# Encoder

Final Project

Designers: Cynthia Babecka & McCleland Idaewor

Class: CpE 3020

Term: Fall 2025

Date: 2025-12-2

## Design Description

Using the PmodENC rotary encoder, the design creates a unique LED display pattern on the Basys3 FPGA board based on the position of the encoder. The system will light up a single LED based on the position determined by if the encoder is rotated to the left or right. The button on the encoder will also change the mode of the leds where speed mode only shows one led and the degree mode shows all leds except for one and the seven segments will display the speed and degree of the rotary encoder respectfully.

The first component takes in the raw values of the encoder and based on its position, displays a represented led on the board that moves left or right. The second component acts as a debounce for the incoming PmodENC rotary encoder which sends the debounced signal to the Encoder component The second component is a modified seven segment that displays a negative sign.

## Components

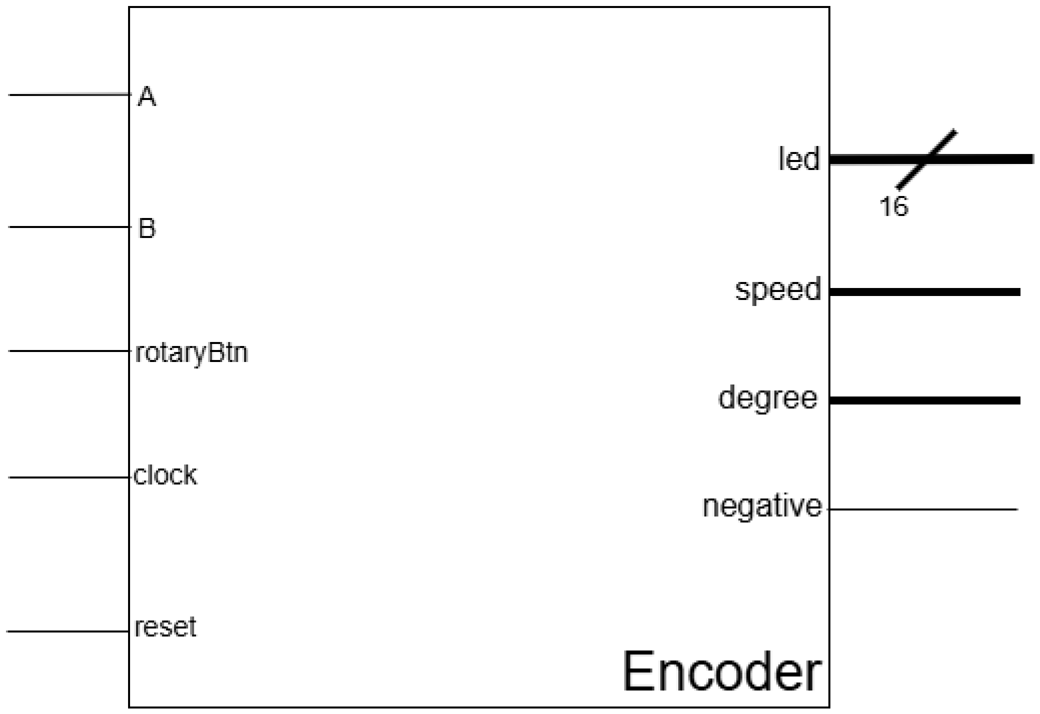


*Figure 1: Encoder Component Diagram*

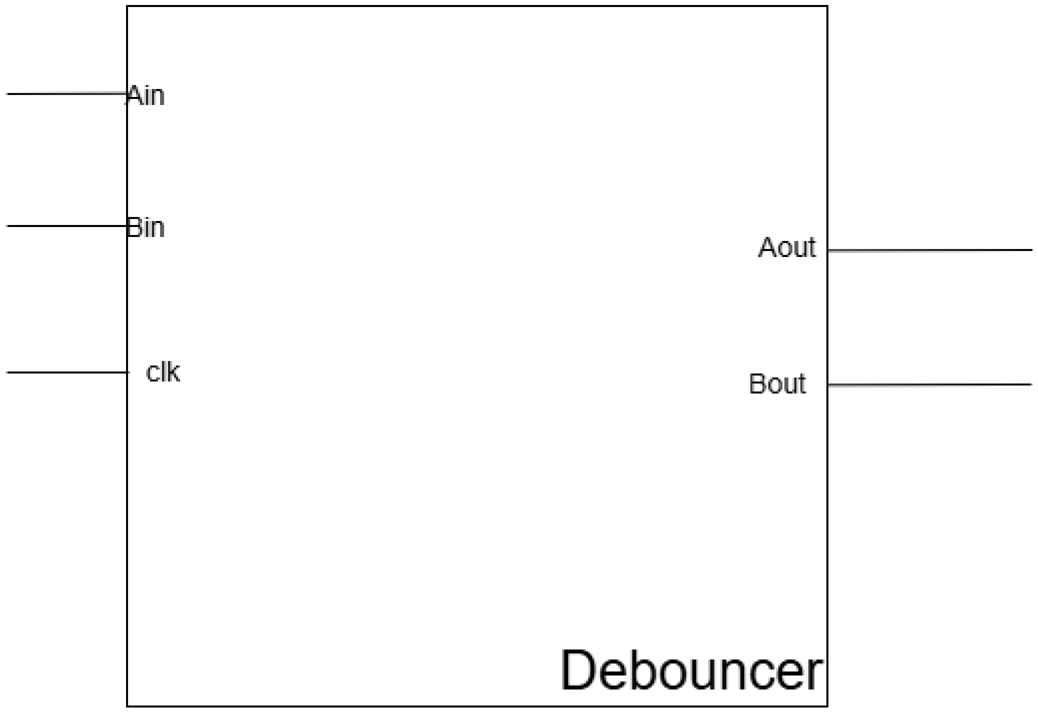


*Figure 2: Debouncer Component Diagram*

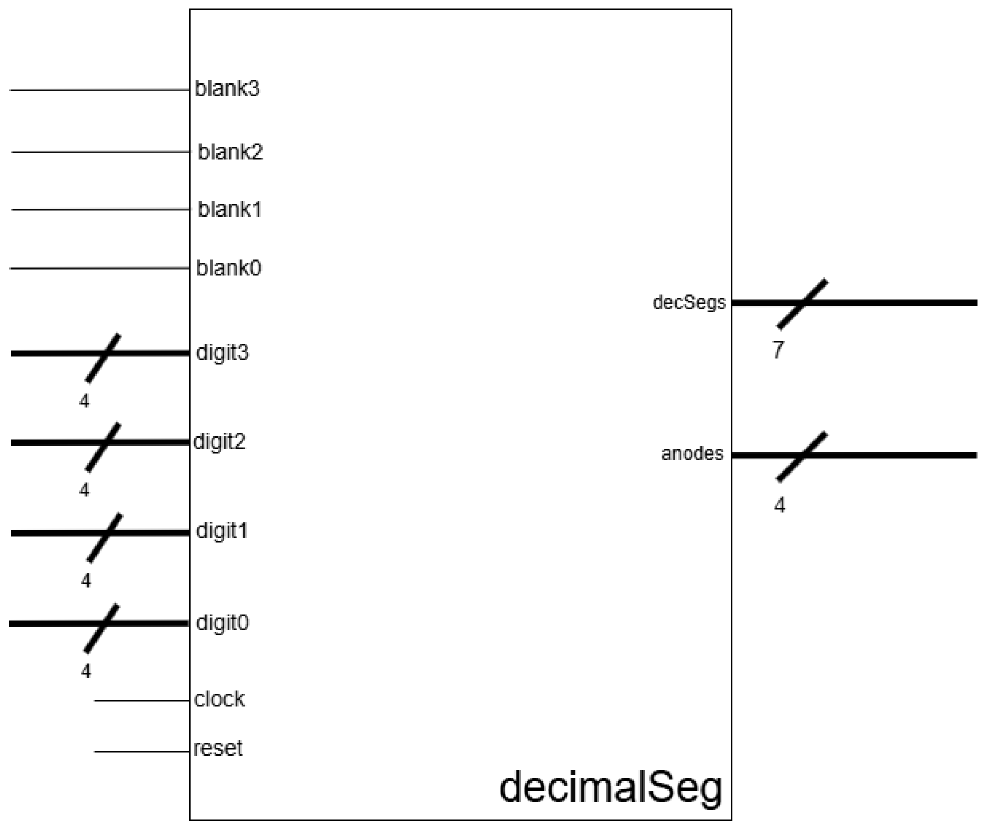


*Figure 3: decimalSeg Component Diagram*

## Design

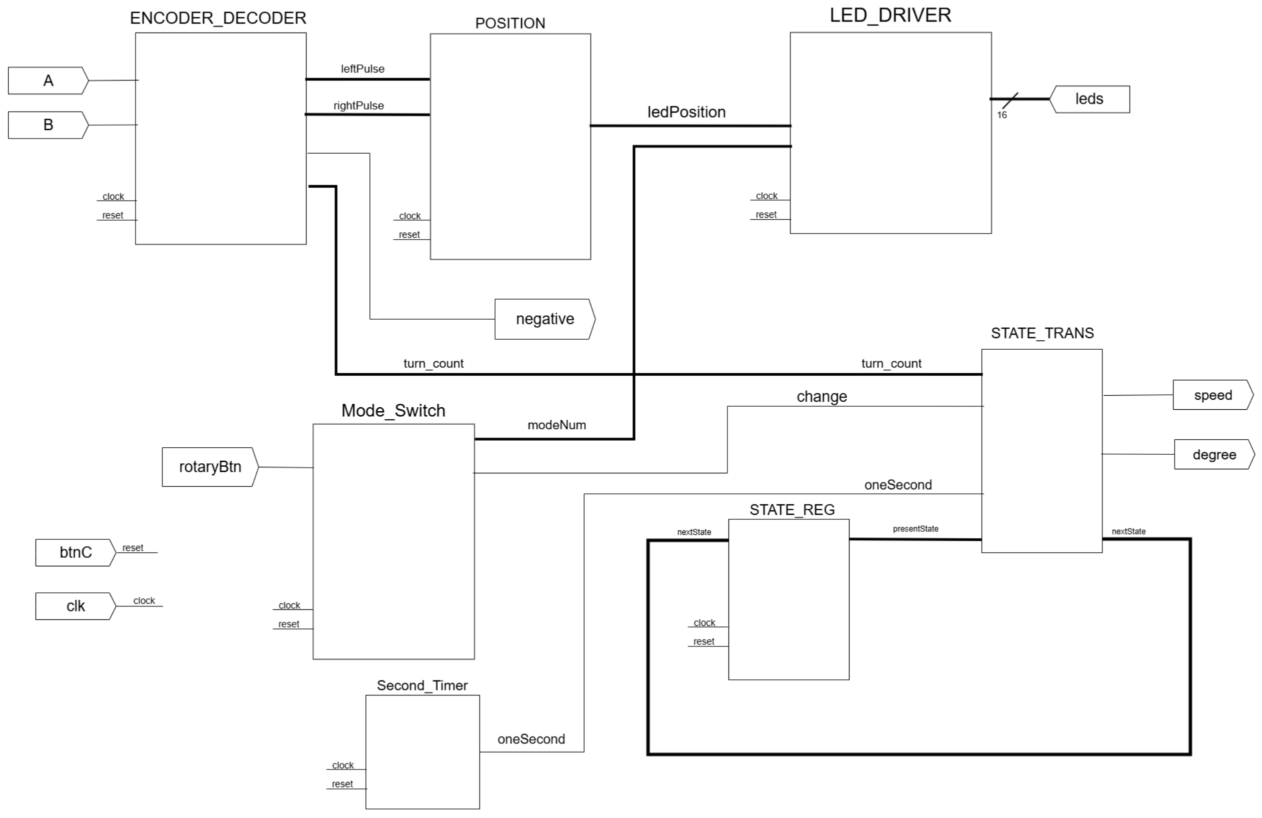


*Figure 4: Encoder Design Diagram*

*Encoder Code*

```
--================================================================
--Rotary encoder Single LED
--Cynthia Babecka and McCleland Idaewor
--
-- This design takes the input from the Pmod encoder, filters
-- it to drive the led position
--
--================================================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Encoder is
      Port (
            A:in std_logic;
            B: in std_logic;
            led: out std_logic_vector(15 downto 0);
            rotaryBtn: in std_logic;
            clock: in std_logic;
            reset: in std_logic;

            --seven seg outpus-----------
            speed    : out integer;
            degree   : out integer;
            negative : out std_logic
            );
end Encoder;

architecture Encoder_ARCH of Encoder is

--Encoder_Decoder---------------------------------------------
signal leftCount: integer := 0;
signal rightCount: integer := 0;
signal rightPulse: std_logic;
signal leftPulse: std_logic;
signal ledPosition: integer := 0;
signal currentState: std_logic_vector (1 downto 0);
signal pastState: std_logic_vector(1 downto 0);
constant ACTIVE: std_logic := '1';
signal A_sync, B_sync : std_logic_vector(1 downto 0);
signal turn_count: integer range 0 to 66;
signal degree_count: integer range 0 to 19;
signal inNegative: std_logic; --input for seven seg

--Mode_Switch-------------------------------------------------
signal change: std_logic;
signal modeNum: integer;
```

```vhdl
--State-----------------------------------------------------------------
type State_t is (RPM, ANGLE);
signal nextState: State_t;
signal presentState: State_t;
signal inDegree: integer;--input for seven seg
signal inSpeed: integer; --input for seven seg

--Second----------------------------------------------------------------
signal oneSecond: std_logic;

-----------------------------------------------------------------------
begin
     --==========================================================
     -- Filter AB inputs to detect direction and generate
     -- left/right pulses
     --==========================================================
     ENCODER_DECODER: process (clock, reset)
     begin
          if (reset = ACTIVE) then
              A_sync <= (others => '0');
              B_sync <= (others => '0');
              currentState <= "00";
              pastState <= "00";
              rightPulse <= '0';
              leftPulse  <= '0';
              turn_count <= 0;
              degree_count <= 0;
              inNegative <= '0';

          elsif (rising_edge(clock)) then
              A_sync(0) <= A;
              A_sync(1) <= A_sync(0);
              B_sync(0) <= B;
              B_sync(1) <= B_sync(0);
              currentState <= A_sync(1) & B_sync(1);


              if (currentState /= pastState) then
                  case pastState & currentState is
                       when "1101"|"0100"|"0010"|"1011" =>
                            leftCount <= leftCount + 1;
                            inNegative <= '0';
                            if (leftCount = 3) then
                                leftPulse <= '1';
                                rightPulse <='0';
                                leftCount <= 0;
                                turn_count <= turn_count + 1;
                                degree_count <= degree_count +1;
                                if (degree_count = 19) then
                                    degree_count <= 0;
                                end if;
                             end if;
                       when "1110"|"1000"| "0001"|"0111" =>
                            rightCount <= rightCount + 1;
                            inNegative <= '1';
                            if (rightCount = 3) then
                                leftPulse <='0';
                                rightPulse <='1';
                                rightCount <= 0;

                                turn_count <= turn_count +1;
                                degree_count <= degree_count - 1;
                                if (degree_count = 0) then
                                    degree_count <= 19;
                                end if;



                             end if;
                       when others =>
                            leftPulse <= '0';
                            rightPulse <= '0';

                   end case;
                   pastState <= currentState;


              else
                    leftPulse <= '0';
                    rightPulse <= '0';

              end if;


          end if;
          negative <= inNegative;
     end process;
     --============================================================
     --Updates Led position based on direction pulses
     --============================================================
     POSITION: process(clock, reset)--component block
     begin
            if (reset = ACTIVE) then
                 ledPosition <= 0;
            elsif (rising_edge(clock)) then
                  if rightPulse = '1' then
                        if ledPosition = 0 then
                           ledPosition <= 15;
```

```
                    else
                        ledPosition <= ledPosition - 1;
                    end if;
                elsif leftPulse = '1' then
                    if ledPosition = 15 then
                        ledPosition <= 0;
                    else
                        ledPosition <= ledPosition + 1;
                    end if;
                end if;
            end if;
    end process;

    --============================================================
    --Takes the Led position and in RPM mode turns the
    --corresponding Led on and all the others off, and in ANGLE
    --the corresponding led off and all other on.
    --============================================================
    LED_DRIVER: process (clock, reset)
    begin
           if (reset = ACTIVE) then
               led <= (others => '0');
               led(0)<= '1';
           elsif(rising_edge(clock)) then
               if (modeNum = 0) then
                   led <= (others => '0');
                   led(ledPosition) <= '1';
               elsif (modeNum = 1) then
                   led <= (others => '1');
                   led(ledPosition) <= '0';
               end if;
           end if;
    end process;

    --============================================================
    --
    --
    --============================================================
    STATE_REG: process (clock, reset)
    begin
         if (reset = ACTIVE) then
             presentState <= RPM;

         elsif (rising_edge(clock)) then
             presentState <= nextState;

         end if;
    end process;
```

```
--==============================================================
--Transitions the State between RPM and ANGLE depending on
--the input from the change signal
--==============================================================
STATE_TRANS: process(presentState, rotaryBtn, turn_count, degree_count)
variable speedUpdate : integer;
begin
     nextState <= presentState;
     case presentState is
            when RPM =>

                if (oneSecond = '1') then
                    speedUpdate := (turn_count*3);
                    inSpeed <= speedUpdate;
                end if;
                speed <= inSpeed;
                if (change = ACTIVE) then
                    nextState <= ANGLE;
                end if;
            when ANGLE =>
                degree <= degree_count*18;
                if (change = ACTIVE) then
                    nextState <= RPM;
                end if;

     end case;
 end process;

 --==============================================================
 --Reads the input of the rotary button to change the state
 --and led mode
 --==============================================================
 Mode_Switch: process(clock, reset)
 begin
      if (reset = ACTIVE) then
           modeNum <= 0;
      elsif (rising_edge(clock)) then
           if (rotaryBtn = ACTIVE) then
               modeNum <= modeNum + 1;
               if (modeNum = 1) then modeNum <= 0; end if;
               change <= '1';
           end if;
     end if;
 end process;

 --==============================================================
 --One second timer to drive the rpm calculations
 --==============================================================
 Second_Timer: process (clock)
```

```
        variable counter: integer;
        begin
              if (reset = ACTIVE) then
                  oneSecond <= '0';
                  counter := 0;
              elsif (rising_edge(clock)) then
                  if (counter = 49_999_999)then
                      counter := 0;
                      oneSecond <= '1';
                  else
                      counter := counter +1;
                      oneSecond <= '0';
                  end if;
              end if;
       end process;


end Encoder_ARCH;
```

*Debounce Code*

```
--================================================================
--Rotary encoder debouncer
--Cynthia Babecka and McCleland Idaewor
--
-- This is the debouncer used for the rotary encoder
--
--================================================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Debouncer is
    generic (
        DEBOUNCE_COUNT : natural := 100_000  -- default: 1 ms at 100 MHz
    );
    port (
        clk  : in  std_logic;
        --signals from 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

    signal counterA : unsigned(31 downto 0) := (others => '0');
    signal counterB : unsigned(31 downto 0) := (others => '0');
    signal stableA  : std_logic := '0';
    signal stableB  : std_logic := '0';
    signal prevA    : std_logic := '0';
    signal prevB    : std_logic := '0';

begin

    process(clk)
    begin
        if rising_edge(clk) then
            --Debounce A----------
            if Ain /= prevA then --checks current input
                counterA <= (others => '0');
                prevA    <= Ain;
             --checks debounce threshold---------
            elsif counterA < to_unsigned(DEBOUNCE_COUNT, counterA'length) then
                counterA <= counterA + 1;
            elsif stableA /= Ain then --assigns stable signal
```

```
                stableA <= Ain;
            end if;

            --Debounce B---------------
            if Bin /= prevB then --checks current input
                counterB <= (others => '0');
                prevB    <= Bin;
             --checks debounce threshold------------
            elsif counterB < to_unsigned(DEBOUNCE_COUNT, counterB'length) then
                counterB <= counterB + 1;
            elsif stableB /= Bin then --assigns stable signal
                stableB <= Bin;
            end if;
        end if;
    end process;

    --assigns stable signal to output
    Aout <= stableA;
    Bout <= stableB;

end Behavioral;
```

## *deccimalSeg Code*

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

--*****************************************************************************
*
--*
--* Name: decimalSegmentDriver
--* Designer: Cynthia Babecka and McCleland Idaewor
--*
--*     This component serves as a numeric driver for a 4 digit seven-segment
--*     display with shared segments and selectable anodes.  It can display
--*     any numeric digit represented by a 4-bit value ('0'-'9'),
--*     negative values, and provides the ability to selectively blank any
--*     digit of the display using blanking control input signals.
--*
--*     This component is designed to multiplex the 4 seven-segment displays
--*      at a scan rate of 1 kHz.
--*
--*****************************************************************************
*


entity decimalSeg is
    port(
        reset: in std_logic;
        clock: in std_logic;

        digit3: in std_logic_vector(3 downto 0);    --leftmost digit
        digit2: in std_logic_vector(3 downto 0);    --2nd from left digit
        digit1: in std_logic_vector(3 downto 0);    --3rd from left digit
        digit0: in std_logic_vector(3 downto 0);    --rightmost digit

        blank3: in std_logic;    --leftmost digit
        blank2: in std_logic;    --2nd from left digit
        blank1: in std_logic;    --3rd from left digit
        blank0: in std_logic;    --rightmost digit

        decSegs:   out std_logic_vector(6 downto 0);    --MSB=g, LSB=a
        anodes:    out std_logic_vector(3 downto 0)    --MSB=leftmost digit
    );
end decimalSeg;

architecture decimalSeg_ARCH of decimalSeg is

    ----general
definitions----------------------------------------------CONSTANTS
```

```
    constant ACTIVE: std_logic := '1';
    constant COUNT_1KHZ: integer := (100000000/1000)-1;

    ----anode settings for digit
selection--------------------------------CONSTANTS
    constant SELECT_DIGIT_0: std_logic_vector(3 downto 0)    := "1110";
    constant SELECT_DIGIT_1: std_logic_vector(3 downto 0)    := "1101";
    constant SELECT_DIGIT_2: std_logic_vector(3 downto 0)    := "1011";
    constant SELECT_DIGIT_3: std_logic_vector(3 downto 0)    := "0111";
    constant SELECT_NO_DIGITS: std_logic_vector(3 downto 0) := "1111";

    ----decimal seven segment
display------------------------------------CONSTANTS
    constant ZERO_7SEG: std_logic_vector(6 downto 0)   := "1000000";
    constant ONE_7SEG: std_logic_vector(6 downto 0)    := "1111001";
    constant TWO_7SEG: std_logic_vector(6 downto 0)    := "0100100";
    constant THREE_7SEG: std_logic_vector(6 downto 0)  := "0110000";
    constant FOUR_7SEG: std_logic_vector(6 downto 0)   := "0011001";
    constant FIVE_7SEG: std_logic_vector(6 downto 0)   := "0010010";
    constant SIX_7SEG: std_logic_vector(6 downto 0)    := "0000010";
    constant SEVEN_7SEG: std_logic_vector(6 downto 0)  := "1111000";
    constant EIGHT_7SEG: std_logic_vector(6 downto 0)  := "0000000";
    constant NINE_7SEG: std_logic_vector(6 downto 0)   := "0011000";
    constant A_7SEG: std_logic_vector(6 downto 0)      := "0001000";
    constant B_7SEG: std_logic_vector(6 downto 0)      := "0000011";
    constant C_7SEG: std_logic_vector(6 downto 0)      := "1000110";
    constant D_7SEG: std_logic_vector(6 downto 0)      := "0100001";
    constant MINUS_SIGN: std_logic_vector(6 downto 0)  := "0111111";
    constant BLANK: std_logic_vector(6 downto 0)       := "1111111";

    ----internal
connections---------------------------------------------SIGNALS
    signal enableCount: std_logic;
    signal selectedBlank: std_logic;
    signal selectedDigit: std_logic_vector(3 downto 0);
    signal digitSelect: unsigned(1 downto 0);

begin

--============================================================================
    --  Convert 4-bit binary value into its equivalent 7-segment pattern

--============================================================================
    BINARY_TO_7SEG: with selectedDigit select
        decSegs <=   ZERO_7SEG  when "0000",
                     ONE_7SEG   when "0001",
                     TWO_7SEG   when "0010",
                     THREE_7SEG when "0011",
                     FOUR_7SEG  when "0100",
                     FIVE_7SEG  when "0101",
                     SIX_7SEG   when "0110",
                     SEVEN_7SEG when "0111",
                     EIGHT_7SEG when "1000",
                     NINE_7SEG  when "1001",
                     A_7SEG     when "1010",
                     B_7SEG     when "1011",
                     C_7SEG     when "1100",
                     D_7SEG     when "1101",
                     MINUS_SIGN when "1110",
                     BLANK      when others;



--==============================================================================
    --  Select the current digit to display

--==============================================================================
    DIGIT_SELECT: with digitSelect select
        selectedDigit <= digit0 when "00",
                         digit1 when "01",
                         digit2 when "10",
                         digit3 when others;




--==============================================================================
    --  Select the current digit to display

--==============================================================================
    BLANK_SELECT: with digitSelect select
        selectedBlank <= blank0 when "00",
                         blank1 when "01",
                         blank2 when "10",
                         blank3 when others;





--==============================================================================
    --  Select the current digit in the seven-segment display unless the
    --  selectedBlank input is active.

--==============================================================================
    ANODE_SELECT: process(selectedBlank, digitSelect)
    begin
        if (selectedBlank = ACTIVE) then
            anodes <= SELECT_NO_DIGITS;
        else
            case digitSelect is
```

```
                when "00" =>    anodes <= SELECT_DIGIT_0;
                when "01" =>    anodes <= SELECT_DIGIT_1;
                when "10" =>    anodes <= SELECT_DIGIT_2;
                when others =>  anodes <= SELECT_DIGIT_3;
            end case;
        end if;
    end process ANODE_SELECT;



--==============================================================================
    --  Set the scan rate for the multiplexed seven-segment displays to 1 kHz.
    --  The enableCount output pulses for one clock cycle at a rate of 1 kHz.

--==============================================================================
    SCAN_RATE: process(reset, clock)
        variable count: integer range 0 to COUNT_1KHZ;
    begin
        --manage-count-value--------------------------------------------
        if (reset = ACTIVE) then
            count := 0;
        elsif (rising_edge(clock)) then
            if (count = COUNT_1KHZ) then
                count := 0;
            else
                count := count + 1;
            end if;
        end if;

        --update-enable-signal-------------------------------------------
        enableCount <= not ACTIVE;  --default value unless count reaches 
terminal
        if (count=COUNT_1KHZ) then
            enableCount <= ACTIVE;
        end if;
    end process SCAN_RATE;



--==============================================================================
    --  Generates the digit selection value

--==============================================================================
    DIGIT_COUNT: process(reset, clock)
    begin
        if (reset = ACTIVE) then
            digitSelect <= "00";
        elsif (rising_edge(clock)) then
            if (enableCount = ACTIVE) then
```

```
                digitSelect <= digitSelect + 1;
            end if;
        end if;
    end process DIGIT_COUNT;
```

## Test Bench

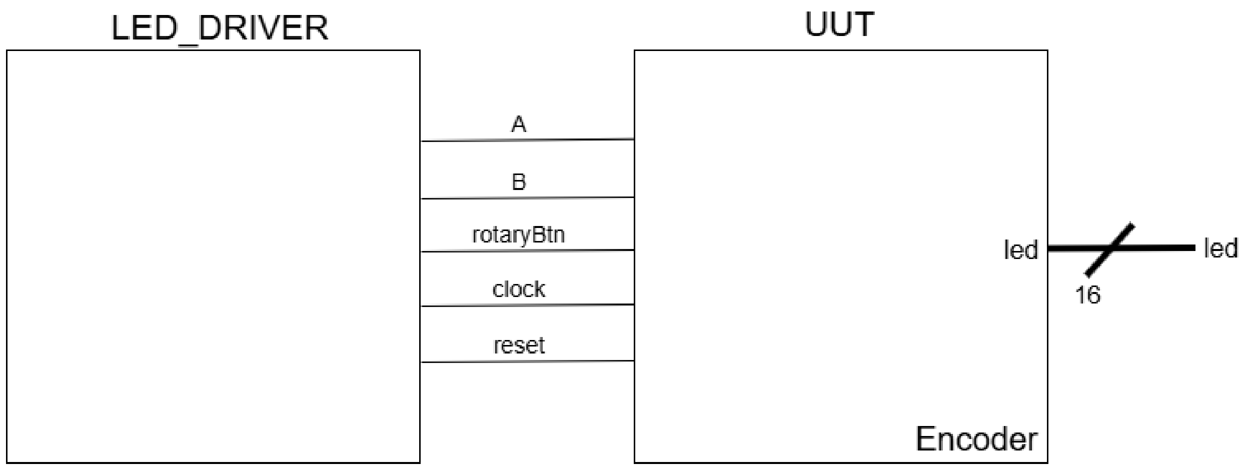
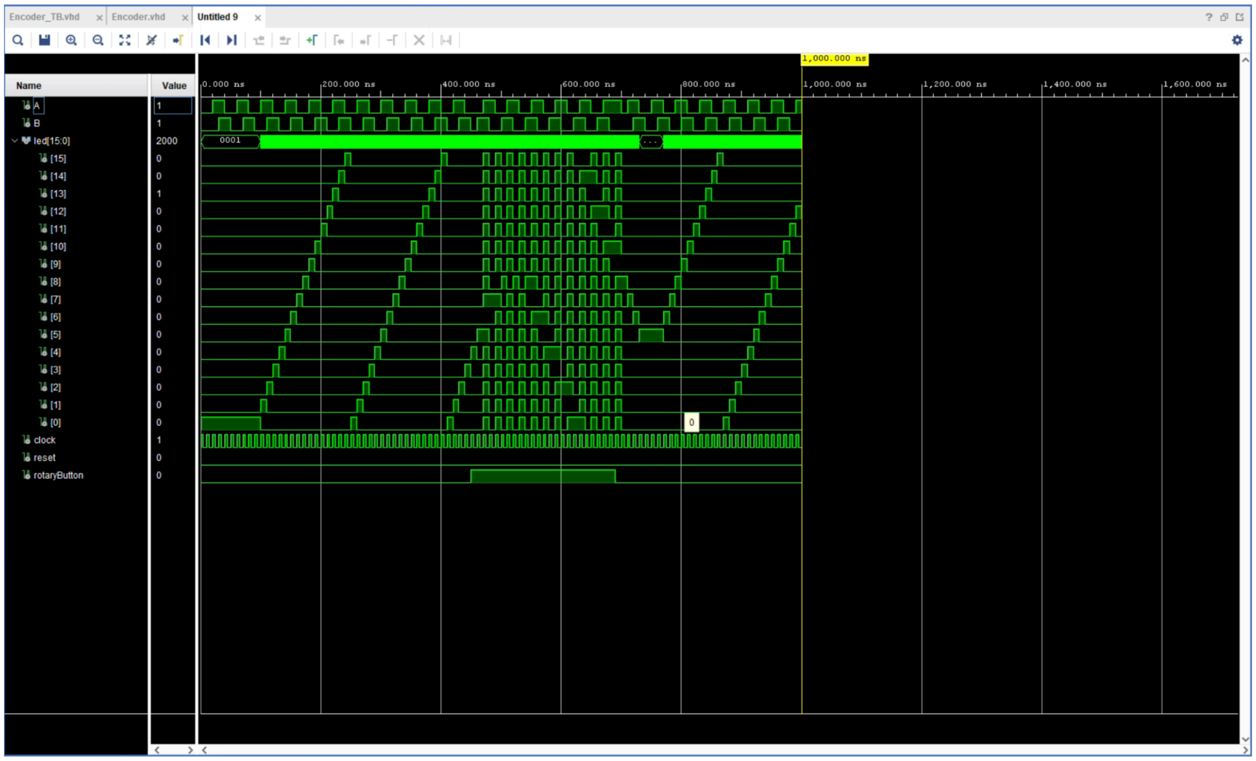


*Figure 5: Encoder Testbench Diagram*



| Name | Value |
|---|---|
| A | 1 |
| B | 1 |
| led[15:0] | 2000 |
| [15] | 0 |
| [14] | 0 |
| [13] | 1 |
| [12] | 0 |
| [11] | 0 |
| [10] | 0 |
| [9] | 0 |
| [8] | 0 |
| [7] | 0 |
| [6] | 0 |
| [5] | 0 |
| [4] | 0 |
| [3] | 0 |
| [2] | 0 |
| [1] | 0 |
| [0] | 0 |
| clock | 1 |
| reset | 0 |
| rotaryButton | 0 |

*Figure 6: Encoder Testbench Waveforms*

## *Test Bench Code*

```
------------------------------------------------------
-- Name: Encoder_TB
-- Designers: Cynthia Babecka and McCleland Idaewor
--
--
-- This is the testbench file for the rotary encoder
-- component. It tests all the possible signal
-- combinations based on the position of the rotary
-- encoder, the encoder button, and the reset button
--
------------------------------------------------------
---
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;



entity encoder_TB is
--  Port ( );
end encoder_TB;

architecture encoder_TB_ARCH of encoder_TB is

component encoder is
      Port (
            A:in std_logic;
            B: in std_logic;
            led: out std_logic_vector(15 downto 0);
            rotaryButton: in std_logic;
            clock: in std_logic;
            reset: in std_logic);
end component;

signal A: std_logic :='0' ;
signal B: std_logic :='0' ;
signal led: std_logic_vector (15 downto 0);
signal clock: std_logic :='0';
signal reset: std_logic :='0';
signal rotaryButton: std_logic := '0';

begin
        UUT: encoder
            port map(
                     A => A, B =>B, led => led,rotaryButton => rotaryButton,
                     clock => clock, reset => reset);

        clockDriver: process
```

```
begin
       while true loop
           clock <= '1';
           wait for 5 ns;
           clock <= '0';
           wait for 5 ns;
       end loop;
end process;


encoder_Driver: process --(clock, reset)
begin
       reset <= '1'; wait until rising_edge(clock);
       reset <= '0'; wait until rising_edge(clock);
       for i in 0 to 10 loop
             A <= '0'; B <= '0'; wait for 10 ns;
             A <= '1'; B <= '0'; wait for 10 ns;
             A <= '1'; B <= '1'; wait for 10 ns;
             A <= '0'; B <= '1'; wait for 10 ns;
       end loop;
       wait until rising_edge(clock);
       rotaryButton <= '1';
       for i in 0 to 5 loop
            A <= '0'; B <= '0'; wait for 10 ns;
            A <= '0'; B <= '1'; wait for 10 ns;
            A <= '1'; B <= '1'; wait for 10 ns;
            A <= '1'; B <= '0'; wait for 10 ns;
       end loop;
       wait until rising_edge(clock);
       rotaryButton <= '0';
       wait until rising_edge(clock);
       for i in 0 to 15 loop
             A <= '0'; B <= '0'; wait for 10 ns;
             A <= '1'; B <= '0'; wait for 10 ns;
             A <= '1'; B <= '1'; wait for 10 ns;
             A <= '0'; B <= '1'; wait for 10 ns;
       end loop;
       --rotaryButton <= '0';
       wait;
end process;
```

## Basys3 Wrapper

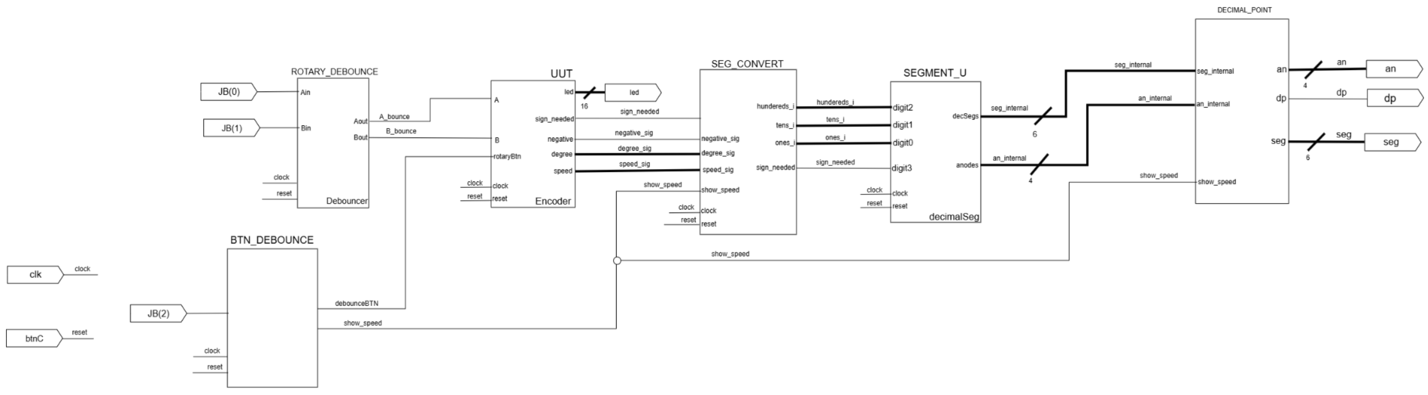


*Figure 3:Encoder Wrapper for Basys3*

## Wrapper code

```vhdl
----------------------------------------------------------------------------------
----
-- Name: Encoder_BASYS3
-- Designers: Cynthia Babecka and McCleland Idaewor
--
-- Wrapper file for the Rotary Encoder. This wrapper
-- is targeting the Basys3 board from Diglient. The Configuration
-- file for the basys3 was retrieved from the diglent GitHub account
--
-- The Rotary Encoder moves the leds on the board and based on the mode
-- selected by the rotary button, will indicate whether the board is in
-- speed or degree mode with the seven segment display reflecting the
-- selected mode respectively
----------------------------------------------------------------------------------
----

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Encoder_BASYS3 is
    Port (
         JB       : in std_logic_vector(2 downto 0);
         btnC     : in std_logic;
         clk      : in std_logic;
         led      : out std_logic_vector(15 downto 0);
         dp       : out std_logic;
         seg      : out std_logic_vector(6 downto 0);
         an       : out std_logic_vector(3 downto 0)
    );
end Encoder_BASYS3;

architecture Encoder_BASYS3_ARCH of Encoder_BASYS3 is

    component Encoder is
        Port (
              A          : in std_logic;
              B          : in std_logic;
              rotaryBtn  : in std_logic;
              led        : out std_logic_vector(15 downto 0);
              clock      : in std_logic;
              negative        : out std_logic;
              degree     : out integer;
              speed      : out integer;
              reset      : in std_logic);
    end component;

    component decimalSeg is
        Port (
        reset: in std_logic;
        clock: in std_logic;

        digit3: in std_logic_vector(3 downto 0);
        digit2: in std_logic_vector(3 downto 0);
        digit1: in std_logic_vector(3 downto 0);
        digit0: in std_logic_vector(3 downto 0);



        decSegs: out std_logic_vector(6 downto 0);
        anodes:  out std_logic_vector(3 downto 0)
        );
    end component;

    component Debouncer is
        generic(
                DEBOUNCE_COUNT : natural := 100_000 -- 1 ms at 100 MHz
                );
        Port(  clock : 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 component;

    -- -------------------------
    -- Internal signals
    -- -------------------------
    signal A_bounce : std_logic;
    signal B_bounce : std_logic;

    -- internal signals to receive Encoder outputs---------
    signal speed_sig    : integer := 0;
    signal degree_sig   : integer := 0;
    signal negative_sig : std_logic := '0';
    signal led_raw : std_logic_vector(15 downto 0);



    --Internal signals for rotaryBTN-----------------------------
    signal btn_sync     : std_logic_vector(1 downto 0) := (others => '0');
    signal btn_prev     : std_logic := '0';
    signal debounceBTN  : std_logic := '0';

    --Mode toggle-----------------------------------------------
    signal show_speed : std_logic := '0';--0 = show speed, 1 = show degree

    --Display conversion signals------------------------------------
    signal hundreds_i   : integer range 0 to 9 := 0;
    signal tens_i       : integer range 0 to 9 := 0;
    signal ones_i       : integer range 0 to 9 := 0;
    signal sign_needed  : std_logic := '0';

    --4-bit digit vectors for decimalSeg-------------------------
    signal digit3, digit2, digit1, digit0 : std_logic_vector(3 downto 0) :=
(others => '0');

    --decimalSeg outputs (no DP)------
    signal seg_internal : std_logic_vector(6 downto 0);
    signal an_internal  : std_logic_vector(3 downto 0);


begin

   ROTARY_DEBOUNCE: Debouncer
       generic map (
            DEBOUNCE_COUNT => 100_000  -- 1 ms at 100 MHz
       )
       port map (
                   Ain => JB(0),
                   Bin => JB(1),
                   clock => clk,
                   Aout => A_bounce,
                   Bout => B_bounce);

      --rotary button debounce--------------------
    BTN_DEBOUUNCE:  process(clk)
    begin
        if rising_edge(clk) then
            -- two-stage synchronizer
            btn_sync(0) <= JB(2);
            btn_sync(1) <= btn_sync(0);

            -- rising-edge detection -> one-clock pulse
            if (btn_sync(1) = '1' and btn_prev = '0') then
                debounceBTN <= '1';
            else
                debounceBTN <= '0';
            end if;

            btn_prev <= btn_sync(1);
        end if;
    end process;
  SEGMENT_U: decimalSeg
      port map (
                 reset => btnC,
                 clock => clk,

                 digit3 => digit3,
                 digit2 => digit2,
                 digit1 => digit1,
                 digit0 => digit0,

                 decSegs => seg_internal,
                 anodes => an_internal);

      UUT: Encoder
      Port map (
                 A => A_bounce,
                 B => B_bounce,
                 rotaryBtn => debounceBTN,
                 clock => clk,
                 reset => btnC,
                 led => led,
                 negative => negative_sig,
                 degree  => degree_sig,
                 speed   => speed_sig
                );



------------------------------------------------------------------------------
    -- Button-driven mode toggle (press to toggle)

------------------------------------------------------------------------------
    process(clk)
    begin
        if rising_edge(clk) then
            if debounceBTN = '1' then
                show_speed <= not show_speed;
            end if;
        end if;
    end process;


    --Convert integer "number" (range -999 to 999) into sign +
hundreds/tens/ones----
    SEG_CONVERT: process(show_speed, speed_sig, degree_sig)
        variable n : integer;
       -- variable q: integer;

    begin
        if (show_speed = '0') then
            n := speed_sig;
            if (negative_sig ='1') then sign_needed <= '1';
            else sign_needed <= '0';
            end if;

        elsif (show_speed = '1') then
            n := degree_sig;
            sign_needed <= '0';

        end if;

             hundreds_i <= (n / 100) mod 10;
             tens_i     <= (n / 10) mod 10;
             ones_i     <= n mod 10;

    end process;


-------------------------------------------------------------------------------
--
    -- Map the integer digits to 4-bit vectors expected by decimalSeg.
    -- If sign_needed = '1', show minus sign in the
    -- leftmost digit (decimalSeg maps "1110" -> minus).

-------------------------------------------------------------------------------
--
    process(clk)
    begin
         if (rising_edge(clk)) then
```

```
                if (sign_needed = '1') then
                    digit3 <= "1110" ;
                elsif (sign_needed = '0') then
                    digit3 <= "0000";
                end if;
                digit2 <= std_logic_vector(to_unsigned(hundreds_i,4));
                digit1 <= std_logic_vector(to_unsigned(tens_i,4));
                digit0 <= std_logic_vector(to_unsigned(ones_i,4));
            end if;

    end process;

----------------------------------------------------------------------------
    -- Blanking policy (decimalSeg uses ACTIVE='1' to BLANK).
    -- Here we blank leading thousands if not needed; keep others visible.

----------------------------------------------------------------------------


-----------------------------------------------------------------------------
    -- Decimal point (DP) logic:
    -- We assert DP (active-LOW) only while the target digit anode is active.
    -- Requirement:
    --   - RPM mode  -> light far-left decimal point (leftmost digit)
    --   - ANGLE mode -> light far-right decimal point (rightmost digit)
    --
    -- decimalSeg uses active-LOW anodes:
    --   leftmost  => "0111"
    --   rightmost => "1110"

-----------------------------------------------------------------------------
    DECIMAL_POINT: process(an_internal, show_speed, seg_internal)
        begin
            if (show_speed = '0' and an_internal = "0111") then
                dp <= '0';
            elsif (show_speed = '1' and an_internal = "1110") then
                dp <= '0';
            else
                dp <= '1';
            end if;

            seg(6 downto 0) <= seg_internal;
            an <= an_internal;
        end process;



end Encoder_BASYS3_ARCH;
```