Saadet Büşra ÇAM-22201857
01.04.2024
EE102-1 Lab 6 Report

LAB 6: Arbitrary Waveform Generation

Purpose

The purpose of this lab is to learn displaying waveforms we choose on oscilloscope with assistance of basys3 and VHDL. Additionally, we can change the waveforms with inputs.

Design

On this lab, since I intend to used servo motors in my ee102 project, I choose to display PWM(pulse width modulation) method. In order to achieve square waves with this method, I used clocking method said as in lab 6 instructions. I choose clocking wizard IP and from there, I get 1 clk_in1 input, 1 clk_out output, and choose reset and locked ports additionally. Since, I had 2 input and 2 output in my clock. However, I almost didn't use locked one and barely used reset. The input clock frequency is 100MHz, clk_in1. In PWM method, 120 Hz frequency is optimum value in achieving maximum productivity, however, since clocking wizard IP's minimum frequency is 10 MHz, I couldn't choose it. Hence, I didn't want to change the frequency. My output clock frequency is 100MHz, same as input.

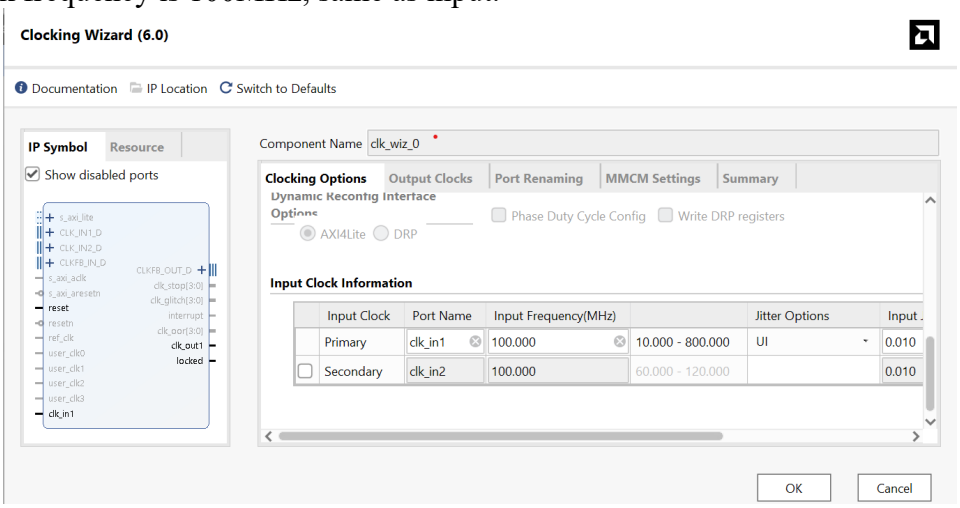

Figure 1: Clocking Wizard

CLK_IN :input clock
reset_n : reset
EN : enable
DUTY_CYCLE: Vector(num_bits-1 downto 0);
PWM_OUT: output signal

Methodology

On my code, on entity declaration and architecture part, I implemented the clock module I assigned and the wizard clock to each other. I keep track of the signal cycle count with counter.

This process increments the counter when EN is high and CLK_IN is on rising edge.(Flip flop) The reset signal is used to asynchronously reset counter to zero. Additionally, I add the clock's information from IP sources, and wizard instantiation template. generate clock signal clk_out1 accordingly clk_in1. This out signal is PWM_OUT. The duty_cyle is a vector with length of num_bits-1 to 0. It specifies the duty cycle of PWM signal. Duty cycle determines PWM's high percentage in comparison with its period. PWM_OUT is the output signal. It is updated based on duty_cycle and counter.
Additionally, the "if rising edge" process is working as initially, checking when CLK_IN signal goes from 0 to 1(high). If rising edge detected, the statement ends with "end if". On this part, the counter increases only the rising edges of the signal. PWM_generator uses it to update pwm_out signal based on duty_cycle input and counter value, only when clock rises and EN is 1.

Results

On this lab, my code is failed initially. I get incorrect simulation result, and the display on the oscilloscope wasn't as desired(the squares weren't so clear, and even I do autoset, the display again couldn't work). Additionally, after doing some changes, my code couldn't be synthesized. However, I get the simulation results as can be seen from figure-3 later on my works. Additionally, I get the oscilloscope waveform correctly after some differences.

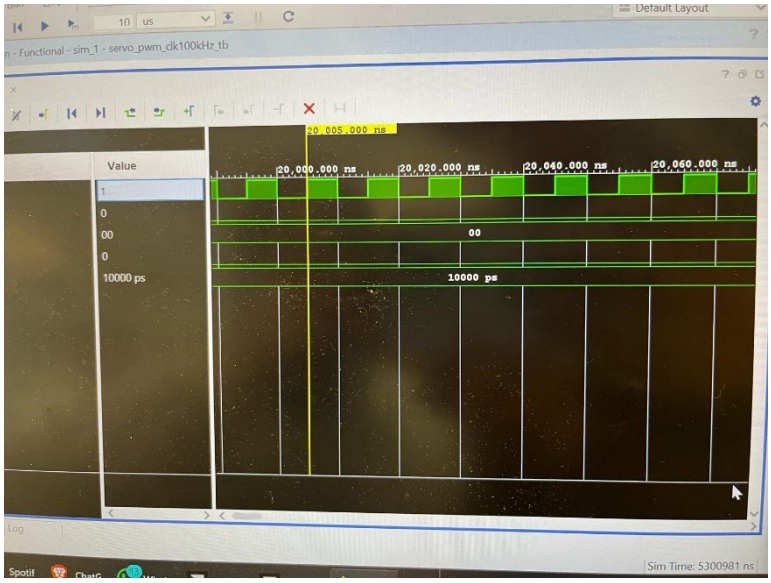

Figure-2 : Incorrect Simulation Result

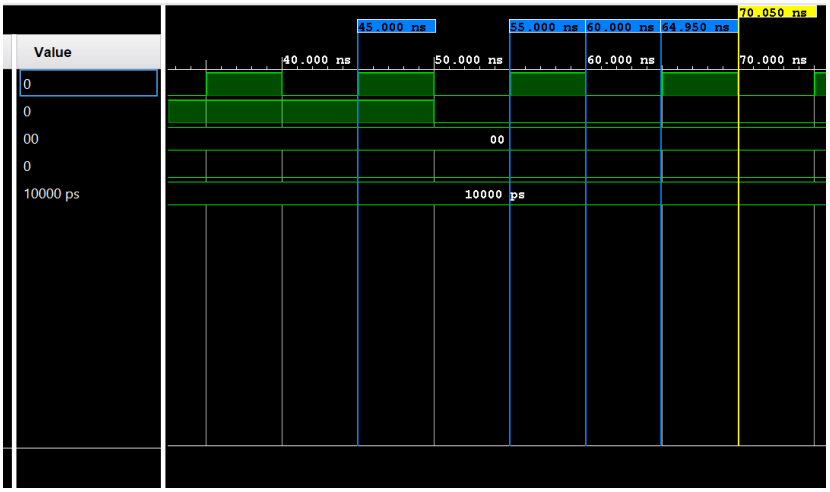

Figure-3: Incorrect Simulation Result-2

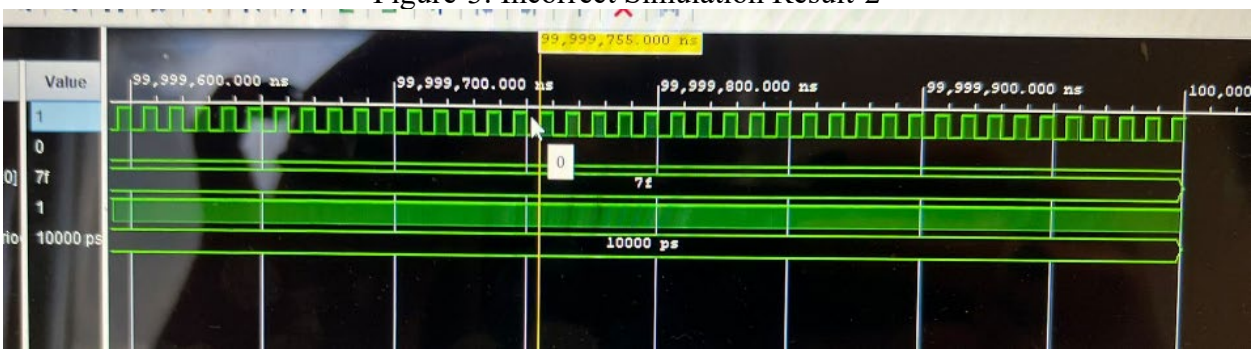

Figure-4: Correct Simulation Result

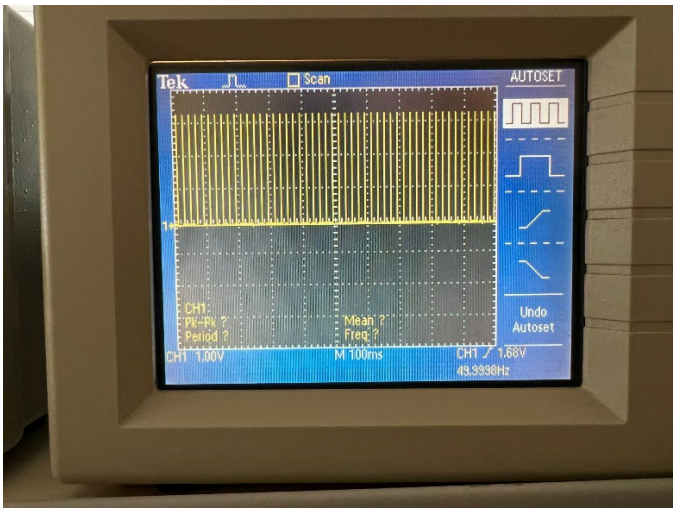

Figure-5: Oscilloscope waveform

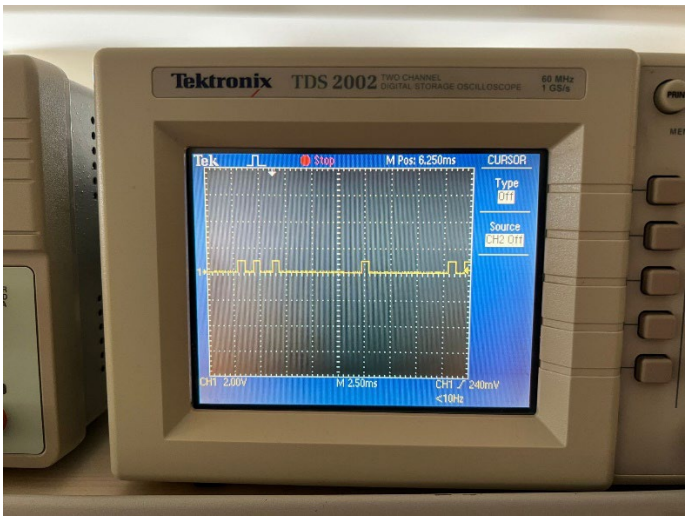

Figure-6: Oscilloscope waveform 2

Conclusion

In conclusion, on this lab, I gained knowledge about displaying PWM waveform on oscilloscope screen. Additionally, I get familiar with the concept of visualizing servo motors while working on this lab. I couldn't get the correct results initially, however, the results finally be as I desired. The reason why my code didn't work initially is that I couldn't assign the clock wizard module in correct place. I used the module but didn't add to my results, and didn't assign the ports on it. Also, I faced with some problems on my constraint file, while connecting the oscilloscope with fpga.

References

https://digilent.com/reference/programmable-logic/basys-3/reference-manual
https://vhdl.lapinoo.net/testbench/
https://github.com/Budea-Patrick/PWM-VHDL/tree/main/pwm.srcs/sources_1
https://stackoverflow.com/questions/29945327/square-waveform-generation

Appendices

Main.pwm

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.math_real.all;
ENTITY main.pwm is
   generic(
      num_bits : INTEGER := 12
   );
   port(
      CLK_IN  : IN STD_LOGIC;
      reset_n   : IN STD_LOGIC;
```

```
        EN     : IN STD_LOGIC;
        DUTY_CYCLE: IN STD_LOGIC_VECTOR(num_bits-1 downto 0);
        PWM_OUT: OUT STD_LOGIC
        );
END main.pwm;
ARCHITECTURE RTL OF main.pwm is
component clk_wiz_0
port
 (
  clk_out1          : out    std_logic;
  clk_in1           : in     std_logic
 );
end component;
signal counter: UNSIGNED(num_bits-1 downto 0);
begin
your_instance_name : clk_wiz_0
   port map (
   clk_out1 => PWM_OUT,
   clk_in1 => CLK_IN
 );
count_proc: process(CLK_IN,reset_n)
begin
    if(reset_n = '0') then
        counter <= (others => '0');

    elsif rising_edge(CLK_IN) then
        if (EN = '1') then
            counter <= counter + 1;
        else
            counter <= counter;
        end if;
    end if;
end process count_proc;
PWM_generator: process(CLK_IN,reset_n)
begin
    if(reset_n = '0') then
        PWM_OUT <= '0';
    elsif rising_edge(CLK_IN) then
        if (EN = '1') then
            if (counter < unsigned(DUTY_CYCLE)) then
                PWM_OUT <= '1';
            else
                PWM_OUT <= '0';
            end if;
        else
            PWM_out <= '0';
```

```
        end if;
    end if;
end process PWM_generator;
END RTL;
```

Tb_main.pwm

```
library ieee;
use ieee.std_logic_1164.all;
entity tb_ main.pwm is
end tb main.pwm;
architecture tb of tb_main.pwm is
    component main.pwm
        port (CLK_IN     : in std_logic;
              reset_n    : in std_logic;
              EN         : in std_logic;
              DUTY_CYCLE : in std_logic_vector (num_bits-1 downto 0);
              PWM_OUT    : out std_logic);
   end component;
    signal CLK_IN     : std_logic;
    signal reset_n    : std_logic;
    signal EN         : std_logic;
    signal DUTY_CYCLE : std_logic_vector (num_bits-1 downto 0);
    signal PWM_OUT    : std_logic;
    constant TbPeriod : time := 1000 ns;
    signal TbClock : std_logic := '0';
    signal TbSimEnded : std_logic := '0';
begin
    dut : main.pwm
    port map (CLK_IN     => CLK_IN,
              reset_n    => reset_n,
              EN         => EN,
              DUTY_CYCLE => DUTY_CYCLE,
              PWM_OUT    => PWM_OUT);
    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';
    CLK_IN <= TbClock;
    stimuli : process
    begin
        EN <= '0';
        DUTY_CYCLE <= (others => '0');
        reset_n <= '1';
        wait for 100 ns;
        reset_n <= '0';
        wait for 100 ns;
        wait for 100 * TbPeriod;
        TbSimEnded <= '1';
        wait;
```

```
    end process;
end tb;
configuration cfg_ tb_ main.pwm of tb_ main.pwm is
    for tb
    end for;
end cfg_ tb_ main.pwm;
```