decodificación se logra por medio de una máquina de estados que primero detecta si el primer byte es una letra válida como la "R", "L" o "V", en caso que si lo sea, entonces procede a verificar si los dígitos son numéricos y están dentro del rango de modificación, para la "R" y "L" aceptan hasta el 20, y el caso de la "V", solo se acepta hasta el 10. En el caso de que se mande "R" o "L" además modifica un registro que indica el giro para el motor.

Una vez identificados los dígitos guarda en hexadecimal el número en su registro correspondiente ya sea de Velocidad (4 bits) o de Pasos a avanzar (5 bits), posteriormente prende un bit (que apaga en el estado de Inicio) para que el dato pueda ser procesado solo una vez por el circuito que le procede.

Código de Displays Multiplexados

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

entity Leds_Display_7 is
    Port ( clkin : in  STD_LOGIC;
           Entrada_Disp_1 : in  STD_LOGIC_VECTOR (7 downto 0);
           Entrada_Disp_2 : in  STD_LOGIC_VECTOR (7 downto 0);
           Entrada_Disp_3 : in  STD_LOGIC_VECTOR (7 downto 0);
           Entrada_Disp_4 : in  STD_LOGIC_VECTOR (7 downto 0);
           Salidas_7seg : out  STD_LOGIC_VECTOR (7 downto 0);
           Control_Disp_7seg : out  STD_LOGIC_VECTOR (3 downto 0));
end Leds_Display_7;

architecture Behavioral of Leds_Display_7 is
signal clkdiv: STD_LOGIC_VECTOR (16 downto 0);
signal contador_disp: STD_LOGIC_VECTOR (1 downto 0):="00";
begin
     process(clkin)
      begin
        if rising_edge(clkin) then
            clkdiv <= clkdiv +1;
        end if;
      end process;
      process(clkdiv(16),contador_disp)
      begin
      if rising_edge(clkdiv(16)) then
         contador_disp<=contador_disp+1;
         if contador_disp=0 then
             Control_Disp_7seg<="0111";
          Salidas_7seg<=not(Entrada_Disp_1);
         elsif contador_disp=1 then
             Control_Disp_7seg<="1011";
              Salidas_7seg<=not(Entrada_Disp_2);
         elsif contador_disp=2 then
             Control_Disp_7seg<="1101";
              Salidas_7seg<=not(Entrada_Disp_3);
         else
             Control_Disp_7seg<="1110";
              Salidas_7seg<=not(Entrada_Disp_4);
         end if;
       end if;
      end process;
end Behavioral;
```

Este código es el que se encarga de poder multiplexar los led's en los displays que se tienen a la salida para poder mostrar que los datos que se reciben son correctos y en qué posición se está, este es el único código opcional que tiene el programa.

Código para la Selección de la Velocidad

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

entity Selector_Velocidad is
    Port ( clk : in  STD_LOGIC;
           Selector : in  STD_LOGIC_VECTOR (3 downto 0);
           clksal : out  STD_LOGIC);
end Selector_Velocidad;

architecture Behavioral of Selector_Velocidad is
signal q: STD_LOGIC_VECTOR (23 downto 0):=X"000000";--01001110001000000=20000=>1seg.
--signal Opcion: STD_LOGIC_VECTOR (3 downto 0):=X"0";--Opción inicial en 0.
signal tiempo_medio: STD_LOGIC_VECTOR (23 downto 0);
signal tiempo_completo: STD_LOGIC_VECTOR (23 downto 0);
constant tiempo_medio_0: STD_LOGIC_VECTOR (23 downto 0):=X"150000";--1seg
constant tiempo_medio_1: STD_LOGIC_VECTOR (23 downto 0):=X"120000";--0.75seg
constant tiempo_medio_2: STD_LOGIC_VECTOR (23 downto 0):=X"100000";--0.5seg
constant tiempo_medio_3: STD_LOGIC_VECTOR (23 downto 0):=X"0F1000";--0.4seg
constant tiempo_medio_4: STD_LOGIC_VECTOR (23 downto 0):=X"0E0500";
constant tiempo_medio_5: STD_LOGIC_VECTOR (23 downto 0):=X"0D0000";
constant tiempo_medio_6: STD_LOGIC_VECTOR (23 downto 0):=X"0C0000";
constant tiempo_medio_7: STD_LOGIC_VECTOR (23 downto 0):=X"0A0000";
constant tiempo_medio_8: STD_LOGIC_VECTOR (23 downto 0):=X"090000";
constant tiempo_medio_9: STD_LOGIC_VECTOR (23 downto 0):=X"085000";
constant tiempo_medio_10: STD_LOGIC_VECTOR (23 downto 0):=X"080000";
begin
     --Obtención del tiempo medio.
     process(Selector)
      begin
          if Selector="0000" then tiempo_medio<=tiempo_medio_0;--Primer tiempo a elegir.
          elsif Selector="0001" then tiempo_medio<=tiempo_medio_1;--Segundo tiempo a elegir.
          elsif Selector="0010" then tiempo_medio<=tiempo_medio_2;--Tercer tiempo a elegir.
            elsif Selector="0011" then tiempo_medio<=tiempo_medio_3;--Cuarto tiempo a elegir.
          elsif Selector="0100" then tiempo_medio<=tiempo_medio_4;--Quinto tiempo a elegir.
          elsif Selector="0101" then tiempo_medio<=tiempo_medio_5;--Sexto tiempo a elegir.
            elsif Selector="0110" then tiempo_medio<=tiempo_medio_6;--Séptimo tiempo a elegir.
          elsif Selector="0111" then tiempo_medio<=tiempo_medio_7;--Octavo tiempo a elegir.
          elsif Selector="1000" then tiempo_medio<=tiempo_medio_8;--Noveno tiempo a elegir.
            elsif Selector="1001" then tiempo_medio<=tiempo_medio_9;--Décimo tiempo a elegir.
            else tiempo_medio<=tiempo_medio_10;--Onceavo tiempo a elegir.
             end if;
    tiempo_completo<=tiempo_medio+tiempo_medio;
     end process;
     --Obtención de la señal de reloj segun el tiempo seleccionado.
     process(clk,tiempo_completo,tiempo_medio,q)
      begin
         if q>tiempo_completo then
              q <= (others => '0');
         elsif rising_edge(clk) then
               q <= q + 1;
         end if;
         if q>=tiempo_medio then
              clksal<='0';
         else
               clksal<='1';
         end if;
      end process;
end Behavioral;
```

Este código se encarga de hacer la división de la señal de reloj en base a la Velocidad que se le indique, para esto se tiene un tiempo medio y un tiempo completo (elegidos por la velocidad que indica el Decodificador Serial) con los cuales mediante comparadores permite la generación de la señal de reloj de salida correctamente.

Código del Conteo de Pasos a Ejecutar

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


entity Contador_Comparador is
    Port ( clk : in  STD_LOGIC;
           Numero : in  STD_LOGIC_VECTOR (4 downto 0);
           Actualizar : in  STD_LOGIC;
           clksal : out  STD_LOGIC;
              clksal2 : out  STD_LOGIC);
end Contador_Comparador;

architecture Behavioral of Contador_Comparador is
--signal Comparador:STD_LOGIC_VECTOR (4 downto 0):="00000";
signal Conteo: STD_LOGIC_VECTOR (5 downto 0):="000000";
signal clkdiv: STD_LOGIC_VECTOR (3 downto 0):="0000";
signal clkdiv2: STD_LOGIC;
signal clksalaux:  STD_LOGIC_VECTOR (0 downto 0);
signal clksal2aux:  STD_LOGIC_VECTOR(0 downto 0);
signal permiso: STD_LOGIC:='0';
```

```
signal Numaux: STD_LOGIC_VECTOR (5 downto 0);
begin
     Numaux<=Numero&"0";
     process(clk,clkdiv)
      begin
         if rising_edge(clk) then
             if clkdiv<9 then
              clkdiv<=clkdiv+1;
                if clkdiv<5 then
                   clkdiv2<='1';
                else
                   clkdiv2<='0';
                end if;
               else
                  clkdiv<="0000";
               end if;
          end if;
      end process;
     process(clkdiv2,Actualizar,Numaux,Conteo,permiso)
      begin
        if (Actualizar='1' and Conteo=Numaux) then
            Conteo<="000000";
             permiso<='0';
        else
                  --if Conteo<Numaux then
                     --permiso<='1';--clksal<=clk
                    --else
                     --permiso<='0';--clksal<=clk
                  --end if;
               if rising_edge(clkdiv2) then
                  if Conteo<Numaux then
                    Conteo<=Conteo+1;
                     permiso<='1';
                  else
                    Conteo<=Numaux;
                     permiso<='0';
                  end if;
               end if;
        end if;
      end process;

      process(clk,permiso)
      begin
           if falling_edge(clk) then
                if permiso='1' then
                    clksalaux<=clksalaux+1;
                else
                    clksalaux<="0";
               end if;
           end if;
      end process;

      process(clkdiv2,permiso)
      begin
           if falling_edge(clkdiv2) then
                if permiso='1' then
                    clksal2aux<=clksal2aux+1;
                else
                    clksal2aux<="0";
               end if;
           end if;
      end process;

      clksal<=clksalaux(0);
      clksal2<=clksal2aux(0);

end Behavioral;
```

Este código en base a lo que se tiene de Pasos a Avanzar y de la señal generada por el Decodificador Serial pasa solo los ciclos de reloj necesarios para hacer avanzar el motor, la entrada de pulso de reloj que tiene viene de acuerdo a la velocidad elegida en el código anterior, y tiene 2 salidas una que genera los pulsos para el motor y otra más lenta que sirva para indicar que cambio de posición el motor.

Código para el avance del Motor.

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

entity Driver_Motor is
    Port ( clk_M : in  STD_LOGIC;--Reloj del Contador del Motor.
                  clk_P : in  STD_LOGIC;--Reloj del contador de Posición.
           selector : in  STD_LOGIC;--Selector de giro del motor
           reset : in  STD_LOGIC;--Reset de los contadores
           Salida : out  STD_LOGIC_VECTOR (3 downto 0);--Salida al motor a pasos.
           Cuenta : out  STD_LOGIC_VECTOR (4 downto 0));--Salida que indica en que posicion se encuentra.
end Driver_Motor;

architecture Behavioral of Driver Motor is
signal Contador1:STD_LOGIC_VECTOR (1 downto 0):="00";--Contador para bobinas del motor a pasos.
signal Contador2:STD_LOGIC_VECTOR (4 downto 0):="00000";--Contador para las 20 posiciones del motor a pasos.
begin
     --Generación de los contadores para el motor a pasos y su posición.
     process(clk_M,selector,reset)
              begin
                  if reset='1' then
                                  Contador1<= (others => '0');
                                  elsif rising_edge(clk_M) then
                                             if selector='1' then--Giro Horario.
                                                Contador1<=Contador1+1;
                                             else--Giro Antihorario.
```

```
                                        Contador1<=Contador1-1;
                                    end if;
                                end if;
            end process;
            process(clk_P,selector,reset)
            begin
                if reset='1' then
                                Contador2<= (others => '0');
                                elsif rising_edge(clk_P) then
                                    if selector='1' then--Giro Horario.
                                       if Contador2>=19 then--Si excede en 20 se resetea.
                                            Contador2<=(others=>'0');--Resetea el registro.
                                        else
                                           Contador2<=Contador2+1;--Incrementa si es menor a 20.
                                        end if;
                                    else--Giro Antihorario.
                                       if Contador2=0 then--Si el contador es igual a 0 resetea en 19.
                                            Contador2<="10011";--Resetea la contador en 19=10011.
                                        else
                                           Contador2<=Contador2-1;--Decrementa si es distinto de 0.
                                        end if;
                                    end if;
                                end if;
            end process;
            --Decodificacion del conteo para generar la señal que va a al motor a pasos.
            process(Contador1)
            begin
              case Contador1 is--Elegir Contador1 como señal para hacer la decodificación.
         when "00" => Salida<="1010";--"0001";--Primera posición del motor a pasos.
         when "01" => Salida<="1001";--"0010";--Segunda posición del motor a pasos.
         when "10" => Salida<="0101";--"0100";--Tercera posición del motor a pasos.
                      when others  => Salida<="0110";--"1000";--Cuarta posición del motor a pasos.
        end case;
            end process;
            process(Contador2) begin
                Cuenta<=Contador2;
            end process;
end Behavioral;
```

Este código con la señal de reloj de entrada (clk_M) y el giro va generando la combinación correcta para hacer avanzar el motor, actuando así como un contador ascendente-descendente decodificado. Además también procesa el pulso de reloj que indica en qué posición va el motor con un lógica simular a la anterior solo que esta no se decodifica teniendo un límite en 19 para luego resetearse.

Código para decodificación en dígitos ASCII

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Decodificador_Tx_Serial is
    Port ( clk : in  STD_LOGIC;
                  clr: in  STD_LOGIC;
           Enviar : in  STD_LOGIC;
           Entrada : in  STD_LOGIC_VECTOR (4 downto 0);
           tdrel : in  STD_LOGIC;
           Registro_Tx : inout  STD_LOGIC_VECTOR (7 downto 0):=X"00";
           Inicio : out  STD_LOGIC);
end Decodificador_Tx_Serial;

architecture Behavioral of Decodificador_Tx_Serial is
type state_type is (mark,espera,esperal,poner_datol,cargar_datol,espera2,poner_dato2,cargar_dato2); --Máquina de estados
signal state: state_type;
signal clkdiv: std_logic_vector(0 downto 0);--Divisor de reloj.
signal contador: std_logic_vector(1 downto 0);--Señal para alargar ciclos en el estado.
signal Regl: std_logic_vector(3 downto 0):=x"0";
signal Reg2: std_logic_vector(0 downto 0):="0";
signal Datol: std_logic_vector(3 downto 0);
signal Dato2: std_logic_vector(0 downto 0);
begin
-- Señal de reloj
    process(clk)
    begin
                if rising_edge(clk) then--Contador de 25000000 pulsos de reloj.
                                clkdiv <= clkdiv +1;--Incrementa el contador en 1.
                        end if;
            end process;

            with Entrada select--Elegir ent como variable para hacer la selección.
        Datol <= X"0" when "00000",--
                 X"1" when "00001",--
                 X"2" when "00010",--
                 X"3" when "00011",--
                 X"4" when "00100",--
                 X"5" when "00101",--
                 X"6" when "00110",--
                 X"7" when "00111",--
                 X"8" when "01000",--
                 X"9" when "01001",--
                 X"0" when "01010",--
                 X"1" when "01011",--
```

```
        X"2" when "01100",--
        X"3" when "01101",--
        X"4" when "01110",--
        X"5" when "01111",--
        X"6" when "10000",--
        X"7" when "10001",--
        X"8" when "10010",--
        X"9" when "10011",--
        X"0" when others;--

    process(Entrada)
    begin
                                    if Entrada>=10 then
                                       Dato2<="1";
                                    else
                                       Dato2<="0";
                                    end if;
    end process;
    --Máquina de estados
    process(clkdiv(0),clr,Dato1,Dato2,Entrada)
    begin
        if clr='1' then--Si el reset esta inactivo carga configuración inicial.
                    state <=mark;--Estado siguiente es mark.
                            Inicio<='0';
                elsif rising_edge(clkdiv(0)) then--Elige clk como reloj de la máquina de estados.
                            case state is
                              when mark=>--Cuando se encuentra en mark(estado de inicio).
                                        Inicio<='0';
                                                    if Enviar='1' and Entrada<20 then
                                                       state<=espera;
                                                       Reg1<=Dato1;
                                                      Reg2<=Dato2;
                                                    else
                                                       state<=mark;
                                                       Reg1<="0000";
                                                       Reg2<="0";
                                                    end if;

                                    when espera=>
                                       if Enviar='0' then
                                                  state<=espera1;
                                            else
                                               state<=espera;
                                            end if;

                                 when espera1=>
                                             Inicio<='0';
                                                    if tdre1='1' then
                                                        state<=poner_dato1;
                                                    else
                                                        state<=espera1;
                                                    end if;

                                    when poner_dato1=>
                                                --Registro_Tx<=(X"3")&Reg1;
                                                          Registro_Tx<=("0011000")&Reg2;
                                            if contador>2 then
                                                contador<="00";
                                                state<=cargar_dato1;
                                            else
                                                contador<=contador+1;
                                                state<=poner_dato1;
                                            end if;

                                    when cargar_dato1=>
                                            --Registro_Tx<=(X"3")&Reg1;
                                            Registro_Tx<=("0011000")&Reg2;
                                            Inicio<='1';
                                            if contador>2 then
                                                contador<="00";
                                                state<=espera2;
                                            else
                                                contador<=contador+1;
                                                state<=cargar_dato1;
                                            end if;

                                    when espera2=>
                                                    Inicio<='0';
                                                    if tdre1='1' then
                                                        state<=poner_dato2;
                                                    else
                                                        state<=espera2;
                                                    end if;

                                    when poner_dato2=>
                                                --Registro_Tx<=("0011000")&Reg2;
                                            Registro_Tx<=(X"3")&Reg1;
                                            if contador>2 then
                                                contador<="00";
                                                state<=cargar_dato2;
                                            else
                                                contador<=contador+1;
                                                state<=poner_dato2;
                                            end if;

                                    when cargar_dato2=>
                                                --Registro_Tx<=("0011000")&Reg2;
                                            Registro_Tx<=(X"3")&Reg1;
                                            Inicio<='1';
                                            if contador>2 then
                                                contador<="00";
                                                state<=mark;
                                            else
                                                contador<=contador+1;
                                                state<=cargar_dato2;
                                            end if;
                                 end case;
                               end if;
    end process;

end Behavioral;
```

Este código se encarga de hacer la decodificación de la posición en que va el motor por medio de un decodificador y una máquina de estados, el decodificador solo separa en dígitos numéricos para posteriormente ser enviados, la máquina de estados detecta cada pulso que envía el contador de posición y base a este pulso envía el primer dígito en ASCII poniendo el dato en el Transmisor serial, una vez que el Transmisor serial le indica que envió el dato, ahora envía el segundo dígito en ASCII por medio del Transmisor y regresa a su estado inicial para saber si se quiere enviar otro dato.

Código de Transmisor Serial

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Transmisor_Serial is
    Port ( clk : in  STD_LOGIC;--Señal de reloj de entrada.
           clr : in  STD_LOGIC;--Clear del transmisor serial.
           tx_data : in  STD_LOGIC_VECTOR (7 downto 0);--Registro de donde se toma el byte.
                    outp: in std_logic;--Señal que indica cuando puede empezar a transmitir (Activa la transmisiòn en alto).
           tdre : out  STD_LOGIC;--Bit que indica que se envió el dato completo o que el dispositivo puede transmitir(1).
           txd : out  STD_LOGIC);--Salida del transmisor serial.
end Transmisor_Serial;

architecture Behavioral of Transmisor_Serial is
type state_type is (mark,start,delay,shift,stop); --Máquina de estados
signal state: state_type;
signal txbuff: std_logic_vector(7 downto 0);--Buffer donde se guarda el dato a transmitir y que se recorre serialmente.
signal baud_count: std_logic_vector(11 downto 0);--Contador de tiempo para lograr los bauds requeridos.
signal bit_count: std_logic_vector(3 downto 0);--Contador de bits que se han transmitido por byte.
constant bit_time: std_logic_vector(11 downto 0) := X"a28";--Constante de tiempo equivalente a 9600 bauds.
signal clkdiv: std_logic_vector(0 downto 0);--Divisor de reloj.
--signal outp: std_logic;--Señal que indica cuando empezar a transmitir.
begin
 -- Señal de reloj
    process(clk)
    begin
              if rising_edge(clk) then
                            clkdiv <= clkdiv +1;--Incrementa el contador en 1.
                        end if;
           end process;

           --Máquina de estados
           process(clkdiv(0),clr,outp)
           begin
                if clr='1' then--Si el reset esta inactivo carga configuración inicial.
                           state <=mark;--Estado siguiente es mark.
                                   txbuff<="00000000";--Limpia el buffer de datos.
                                   baud_count<=X"000";--Contador de clk para los bauds en 0.
                                   bit count<="0000";--Contador de bits en 0.
                                   txd<='1';--Transmisor serial es 1 que indica estado de reposo.
                      elsif rising_edge(clkdiv(0)) then--Elige clk como reloj de la máquina de estados.
                                   case state is
                                     when mark=>--Cuando se encuentra en mark(estado de inicio y espera).
                                               bit_count<="0000";--Pone la cuenta de bits recorridos en 0.
                                               tdre<='1';--El serial se encuentra listo o ya fue cargado en otro dispositivo.
                                              if outp='0' then--Si no se quiere empezar a mandar un dato.
                                               state<=mark;--Asigna como estado siguiente a mark.
                                               txbuff<=tx_data;--Carga el buffer con el registro de datos.
                                              else--En caso que se quiera iniciar a mandar un dato se ejecuta.
                                                       baud count<=X"000";--Inicia la cuenta del tiempo de bauds requeridos en 0.
                                                       state<=start;--Estado siguiente start.
                                              end if;

                                            when start=>--Cuando se encuentra en start(Estado de comienzo de comunicación serial).
                                               baud_count<=X"000";--Inicia la cuenta del tiempo de bauds requeridos en 0.
                                                       txd<='0';--Manda el bit de inicio.
                                               tdre<='0';--Pone el bit de dato listo en 0.
                                                       state<= delay;--Pasa al siguiente estado "delay".

                                     when delay=>--Estado de espera de tiempo de bauds
                           tdre<='0';--Pone el bit de dato listo en 0.
                           if baud_count>=bit_time then--Si se supera la cuenta reinicia el contador de bauds.
                                                    baud_count<=X"000";--Reinicia el contador de bauds.
                                                            if bit_count<"1000" then--Si el conteo de bits es menor a 8.
                                                               state<=shift;--Asigna como estado siguiente a shift.
                                                            else-- En caso contrario se el corrimiento.
                                                               state<=stop;--Asigna como estado siguiente a stop.
                                                  end if;
                                             else--Si el contador de bauds es menor al tiempo requerido sigue esperando.
                                                     baud count<=baud count+1;--Aumenta en 1 el contador de bauds.
                                                       state<=delay;--Asigna como siguiente estado a delay.
                                                end if;

                                            when shift=>--Estado de recorrimiento.
                                                 tdre<='0';--Pone el bit de dato listo en 0.
                                                       txd<=txbuff(0);--Carga a la salida el primer bit de txbuff.
                                                       txbuff(6 downto 0)<=txbuff(7 downto 1);--Hace el recorrimiento del dato.
                                                       bit_count<=bit_count+1;--Incrementa en 1 el conteo de bits enviados.
                                                       state<=delay;--Asigna como siguiente estado a delay.
```

```
                                    when stop=>--Estado de parada.
                tdre<='0';--Pone el bit de dato listo en 0.
                txd<='1';--Envia el bit de parada.
                if baud_count>= bit_time then--Si se supera la cuenta reinicia el contador de bauds.
                   baud_count<=X"000";--Reinicia el contador de bauds.
                                                        state<=mark;--Asigna como estado siguiente a mark entrando a modo espera.
                else--en caso contrario sigue contando.
                   baud count<= baud count+1;--Aumenta en 1 el contador de bauds.
                   state<=stop;--Asigna como estado siguiente a stop para seguir enviando el bit de parada.
                end if;
                                end case;
                               end if;
        end process;
end Behavioral;
```

Este último código se encarga de poder transmitir un dato de 8 bits serialmente a 9600 bauds, generando además una señal de salida que nos indica que ya fue transmitido el dato.

Imágenes del Proyecto