```
------------------------------------------------------------
-- CWRUCutter_PWMInput.vhd
-- EJ Kreinar
--
-- Measures a PWM signal in ticks
--
-- Inputs:
--   PWM_IN: the PWM signal to measure
--
-- Outputs:
--   POS_PULSE_LEN: # Ticks that the PWM was high
--   CYC_PULSE_LEN: # Ticks to complete an entire cycle (rising-edge to rising-edge)
--
-- Notes:
--
-- History
-- 9/19: ejk43- Created
------------------------------------------------------------

Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity CWRUCutter_PWMInput is
    port (
        CLK           : in  std_logic;
        aRESET        : in  std_logic;
        PWM_IN        : in  std_logic;
        POS_PULSE_LEN : out std_logic_vector(31 downto 0) := (others => '0');
        CYC_PULSE_LEN : out std_logic_vector(31 downto 0) := (others => '0')
      );
end CWRUCutter_PWMInput;

architecture rtl of CWRUCutter_PWMInput is
    -- At 40MHz, MAX_CYCLE_LEN 800000 ticks will be a 20ms period timeout
    constant MAX_CYCLE_LEN: integer := 800000;

    -- Set DEBOUNCE_LEN to .1ms (4000 ticks @ 40MHz)
    -- Do not allow a transisiton to occur earlier than
    -- the DEBOUNCE_LEN after a transition
    constant DEBOUNCE_LEN:  integer := 4000;

    signal pwm_dly: std_logic_vector(1 downto 0);
    signal pos_len: unsigned(31 downto 0);
    signal cyc_len: unsigned(31 downto 0);
    signal pos_reg: unsigned(31 downto 0);
    signal cyc_reg: unsigned(31 downto 0);
    signal debounce_count: unsigned(15 downto 0);
    signal debounce_allow: std_logic;
    signal cyc_debounce  : std_logic;
    signal transition    : std_logic;
begin
    POS_PULSE_LEN <= std_logic_vector(pos_reg);
    CYC_PULSE_LEN <= std_logic_vector(cyc_reg);
```

```
process(aRESET, CLK) begin
  if(aRESET = '1') then
    pwm_dly         <= (others => '1');
    pos_len         <= (others => '0');
    cyc_len         <= (others => '0');
    debounce_count  <= (others => '0');
    debounce_allow  <= '0';
    cyc_debounce    <= '0';
    transition      <= '0';
    pos_reg         <= (others => '0');
    cyc_reg         <= (others => '0');
  elsif rising_edge(clk) then
    pwm_dly(0) <= PWM_IN;
    pwm_dly(1) <= pwm_dly(0);

    -- PWM TRANSITIONS
    -- Look at the previous PWM values and react based on
    -- whether PWM is high, low, rising, or falling
    -- Also signal whether a transition occured so we can perform debouncing
    if cyc_len = MAX_CYCLE_LEN then
        -- 100% OR 0% DUTY CYCLE
        -- Make sure that we dont get stuck if the PWM is
        -- completely high or completely low
        pos_reg <= pos_len;
        cyc_reg <= cyc_len;
        pos_len <= (others => '0');
        cyc_len <= (others => '0');
        transition <= '0';
    elsif (pwm_dly = "00") then
        -- PWM IS LOW
        -- increment cycle count
        cyc_len <= cyc_len + 1;
        transition <= '0';
    elsif (pwm_dly = "11") then
        -- PWM IS HIGH
        -- increment positive count and cycle count
        pos_len <= pos_len + 1;
        cyc_len <= cyc_len + 1;
        transition <= '0';
    elsif (pwm_dly = "10" and debounce_allow = '1' and transition = '0') then
        -- FALLING EDGE
        -- On a falling edge, save previous Positive length and reset value
        pos_reg <= pos_len;
        pos_len <= (others => '0');
        cyc_len <= cyc_len + 1;
        transition <= '1';
    elsif (pwm_dly = "01" and debounce_allow = '1' and transition = '0') then
        -- RISING EDGE
        -- On a rising edge, save previous Cycle length and reset values
        cyc_reg <= cyc_len;
        cyc_len <= (others => '0');
        transition <= '1';
    end if;
```

```
        -- DEBOUNCE
        -- Debouncing starts when we see a transition, and then we allow
        -- a new transition to occur after debounce_count = DEBOUNCE_LEN
        if transition = '1' then
            debounce_count <= (others => '0');
            debounce_allow <= '0';
        elsif debounce_count = DEBOUNCE_LEN then
            debounce_allow <= '1';
        else
            debounce_count <= debounce_count + 1;
        end if;

      end if;
    end process;
end rtl;
```