Encoder mecánico rotatorio.

Se presenta un programa en VHDL y uno en verilog para el control de giro de un encoder mecánico rotatorio y visualización de datos en display, utilizando el módulo periférico PmodENC de digilent, mostrado en la figura 1, que es modificable para el control de motores en sus distintas aplicaciones (grippers, brazos, persianas, ventanas, puertas, compuertas, etc.) así como el manejo de displays, volumen, etc.

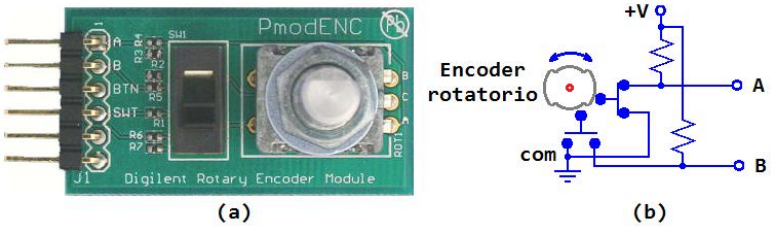

Figura 1. Encoder mecánico rotatorio (a) módulo periférico para las tarjetas de desarrollo, (b) circuito equivalente.

En la figura 2 se presenta el proyecto implementado en el ISE, con un diseño TOP LEVEL (Top Level Design o TLD) realizado en VHDL.

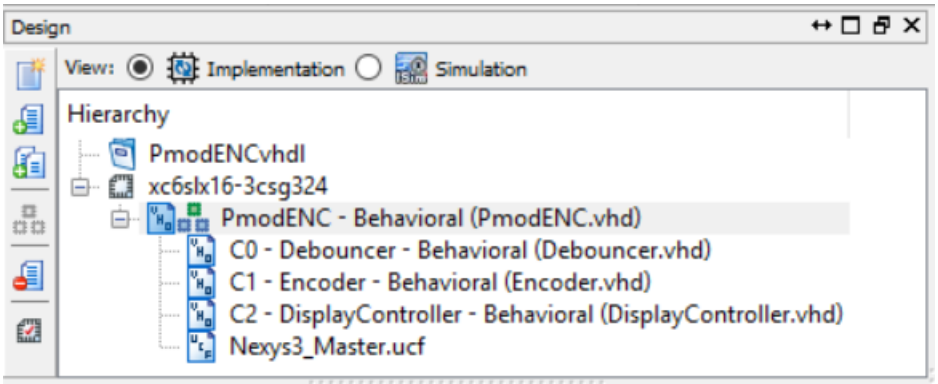

Figura 2. Proyecto en vhdl del controlador de un encoder.

A continuación, se listan los 5 programas utilizados:

```
----------------------------------------------------------------------------------------------------------------------------
-- Project Name:  PmodENC (PmodENC.vhd)
-- Target Devices: Nexys3
-- This project changes the seven segments display when the position of rotary shaft is changed.
-- The number on the 7 segments display is relative to the start position. When the rotary button
-- is pressed, the program resets. The switch controls whether the 7segments display turns on
-- or off. LED 0 and 1 indicated the direction the rotary shaft is turned. LED 0 is on when the shaft
-- is turned right, LED 1 is on when the shaft is turned left.
----------------------------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PmodENC is
```

```
   Port (
          clk: in std_logic;
          JA : in STD_LOGIC_VECTOR (7 downto 4); -- to the lower row of connector JA
          an : out  STD_LOGIC_VECTOR (3 downto 0); -- controls the display digits
          seg: out  STD_LOGIC_VECTOR (6 downto 0); -- controls what digit to display
          Led: out STD_LOGIC_VECTOR (1 downto 0) -- Led indicates the direction the shaft
          );
end PmodENC;

architecture Behavioral of PmodENC is

-- signals
signal EncO : std_logic_vector (4 downto 0);
signal AO, BO: std_logic;

begin
  C0: entity work.Debouncer
          port map ( clk=>clk, Ain=>JA(4), Bin=>JA(5), Aout=> AO, Bout=> BO);
  C1: entity work.Encoder
          port map ( clk=>clk, A=>AO, B=>BO, BTN=>JA(6), EncOut=>EncO, LED=>Led);
 C2: entity work.DisplayController
          port map (clk=>clk, SWT=>JA(7), DispVal=>EncO, anode=>an, segOut=>seg );

end Behavioral;



----------------------------------------------------------------------------------------------------------------------------
-- Module Name:    debouncer - Behavioral (Debouncer.vhd), component C0
-- Project Name:           PmodENC
-- Target Devices: Nexys 3
-- This file defines a debouncer for eliminating the logic noise presented when the shaft is rotated.
----------------------------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Debouncer is
   Port ( clk : in  STD_LOGIC;
                          -- signals from the pmod
        Ain : in  STD_LOGIC;
        Bin : in  STD_LOGIC;
                          -- debounced signals
                           Aout: out STD_LOGIC;
                           Bout: out STD_LOGIC
                           );
end Debouncer;
```

```
architecture Behavioral of Debouncer is
-- signals
signal sclk: std_logic_vector (6 downto 0);
signal sampledA, sampledB : std_logic;

begin

	process(clk)
	  begin
		if clk'event and clk = '1' then
			sampledA <= Ain;
			sampledB <= Bin;
			-- clock is divided to 1MHz
			-- samples every 1uS to check if the input is the same as the sample
			-- if the signal is stable, the debouncer should output the signal
			if sclk = "1100100" then
			-- A
			   if sampledA = Ain then
				Aout <= Ain;
			   end if;
			-- B
			   if sampledB = Bin then
				Bout <= Bin;
			   end if;

			   sclk <="0000000";
			else
				sclk <= sclk +1;
			end if;
		end if;
	end process;

end Behavioral;


----------------------------------------------------------------------------------------------------------------------------
-- Module Name:    Encoder - Behavioral (Encoder.vhd), component C1
-- Project Name:           PmodENC
-- Target Devices: Nexys 3
-- This module defines a component Encoder with a state machine that reads
-- the position of the shaft relative to the starting position.
----------------------------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity Encoder is
   Port (
          clk: in STD_LOGIC;
          -- signals from the pmod
          A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          BTN : in  STD_LOGIC;
          -- position of the shaft
          EncOut: inout STD_LOGIC_VECTOR (4 downto 0);
          -- direction indicator
          LED: out STD_LOGIC_VECTOR (1 downto 0)
           );
end Encoder;

architecture Behavioral of Encoder is
-- FSM states and signals
type stateType is ( idle, R1, R2, R3, L1, L2, L3, add, sub);
signal curState, nextState: stateType;

begin

--clk and button
  clock: process (clk, BTN)
  begin
          -- if the rotary button is pressed the count resets
      if (BTN='1') then
          curState <= idle;
          EncOut <= "00000";
      elsif (clk'event and clk = '1') then
          -- detect if the shaft is rotated to right or left
          -- right: add 1 to the position at each click
          -- left: subtract 1 from the position at each click
          if curState /= nextState then
                  if (curState = add) then
                          if EncOut < "10011" then
                                  EncOut <= EncOut+1;
                          else
                                  EncOut <= "00000";
                          end if;
                  elsif (curState = sub) then
                          if EncOut > "00000" then
                                  EncOut <= EncOut-1;
                          else
                                  EncOut <= "10011";
                          end if;
                  else
                          EncOut <= EncOut;
                  end if;
```

```
            else
                    EncOut <= EncOut;
            end if;
           curState <= nextState;
        end if;
    end process;

-----FSM process
    next_state: process (curState, A, B)
    begin
            case curState is
            --detent position
            when idle =>
                    LED<= "00";
                    if B = '0' then
                            nextState <= R1;
                    elsif A = '0' then
                            nextState <= L1;
                    else
                            nextState <= idle;
                    end if;
            -- start of right cycle
            --R1
            when R1 =>
                    LED<= "01";
                    if B='1' then
                            nextState <= idle;
                    elsif A = '0' then
                            nextState <= R2;
                    else
                            nextState <= R1;
                    end if;
            --R2
            when R2 =>
                    LED<= "01";
                    if A ='1' then
                            nextState <= R1;
                    elsif B = '1' then
                            nextState <= R3;
                    else
                            nextState <= R2;
                    end if;
            --R3
            when R3 =>
                    LED<= "01";
                    if B ='0' then
                            nextState <= R2;
                    elsif A = '1' then
```

```
                    nextState <= add;
                else
                    nextState <= R3;
                end if;
            when add =>
                LED<= "01";
                nextState <= idle;
            -- start of left cycle
            --L1
            when L1 =>
                LED<= "10";
                if A ='1' then
                    nextState <= idle;
                elsif B = '0' then
                    nextState <= L2;
                else
                    nextState <= L1;
                end if;
            --L2
            when L2 =>
                LED<= "10";
                if B ='1' then
                    nextState <= L1;
                elsif A = '1' then
                    nextState <= L3;
                else
                    nextState <= L2;
                end if;
            --L3
            when L3 =>
                LED<= "10";
                if A ='0' then
                    nextState <= L2;
                elsif B = '1' then
                    nextState <= sub;
                else
                    nextState <= L3;
                end if;
            when sub =>
                LED<= "10";
                nextState <= idle;
            when others =>
                LED<= "11";
                nextState <= idle;
            end case;
            end process;

end Behavioral;
```

```
----------------------------------------------------------------------------------------------------------------------------
-- Module Name:    DisplayController - Behavioral (DisplayController.vhd), component C2
-- Project Name:           PmodENC
-- Target Devices: Nexys 3
-- This module defines a DisplayController that controls the seven segments display to work with
-- the output of the Encoder
----------------------------------------------------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DisplayController is
   Port (
            clk : in std_logic;
            --signal from the Pmod
            SWT: in std_logic;
            --output from the Encoder
            DispVal : in  STD_LOGIC_VECTOR (4 downto 0);
            --controls the display digits
            anode: out std_logic_vector(3 downto 0);
            --controls which digit to display
            segOut : out  STD_LOGIC_VECTOR (6 downto 0));
   end DisplayController;

architecture Behavioral of DisplayController is
          -- signals
signal sclk: std_logic_vector (17 downto 0);
signal seg: std_logic_vector (6 downto 0);

begin
   process(clk, SWT)
      begin
         -- turns off the seven segments display when the switch is off
         --or else turn on the seven segments display
         if (SWT = '0') then
                  anode<="1111";
         --refresh each digit
         elsif clk'event and clk = '1' then
                  -- 0ms refersh digit 0
                  if sclk = "000000000000000000" then
                           anode<="1110";
                           segOut <= seg;
                           sclk <= sclk +1;
                  -- 1ms refresh digit 1
                  elsif sclk = "011000011010100000" then
                  -- display a 1 on the tenth digit if the number is greater than 9
```

```
                        if DispVal > "01001" then
                                segOut <= "1111001";
                                anode<="1101";
                        end if;
                        sclk <= sclk +1;
                --      2ms
                elsif sclk = "110000110101000000" then
                        sclk <= "000000000000000000";
                else
                        sclk <= sclk +1;
                end if;
        end if;
        end process;

         with DispVal select
--                  gfedcba
        seg <=  "1000000" when "00000", --0
                "1111001" when "00001", --1
                "0100100" when "00010", --2
                "0110000" when "00011", --3
                "0011001" when "00100", --4
                "0010010" when "00101", --5
                "0000010" when "00110", --6
                "1111000" when "00111", --7
                "0000000" when "01000", --8
                "0010000" when "01001", --9
                "1000000" when "01010", --10
                "1111001" when "01011", --11
                "0100100" when "01100", --12
                "0110000" when "01101", --13
                "0011001" when "01110", --14
                "0010010" when "01111", --15
                "0000010" when "10000", --16
                "1111000" when "10001", --17
                "0000000" when "10010", --18
                "0010000" when "10011", --19
                "0111111" when others;

end Behavioral;

##############################################################################
## Users Constrains File (Nexys3_Master.ucf) for Nexys3 rev B board          #
##############################################################################

##Clock signal
Net "clk" LOC=V10 | IOSTANDARD=LVCMOS33;                # 100MHz
## 7 segment display
Net "seg<0>" LOC = T17 | IOSTANDARD = LVCMOS33;         # CA
```

```
Net "seg<1>" LOC = T18 | IOSTANDARD = LVCMOS33;          # CB
Net "seg<2>" LOC = U17 | IOSTANDARD = LVCMOS33;          # CC
Net "seg<3>" LOC = U18 | IOSTANDARD = LVCMOS33;          # CD
Net "seg<4>" LOC = M14 | IOSTANDARD = LVCMOS33;          # CE
Net "seg<5>" LOC = N14 | IOSTANDARD = LVCMOS33;          # CF
Net "seg<6>" LOC = L14 | IOSTANDARD = LVCMOS33;          # CG
#Net "seg<7>" LOC = M13 | IOSTANDARD = LVCMOS33;         # DP

Net "an<0>" LOC = N16 | IOSTANDARD = LVCMOS33;   # AN0
Net "an<1>" LOC = N15 | IOSTANDARD = LVCMOS33;   # AN1
Net "an<2>" LOC = P18 | IOSTANDARD = LVCMOS33;   # AN2
Net "an<3>" LOC = P17 | IOSTANDARD = LVCMOS33;   # AN3

## Leds
Net "Led<0>" LOC = U16 | IOSTANDARD = LVCMOS33;          # LD0
Net "Led<1>" LOC = V16 | IOSTANDARD = LVCMOS33;          # LD1
#Net "Led<2>" LOC = U15 | IOSTANDARD = LVCMOS33;         # LD2
#Net "Led<3>" LOC = V15 | IOSTANDARD = LVCMOS33;         # LD3
#Net "Led<4>" LOC = M11 | IOSTANDARD = LVCMOS33;         # LD4
#Net "Led<5>" LOC = N11 | IOSTANDARD = LVCMOS33;         # LD5
#Net "Led<6>" LOC = R11 | IOSTANDARD = LVCMOS33;         # LD6
#Net "Led<7>" LOC = T11 | IOSTANDARD = LVCMOS33;         # LD7

##JA
#Net "JA<0>" LOC = T12 | IOSTANDARD = LVCMOS33;   # JA1
#Net "JA<1>" LOC = V12 | IOSTANDARD = LVCMOS33;   # JA2
#Net "JA<2>" LOC = N10 | IOSTANDARD = LVCMOS33;   # JA3
#Net "JA<3>" LOC = P11 | IOSTANDARD = LVCMOS33;   # JA4
Net "JA<4>" LOC = M10 | IOSTANDARD = LVCMOS33;   # JA7
Net "JA<5>" LOC = N9 | IOSTANDARD = LVCMOS33;    # JA8
Net "JA<6>" LOC = U11 | IOSTANDARD = LVCMOS33;   # JA9
Net "JA<7>" LOC = V11 | IOSTANDARD = LVCMOS33;   # JA10
```

En la figura 3 se presenta el proyecto implementado en el ISE, con un diseño TOP LEVEL (Top Level Design o TLD) para el encoder escrito en verilog.

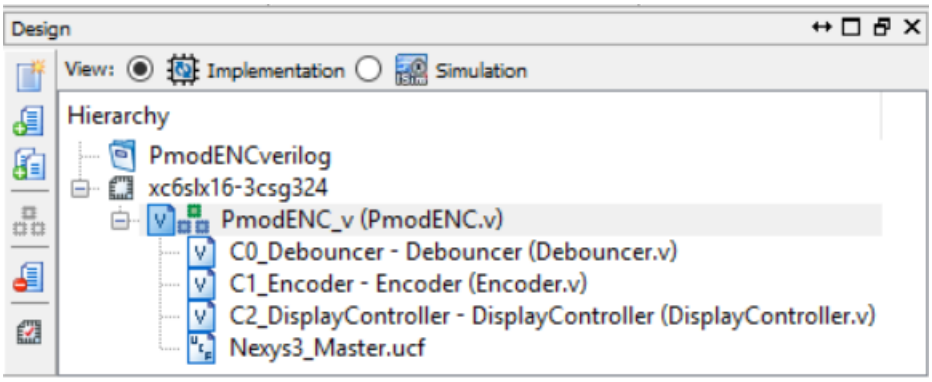

Figura 3. Proyecto en verilog del controlador de un encoder.

A continuación, se listan los 4 programas utilizados (el Nexys3_Master.ucf es el mismo que se presentó anteriormente):

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Module Name:    PmodENC
// Project Name:  PmodENC (PmodENC.vhd)
// Target Devices: Nexys3
// This project changes the 7 segments display when the position of rotary shaft is changed.
// The number on the 7 segments display is relative to the start position. When the rotary button
// is pressed, the program resets. The switch controls whether the 7 segments display turns on
// or off. LED 0 and 1 indicated the direction the rotary shaft is turned. LED 0 is on when the shaft
// is turned right, LED 1 is on when the shaft is turned left.
//////////////////////////////////////////////////////////////////////////////////
module PmodENC_v(
   clk,
   JA,
   an,
   seg,
   Led
   );

// ============================================================================
//                                    Port Declarations
// ============================================================================
   input clk;
   input [7:4] JA;
   output [3:0] an;
   output [6:0] seg;
   output [1:0] Led;
```

```
// ======================================================================
//                          Parameters, Regsiters, and Wires
// ======================================================================
        wire [3:0] an;
        wire [6:0] seg;
        wire [1:0] Led;

        wire [4:0] EncO;

// ======================================================================
//                            Implementation
// ======================================================================
        Debouncer C0_Debouncer (
                                        .clk(clk),
                                        .Ain(JA[4]),
                                        .Bin(JA[5]),
                                        .Aout(AO),
                                        .Bout(BO)
        );

        Encoder C1_Encoder (
                                        .clk(clk),
                                        .A(AO),
                                        .B(BO),
                                        .BTN(JA[6]),
                                        .EncOut(EncO),
                                        .LED(Led)
        );

        DisplayController C2_DisplayController (
                                        .clk(clk),
                                        .SWT(JA[7]),
                                        .DispVal(EncO),
                                        .anode(an),
                                        .segOut(seg)
        );
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Module Name:    Debouncer (Debouncer.v)
// Project Name:          PmodENC
// Target Devices: Nexys 3
// This file defines a debouncer for eliminating the logic noise presented when the shaft is rotated.
//////////////////////////////////////////////////////////////////////////////////
module Debouncer(
   clk,
   Ain,
   Bin,
   Aout,
   Bout
   );

// ============================================================================
//                                      Port Declarations
 // ============================================================================
   input clk;
   input Ain;
   input Bin;
   output Aout;
   output Bout;

// ============================================================================
//                                      Parameters, Regsiters, and Wires
// ============================================================================
           reg Aout = 0;
           reg Bout = 0;

           reg [6:0] sclk = 0;
           reg sampledA = 0;
           reg sampledB = 0;

// ============================================================================
//                                      Implementation
// ============================================================================
  always @(posedge clk) begin
          sampledA <= Ain;
          sampledB <= Bin;
          // clock is divided to 1MHz
          // samples every 1uS to check if the input is the same as the sample
          // if the signal is stable, the debouncer should output the signal

          if(sclk == 7'b1100100) begin
                  if(sampledA == Ain) begin
                          Aout <= Ain;
                  end
```

```
                if(sampledB == Bin) begin
                        Bout <= Bin;
                end

                sclk <= 7'b0000000;
        end

        else begin
                sclk <= sclk + 7'b0000001;
        end
  end

endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Module Name:    Encoder (Encoder.v)
// Project Name:          PmodENC
// Target Devices: Nexys 3
// This module defines a component Encoder with a state machine that reads
// the position of the shaft relative to the starting position.
//////////////////////////////////////////////////////////////////////////////////
module Encoder(
   clk,
   A,
   B,
   BTN,
   EncOut,
   LED
   );

// ============================================================================
//                                          Port Declarations
// ============================================================================
   input clk;
   input A;
   input B;
   input BTN;
   output [4:0] EncOut;
   output [1:0] LED;


// ============================================================================
//                                      Parameters, Regsiters, and Wires
// ============================================================================
          reg [4:0] EncOut;
```

```
        reg [1:0] LED;
        reg [31:0] curState = "idle";
        reg [31:0] nextState;

// ============================================================================
//                                  Implementation
// ============================================================================

// ****************************************
//                      Clock
// ****************************************
  always@(posedge clk or posedge BTN)
  begin
        if (BTN == 1'b1) begin
                curState <= "idle";
                EncOut <= 5'b00000;
        end
        // detect if the shaft is rotated to right or left
        // right: add 1 to the position at each click
        // left: subtract 1 from the position at each click
        else begin
                if(curState != nextState) begin
                        if(curState == "add") begin
                                if(EncOut < 5'b10011) begin
                                        EncOut <= EncOut + 1'b1;
                                end
                                else begin
                                        EncOut <= 5'b00000;
                                end
                        end
                        else if(curState == "sub") begin
                                if(EncOut > 5'b00000) begin
                                        EncOut <= EncOut - 1'b1;
                                end
                                else begin
                                        EncOut <= 5'b10011;
                                end
                        end
                        else begin
                                EncOut <= EncOut;
                        end
                end
                else begin
                        EncOut <= EncOut;
                end

        curState <= nextState;
```

```
        end
    end


// ****************************************
//                Next State
// ****************************************
    always@(curState or A or B)
    begin
        case (curState)
         //detent position
         "idle" : begin
                LED <= 2'b00;

                if (B == 1'b0) begin
                        nextState <= "R1";
                end
                else if (A == 1'b0) begin
                        nextState <= "L1";
                end
                else begin
                        nextState <= "idle";
                end
         end
         // start of right cycle
         // R1
         "R1" : begin
                LED <= 2'b01;

                if (B == 1'b1) begin
                        nextState <= "idle";
                end
                else if (A == 1'b0) begin
                        nextState <= "R2";
                end
                else begin
                        nextState <= "R1";
                end
         end
         // R2
         "R2" : begin
                LED <= 2'b01;

                if (A == 1'b1) begin
                        nextState <= "R1";
                end
                else if (B == 1'b1) begin
                        nextState <= "R3";
```

```
        end
        else begin
                nextState <= "R2";
        end
end
// R3
"R3" : begin
        LED <= 2'b01;

        if (B == 1'b0) begin
                nextState <= "R2";
        end
        else if (A == 1'b1) begin
                nextState <= "add";
        end
        else begin
                nextState <= "R3";
        end
end
// Add
"add" : begin
        LED <= 2'b01;
        nextState <= "idle";
end
// Start of left cycle
// L1
"L1" : begin
        LED <= 2'b10;

        if (A == 1'b1) begin
                nextState <= "idle";
        end
        else if (B == 1'b0) begin
                nextState <= "L2";
        end
        else begin
                nextState <= "L1";
        end
end
// L2
"L2" : begin
        LED <= 2'b10;

        if (B == 1'b1) begin
                nextState <= "L1";
        end
        else if (A == 1'b1) begin
                nextState <= "L3";
```

```
                end
                else begin
                        nextState <= "L2";
                end
        end
        // L3
        "L3" : begin
                LED <= 2'b10;

                if (A == 1'b0) begin
                        nextState <= "L2";
                end
                else if (B == 1'b1) begin
                        nextState <= "sub";
                end
                else begin
                        nextState <= "L3";
                end
        end
        // Sub
        "sub" : begin
                LED <= 2'b10;
                nextState <= "idle";
        end
        //  Default
        default : begin
                LED <= 2'b11;
                nextState <= "idle";
        end
       endcase

  end

endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Module Name:    DisplayController (DisplayController.v)
// Project Name:          PmodENC
// Target Devices: Nexys 3
// This module defines a DisplayController that controls the 7 segments display to
// work with the output of the Encoder
//////////////////////////////////////////////////////////////////////////////////
module DisplayController(
   clk,
   SWT,
   DispVal,
```

```
    anode,
    segOut
    );

// ======================================================================
//                          Port Declarations
// ======================================================================
    input clk;
    input SWT;
    input [4:0] DispVal;
    output [3:0] anode;
    output [6:0] segOut;

// ======================================================================
//                   Parameters, Regsiters, and Wires
// ======================================================================
        reg [3:0] anode;
        reg [6:0] segOut;

        reg [17:0] sclk;
        reg [6:0] seg;

// ======================================================================
//                          Implementation
// ======================================================================
        always @(posedge clk)
        begin
        // turns off the 7 segments display when the switch is off
        // or else turn on the 7 segments display
            if(SWT == 1'b0)  begin
                anode <= 4'b1111;
            end

            // refresh each digit
            else begin
            // 0 ms refersh digit 0
                if(sclk == 18'b000000000000000000) begin
                    anode <= 4'b1110;
                    segOut <= seg;
                    sclk <= sclk + 1'b1;
                end

            // 1ms refresh digit 1
                else if(sclk == 18'b011000011010100000) begin
            // display a 1 on the tenth digit if the number is greater than 9
                    if(DispVal > 5'b01001) begin
                        segOut <= 7'b1111001;
                        anode <= 4'b1101;
```

```
                    end

                sclk <= sclk + 1'b1;
                end

        //      2ms
                else if(sclk == 18'b110000110101000000) begin
                    sclk <= 18'b000000000000000000;
                end

        // Increment
                else begin
                    sclk <= sclk + 1'b1;
                end
        end
end

//  Selects the value to display on the SSD
always @(DispVal[4] or DispVal[3] or DispVal[2] or DispVal[1] or DispVal[0])
case (DispVal[4:0])
                    //     gfedcba
        5'b00000 : seg <= 7'b1000000;       // 0
        5'b00001 : seg <= 7'b1111001;       // 1
        5'b00010 : seg <= 7'b0100100;       // 2
        5'b00011 : seg <= 7'b0110000;       // 3
        5'b00100 : seg <= 7'b0011001;       // 4
        5'b00101 : seg <= 7'b0010010;       // 5
        5'b00110 : seg <= 7'b0000010;       // 6
        5'b00111 : seg <= 7'b1111000;       // 7
        5'b01000 : seg <= 7'b0000000;       // 8
        5'b01001 : seg <= 7'b0010000;       // 9
        5'b01010 : seg <= 7'b1000000;       // 10
        5'b01011 : seg <= 7'b1111001;       // 11
        5'b01100 : seg <= 7'b0100100;       // 12
        5'b01101 : seg <= 7'b0110000;       // 13
        5'b01110 : seg <= 7'b0011001;       // 14
        5'b01111 : seg <= 7'b0010010;       // 15
        5'b10000 : seg <= 7'b0000010;       // 16
        5'b10001 : seg <= 7'b1111000;       // 17
        5'b10010 : seg <= 7'b0000000;       // 18
        5'b10011 : seg <= 7'b0010000;       // 19
        default :    seg <= 7'b0111111;

endcase

endmodule
```

Referencias.

Módulo periférico PmodENC, consultada en febrero-2018,
https://reference.digilentinc.com/pmod/pmod/enc/example_code