AHMET FARUK ÇOLAK-22102104

1 Purpose

The purpose of this lab is to design a VHDL code that generates arbitrary digital waveforms. The waveforms can be chosen by the changed based on an input signal. The code will use the Clocking Wizard IP to generate a precise clock signal at the desired frequency.

2 Methodolgy

1. Design a VHDL code that generates the desired digital waveform. This code can include logic statements to create various wave shapes, such as square waves.
2. Use the Clocking Wizard IP to generate a clock signal with the appropriate frequency. The desired waveform frequency will determine the clock frequency needed.
3. Implement a testbench to simulate the code and verify that the waveform is generated as expected. Use markers in the simulation window to measure the time intervals and ensure they match the design specifications.
4. Simulate the code and capture the test bench results.
5. Synthesize the design and download it to BASYS3 board.
6. Connect the output signal to an oscilloscope and measure the actual waveform generated by the circuit. Compare the measured waveform to the simulated waveform to ensure they match.

3 Results

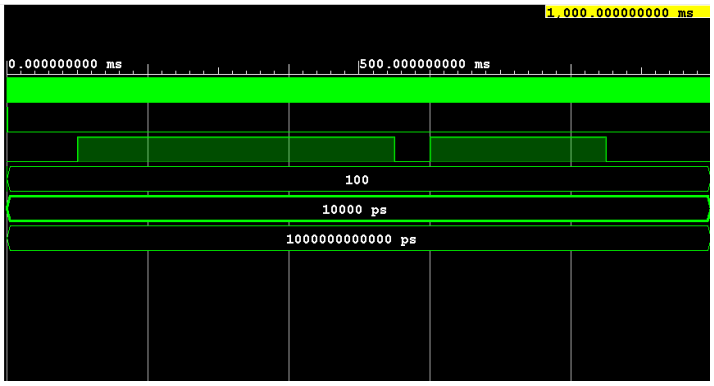

Figure-1: Test Bench Simulation Result

Unfortunately, I could not implement super-working test-bench code for 2 hours. Becasue of this, there is a tiny problem with my test bench.

Figure-2: Oscilloscope Result

This is the oscilloscope result. There is not problem problem with it. The only problem I encountered was on constriant file. Becasue of it I could not see any waveform on oscilloscope. Then one of my friend (Kayra, thanks to him) found a constraint file on github and I used that code for constraint file.

4 Conclusion

This lab demonstrates how to design and implement a VHDL code to generate arbitrary digital waveforms. The Clocking Wizard IP ensures an accurate clock signal for the desired waveform frequency. By using simulation and testing, I can verify that the generated waveform meets the design requirements.

Appendix

5 Main Function

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;


entity Arbitrary_Waveform_Generator is

    Port (

        clk : in STD_LOGIC; -- Clock input from Clocking Wizard IP

        reset : in STD_LOGIC; -- Reset input

        waveform_out : out STD_LOGIC -- Output waveform

    );

end Arbitrary_Waveform_Generator;


architecture Behavioral of Arbitrary_Waveform_Generator is


    -- Define waveform parameters

    type period_array is array (0 to 9) of integer range 1 to 1000000000; -- Array to hold different periods

    constant PERIODS : period_array := (10000000, 20000000, 30000000, 40000000, 50000000, 60000000, 70000000, 80000000, 90000000, 100000000); -- Periods in clock cycles

    constant HIGH_TIME : integer := 25000000; -- High time of the waveform (in clock cycles) - 25 ms for 100MHz clock

    signal counter : unsigned(31 downto 0); -- Counter to keep track of time

    signal period_index : integer range 0 to 9 := 0; -- Index to cycle through periods


begin


    process(clk, reset)

    begin

        if reset = '1' then

            counter <= (others => '0');
```

```
            waveform_out <= '0';
        elsif rising_edge(clk) then
            if counter = PERIODS(period_index) - 1 then -- When counter reaches the end of the current period
                counter <= (others => '0');
                waveform_out <= '1'; -- Set waveform high
            elsif counter = HIGH_TIME - 1 then -- When counter reaches the end of the high time
                waveform_out <= '0'; -- Set waveform low
                counter <= counter + 1; -- Increment counter
            else
                counter <= counter + 1; -- Increment counter
            end if;

            if counter = PERIODS(period_index) - 1 and period_index < 9 then -- Move to the next period
                period_index <= period_index + 1;
            elsif counter = PERIODS(period_index) - 1 and period_index = 9 then
                period_index <= 0; -- Wrap around to the first period
            end if;
        end if;
    end process;

end Behavioral;
```

6 Test Bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Arbitrary_Waveform_Generator_TB is
end Arbitrary_Waveform_Generator_TB;

architecture Behavioral of Arbitrary_Waveform_Generator_TB is
```

```
    -- Constants
    constant CLK_PERIOD : time := 10 ns; -- Clock period (100 MHz)
    constant SIM_DURATION : time := 1000 ms; -- Simulation duration

    -- Signals
    signal clk_tb : std_logic := '0';
    signal reset_tb : std_logic := '0';
    signal waveform_out_tb : std_logic;
    signal period_index_tb : integer range 0 to 9 := 0;

begin
    -- Instantiate the DUT (Design Under Test)
    dut: entity work.Arbitrary_Waveform_Generator
        port map (
            clk => clk_tb,
            reset => reset_tb,
            waveform_out => waveform_out_tb
        );

    -- Clock generation process
    clk_process: process
    begin
        while now < SIM_DURATION loop
            clk_tb <= not clk_tb; -- Toggle clock
            wait for CLK_PERIOD / 2;
        end loop;
        wait;
    end process;

    -- Reset process
    reset_process: process
```

```
    begin
        reset_tb <= '1';
        wait for CLK_PERIOD * 5; -- Hold reset for a brief period
        reset_tb <= '0';
        wait;
    end process;

    -- Monitor waveform process
    monitor_waveform: process
    begin
        wait until rising_edge(clk_tb); -- Wait for rising edge of clock
        wait for CLK_PERIOD * 5; -- Wait for stabilization

        for i in 1 to 100 loop
            wait until rising_edge(clk_tb);
            period_index_tb <= period_index_tb + 1;
            report "Waveform output at time " & integer'image(i * 10) & " ms: " &
std_logic'image(waveform_out_tb);
        end loop;

        wait;
    end process;

end Behavioral;
```

7 Constraint

```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports { clk }];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports { clk }];

## Reset button
set_property -dict { PACKAGE_PIN W19  IOSTANDARD LVCMOS33 } [get_ports { reset }];
```

```
## Output waveform
set_property -dict { PACKAGE_PIN A14  IOSTANDARD LVCMOS33 } [get_ports { waveform_out }];
```