Emre Akdemir

22302606

EE102-02

EE-102 LAB 6 REPORT: ARBITRARY WAVEFORM GENERATOR

09/04/2025

Purpose

The aim of this lab was to create a VHDL code that uses clocking wizard to generate an arbitrary waveform and displaying the waveform on the oscilloscope to test whether the simulation result and the waveform on the oscilloscope matches or not.

Methodology

The objective of this lab was to generate a waveform using clocking wizard property of the VHDL. First, a clocking wizard IP has been created and set as 100 MHz input and 10 MHz output. Then, I wrote the code for the top module. The clock input is set to the default clock of the BASYS3 board. The clocking wizard IP is instantiated and the input of the clocking wizard set to the BASYS3's internal clock. The output signal is used as the rising edge condition of the code. Initially, a selection input is used to generate arbitrary random waveforms. Although the output of the simulation was as intended, since the selection input was related to the clock output, the result at oscilloscope was corrupted due to mismatch of 100Mhz and 10Mhz clock frequencies. Thus, I made changes to generate an waveform that behaves in a regular manner.

Design Specifications

There are two inputs and two outputs in the top module. Inputs are clk_in and reset. Dout_o is the output of the design. In the architecture part, the clk_wiz_0 component is declared, and the clock of the wizard is connected to the internal clock input, which is clk_in1. For the signal part, two signals are used in the code. First one is clk_out_1, which is basically the output of the clock wizard and set to 10 MHz. Other one is clk_locked and it is the signal that shows whether the wizard produces a stable and reliable clock or not. My logic for the process part is the following: I preset a limit

value, and I declare an unsigned 13-bit variable called operation. At each rising_edge of the clk_out_1 signal, 1 is added to the operation variable. If the operation variable is less than the limit, then dout_o value is assigned to 1. If it is between limit and two times of the limit value, then dout_o is assigned to 0. When dout_o is bigger than 2*limit, operation is reset back to 0.

Results

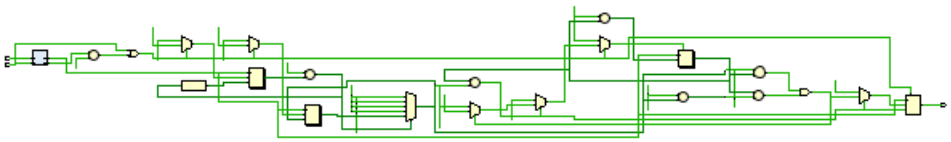

Figure 1: RTL Schematics for the Top Module

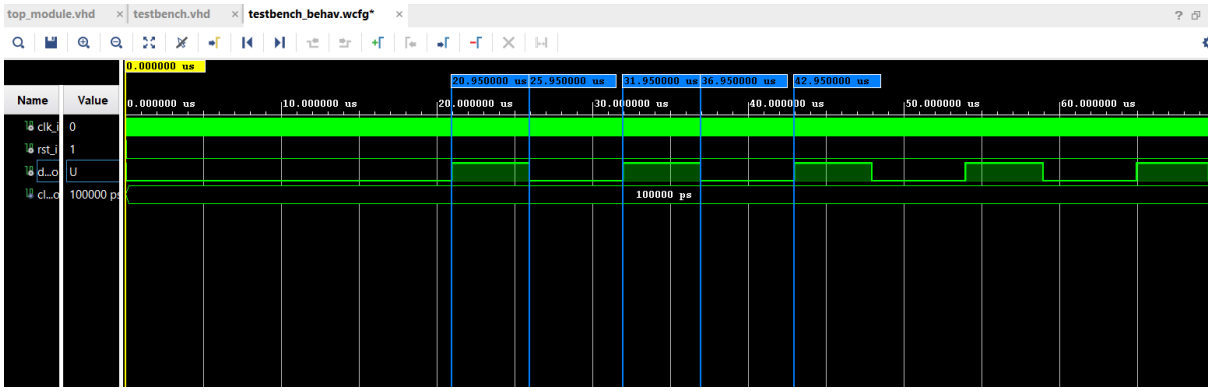

Figure 2: Simulation Results (5 microseconds High, 6 microseconds Low)

Figure 3: Waveform on Oscilloscope Screen (Most possible two reasons for the discrepancies of the waveform at edges is the noise or improper compensation of the probe.)

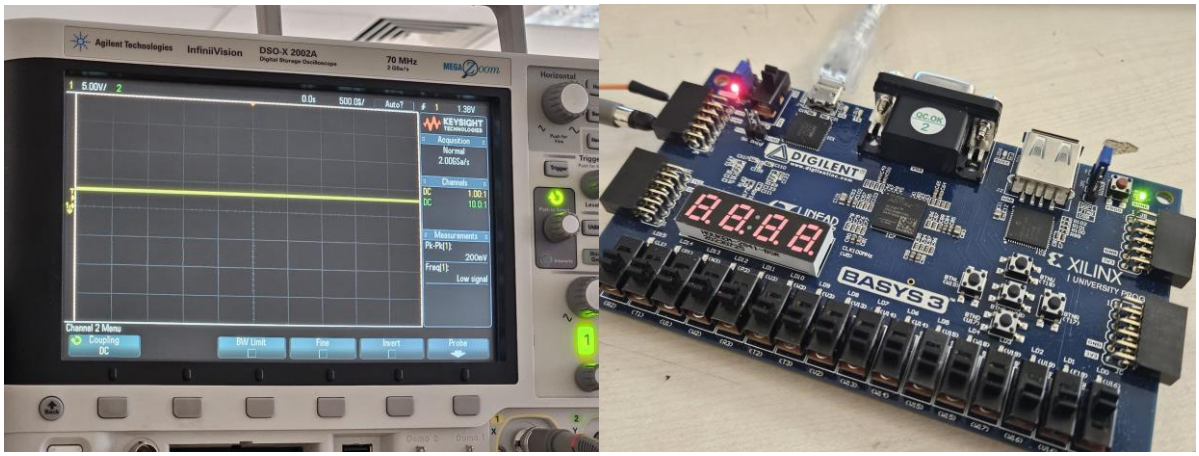

Figure 4: The signal is 0 when reset switch is activated.

Conclusion

The aim of this experiment was to create an arbitrary waveform generator using clock wizard property of the VHDL and displaying the result on oscilloscope by sending the bit data to pin of the BASYS3. Although I had problems about the selection input of the limit value in the code, I mostly apprehended the Xilinx IP's. As a result, the lab was successful and played a critical role in understanding of creating waveforms using clock as intended. I believe that it will be helpful in my term project in terms of testing PWM signals of the servo motors.

Code

top_module.vhd

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

entity clock_main is Port (clk_in : in std_logic; -- input clock (e.g., from FPGA pin) reset : in std_logic; dout_o : out std_logic -- Output clock signal); end clock_main;

architecture Behavioral of clock_main is

```
-- Component Declaration for the clocking wizard IP
component clk_wiz_0  -- IP generated by Vivado Clocking Wizard
    port (
        clk_in1  : in  std_logic;  -- Input clock
        clk_out1 : out std_logic;  -- Output clock
        reset    : in  std_logic;  -- Reset signal
        locked   : out std_logic   -- Locked signal indicating the
clock is stable
    );
end component;

-- Signals for internal clock outputs and the locked signal
signal clk_out_1  : std_logic;
signal clk_locked : std_logic;
```

begin -- Instantiating the clocking wizard IP clk_wiz_inst : clk_wiz_0 port map (clk_in1 => clk_in, -- Connect the input clock to the clk_in1 of the clocking wizard clk_out1 => clk_out_1, -- Connect the generated clock output to internal signal reset => reset, -- Connect the reset signal locked => clk_locked -- Connect the locked signal (indicating if the clock is stable));

```
-- Process to handle the output signal based on the input clock and
reset
process(clk_out_1)
variable operation : unsigned(12 downto 0) := (others => '0');
variable sel: unsigned(2 downto 0) := (others => '0');
variable limit: unsigned(7 downto 0) := (others => '0');
begin
    if rising_edge(clk_out_1) then
        if reset = '1' or clk_locked = '0' then
            dout_o <= '0';  -- Hold the output low if reset is high
or clock is not locked
            operation := (others => '0');
            report "Operation reset to 0";
        else
            sel := sel + 1;
            -- sel logic to increment limit values
            case sel is
                when "000" =>
                    limit := to_unsigned(5, 8);
                    sel := sel + 1;
                when "001" =>
                    limit := to_unsigned(5, 8);
                    sel := sel + 1;
                when "010" =>
                    limit := to_unsigned(5, 8);
                    sel := sel + 1;
                when "011" =>
                    limit := to_unsigned(5, 8);
                    sel := sel + 1;
                when others =>
                    sel := (others => '0');
            end case;

            -- Operation logic
            if (to_integer(operation) < to_integer(limit)) then
                operation := operation + 1;
                dout_o <= '1';  -- Otherwise, pass the clock output
to the dout_o signal
                report "Operation incremented: " &
integer'image(to_integer(operation));
            else
                if (to_integer(limit) <= to_integer(operation) and
```

```
to_integer(operation) < 2*(to_integer(limit))) then
                    operation := operation + 1;
                    dout_o <= '0';
                    report "Operation in the range limit to 2*limit:
" & integer'image(to_integer(operation));
                else
                    operation := (others => '0');
                    report "Operation reset to 0 after exceeding
2*limit: " & integer'image(to_integer(operation));
                end if;
            end if;
        end if;
    end if;
end process;
```

end Behavioral;

testbench.vhd

library IEEE; use IEEE.STD_LOGIC_1164.ALL;

entity testbench is end testbench;

architecture Behavioral of testbench is -- Signals for clock, reset, and output signal clk_i : std_logic := '0'; -- Input clock signal rst_i : std_logic := '1'; -- Reset signal signal dout_o : std_logic; -- Output clock

```
-- Clock generation process for 10 MHz (period = 100 ns)
constant clk_period : time := 100 ns;   -- Clock period is 100 ns
(10 MHz)
```

begin -- Instantiate the Unit Under Test (UUT) uut: entity work.clock_main port map (clk_in => clk_i, reset => rst_i, dout_o => dout_o);

```
-- Clock generation process: 100 MHz
clk_proc: process
begin
    while true loop
        clk_i <= '0';
        wait for clk_period / 2;
```

```
        clk_i <= '1';
        wait for clk_period / 2;
    end loop;
end process;

-- Reset release after 100 ns
reset_proc: process
begin
    rst_i <= '1';  -- Assert reset
    wait for 10 ns;  -- Hold reset for 100 ns
    rst_i <= '0';  -- Deassert reset
    wait;
end process;


end Behavioral;
```

basys3.xdc (constraint file)

set_property PACKAGE_PIN W5 [get_ports clk_in]

set_property IOSTANDARD LVCMOS33 [get_ports clk_in] create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk_in]

set_property PACKAGE_PIN V17 [get_ports {reset}]

set_property IOSTANDARD LVCMOS33 [get_ports {reset}]

set_property PACKAGE_PIN J1 [get_ports {dout_o}]

set_property IOSTANDARD LVCMOS33 [get_ports {dout_o}]