RIT | Rochester Institute of Technology

Date: February 12nd

From: Jaric Sloan

To: Dr. Kaputa

Subject: PVM module

Introduction

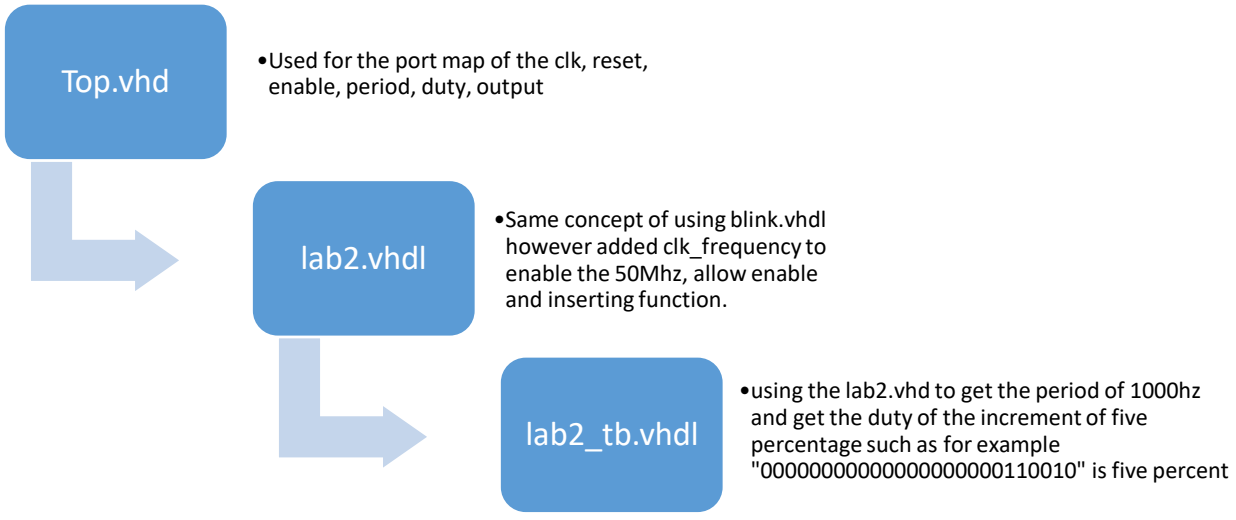
For the PVM module, the three different sources were used to give me assistance and use their concept to meet the requirements. The three different sources are blink, generic counter, and rising edge synchronizer is used to help reach the requirements as shown below. The period and duty input ports were made into 26-bit vectors to support the period up to one second. Both period and duty inputs were meet the desired frequency for the period to be able to meet the duty cycle of the given percentage of zero percent to 100 percent. The vector sizes were calculated with the provided of 50MHz system clock to support one second of the period such as (log (50000000)/1)/log (2) is approximately 25.6 bits, therefore, using 26 bits is the best idea. By using the generic counter example to use the internal count signal to keep track of the increments.

- To have an enable pulse
- To be able to control both the duty cycle and the period of the PVM pulse train
- Ensure the entity inputs not 'generics' for the period and duty inputs

PVM Module will be beneficial to Drone to be able to manifest itself in the motor.

Analysis

By using the similar code fundamental to the Blink file that was provided, the fundamental is still the same for the lab 2 code fundamental.



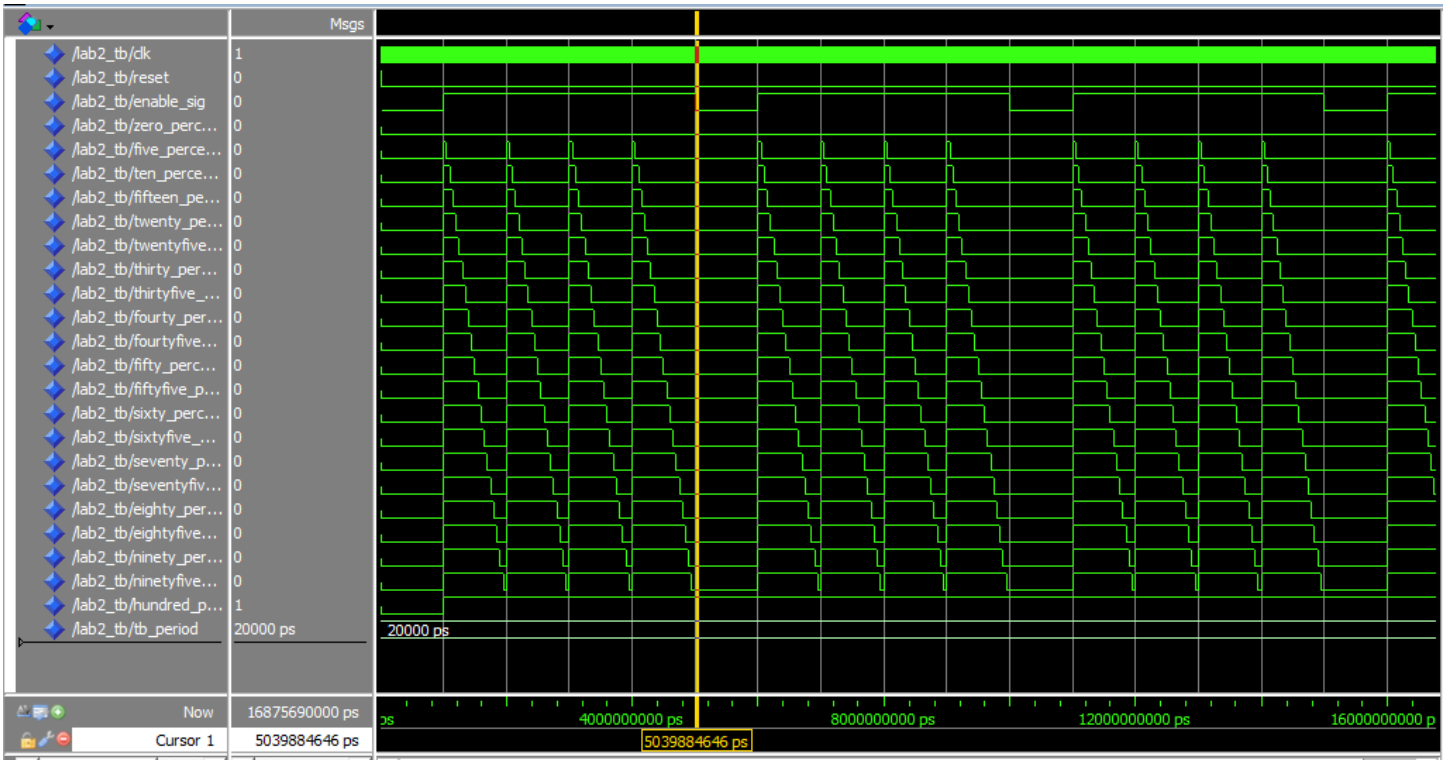


Figure 1:Result of lab2_tb

As shown in Figure 1, shown the results of the lab2_tb of increment of five percentage out of 0-100 percent to show a better picture of what it looks like. In the lab2_tb, a 1 KHz period was used for different cases of duty. The duty remains constant as for other variables remain constant and it would be great if can test the difference of duty changes and something happens during a real-time event. But it does meet all the requirements as shown in Introduction however to control the duty cycle and the period of the PVM pulse train remains remain constant.

Conclusion

Since the analysis determine that to control the duty cycle and the period of the PVM pulse train was remain constant. As shown in Figure 1 shows all the possibilities result from various output signals of the increments of five for 0-100 percentage of the constant duty cycles. It's required more testing than forcing them to be constant vectors however it is running smoothly.

Code Appendix

```
--------------------------------------------------------------------------------
-- Jaric Sloan
-- Lab2
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity top is
  port (
    clk             : in  std_logic;
    reset           : in  std_logic;
    enable          : in  std_logic;
    period          : in  std_logic_vector(25 DOWNTO 0);
    duty            : in  std_logic_vector(25 DOWNTO 0);
    output          : out std_logic
  );
end top;

architecture beh of top is

component lab2 is
  port (
    clk             : in  std_logic;
    reset           : in  std_logic;
    enable          : in  std_logic;
    period          : in  std_logic_vector(25 DOWNTO 0);
    duty            : in  std_logic_vector(25 DOWNTO 0);
    output          : out std_logic
  );
end component;

begin

uut: lab2
  port map(
    clk       => clk,
    reset     => reset,
    enable    => enable,
    period    => period,
    duty      => duty,
    output    => output
  );
end beh;
```

Figure 2: Top.vdhl file shown that lab2 clk are getting clk from the top

```
architecture beh of lab2  is

constant clk_freq    : integer := 50000000;  -- 50MHz
constant clk_period : time := 20ns; -- 50Mhz

signal count_sig     : integer range 0 to clk_freq := 0;  --same as blink
signal output_sig    : std_logic; -- same as blink
signal enable_sig    : std_logic;

begin
process(clk,reset)
  constant duty_zeros     : std_logic_vector(duty'range) := (others => '0'); -- whatever is in duty range and then the rest are zeros
  variable ticks_period : integer := clk_freq/to_integer(unsigned(period));  -- to find the ticks of the period
  begin
    if (reset = '1') then  -- if reset then all is zero, same as blink
      count_sig <= 0;
      output_sig <= '0';
    elsif (clk'event and clk = '1') then
      if (enable_sig = '1') then -- if enable is on then do this
        if (period = duty) then -- if period is same as duty then output will be one high
          output_sig <= '1'; -- ouput_sig will be high
        elsif (duty = duty_zeros) then -- otherwise its is zero
          output_sig <= '0';
        else
          if (count_sig <= (to_integer(unsigned(duty)) * (ticks_period/(to_integer(unsigned(period)))))) then
            output_sig <= '1'; -- will be high
          else
            output_sig <= '0'; -- will be low
          end if;
          if (count_sig = ticks_period) then
            count_sig <= 0; -- will be low
            output_sig <= not output_sig; -- opposite of count_sig
          else
            count_sig <= count_sig + 1; -- increments
          end if;
        end if;
      end if;
    end if;
  end process;
  enable_sig <= enable;
  output <= output_sig; -- output depends on the high and low from if statements
end beh;
```

Figure 3 : lab2.vhdl to calculate the count_sig depends on the duty and ticks of period to show the high and low of the duty percentage

```
begin
-- clock process
clock: process -- same as blink_tb
  begin
    clk <= not clk;
    wait for tb_period/2;
end process;

-- reset process
async_reset: process -- same as blink_tb
  begin
    wait for 2 * tb_period;
    reset <= '0';
    wait;
end process;

-- enable process
enable: process
  begin
    wait for 1ms; -- wait for one milliseconds
    enable_sig <= '1';
    -- disable
    wait for 4ms; -- wait for four milliseconds
    enable_sig <= '0';
end process;


uut0: lab2
  port map(
    clk       => clk,
    reset     => reset,
    enable    => enable_sig,
    period    => "00000000000000001111101000",  --1000
    duty      => "00000000000000000000000000",  --0 percent
    output    => zero_percentage
  );

uut1: lab2
  port map(
    clk       => clk,
    reset     => reset,
    enable    => enable_sig,
    period    => "00000000000000001111101000",  --1000
    duty      => "00000000000000000000110010",  --5 percent
    output    => five_percentage
  );
```

Figure 4: the test bench of lab 2 show the duty percentage of 0 to 100, period of 1000Hz