April 9th 2025

Alp Efe Kılıçarslan, 22402390

EEE 102-2

LAB-06: Arbitrary Waveform Generator

Purpose:

The purpose of this experiment is to create an arbitrary waveform generator in VHDL using Vivado and implement it on a BASYS 3 using the IP Catalogue modules. The generated waveform will be measured using an oscilloscope to analyze it's behavior.

Design Specifications:

In order to generate the waveform, a clock input was required. This was done by using the built-in IP Catalogue and adding the clocking wizard module onto the project. The instantiated clock was used in the waveform generator module with its default clock frequency, which is 100 MHz. A simple counter was used to keep logic high and logic low active for the desired amount of time. The generated waveform had a net period consisting of 50 µs low and 70 µs high. To get these values, two if statements were used. When the current state of the waveform was zero and the counter reached 5000 (which is the desired low-time divided by 10 ns, which is the main clock's smallest timeframe) the wave state would be set to one and if the current state was one and the counter reached 7000 (desired time divided by 10 ns) the state would be set to zero. If neither condition was met, the counter would be incremented by one for each rising edge of the clock. Finally, the clocking wizard and the waveform generator were combined in a main module and the output was sent to a PMOD port. The constraint file was edited to accommodate this port and utilize the clock as needed.

After the main part of the code was created, a testbench that utilized the main clock of the project was made. The waveform generator module was instantiated in the testbench and its output was set to the testbench's output.

Methodology:

Task-1) The testbench was set as the top of the simulation files. After this declaration, the simulation duration was set to 1000 µs and the testbench was ran. Both the low and high signals' durations were analyzed to see if they had no delay. The resultant graph was recorded.

Task-2) The written code was implemented onto the FPGA and ran. An oscilloscope probe was connected to the port that was defined in the constraint file. The waveform that was displayed on the oscilloscope was compared with the testbench simulation's waveform.

Results:

Task-1) The testbench was ran and expanded (Figure 1.1). After it was generated, each pulse was analyzed in conjunction with their durations. The low parts were found to be exactly 50 µs long and the high parts were found to be 70 µs long. This matched our ideal results. Markers were placed along the rising and falling edges of the waveform to have a detailed view of multiple points along the waveform. The small points in time mathematically matched the desired output of 50 µs low and 70 µs high (Figure 1.2).

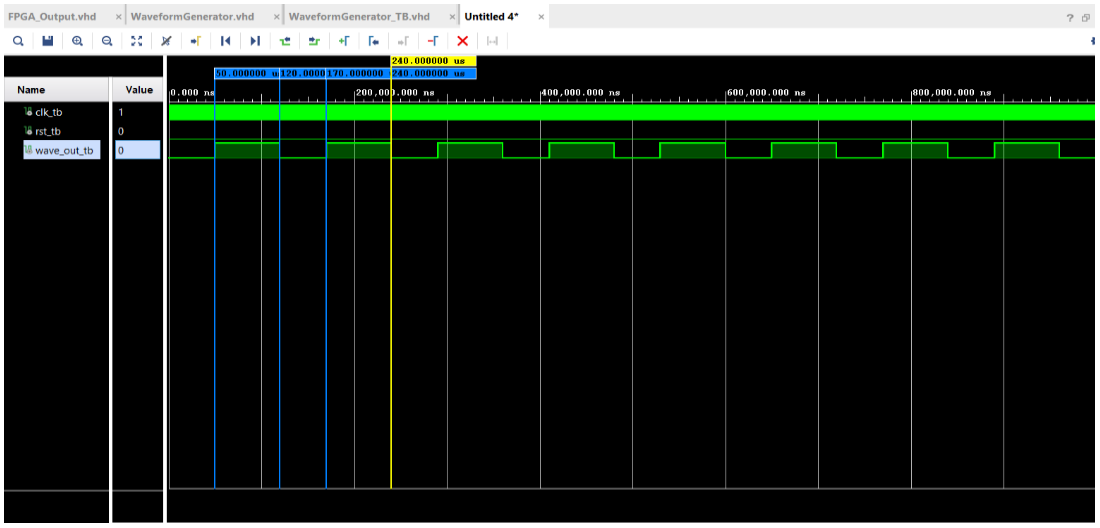

Figure 1.1: Full view of the simulation

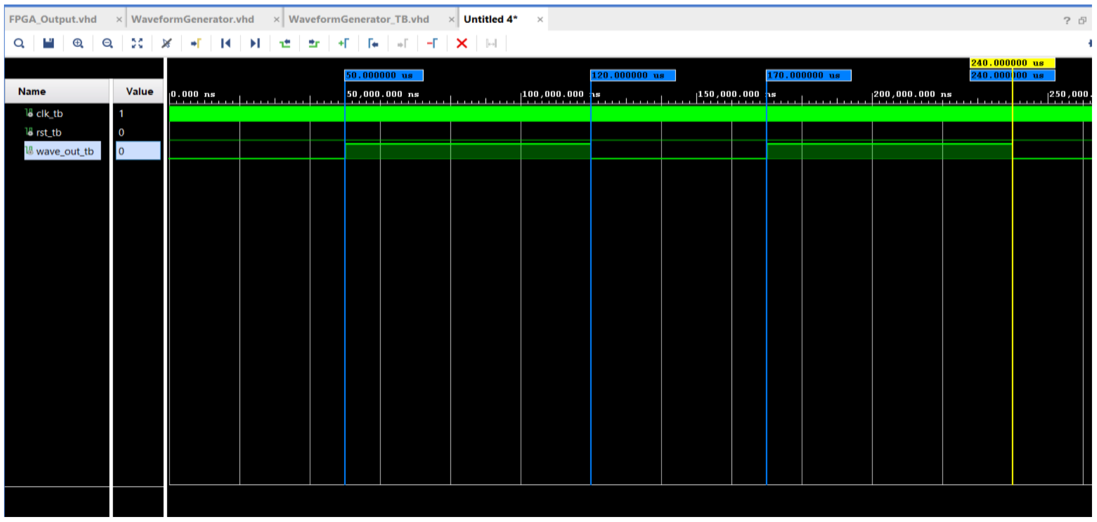

Figure 1.2: Close-up of the simulation showing precise timings of the waveform

Task-2) The bitstream of the project was generated and ran on the BASYS 3. An oscilloscope probe was connected to an oscilloscope and grounded. The tip of the probe was connected to the designated PMOD port of the BASYS 3. After running the code on the BASYS 3, a rectangular wave was observed on the digital screen of the oscilloscope. Cursors were placed on the rising and falling edges on the waveform and the timings were analyzed (Figure 2.1-2.2).

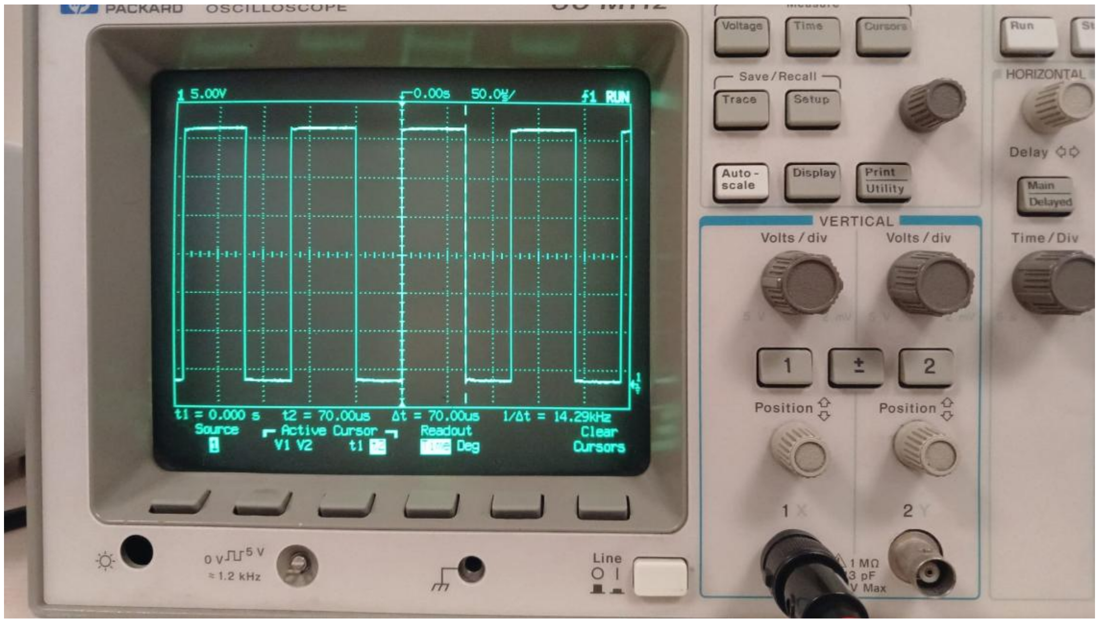

Figure 2.1: Cursors set at 0 and 70µs, showing the high signal is active for 70µs

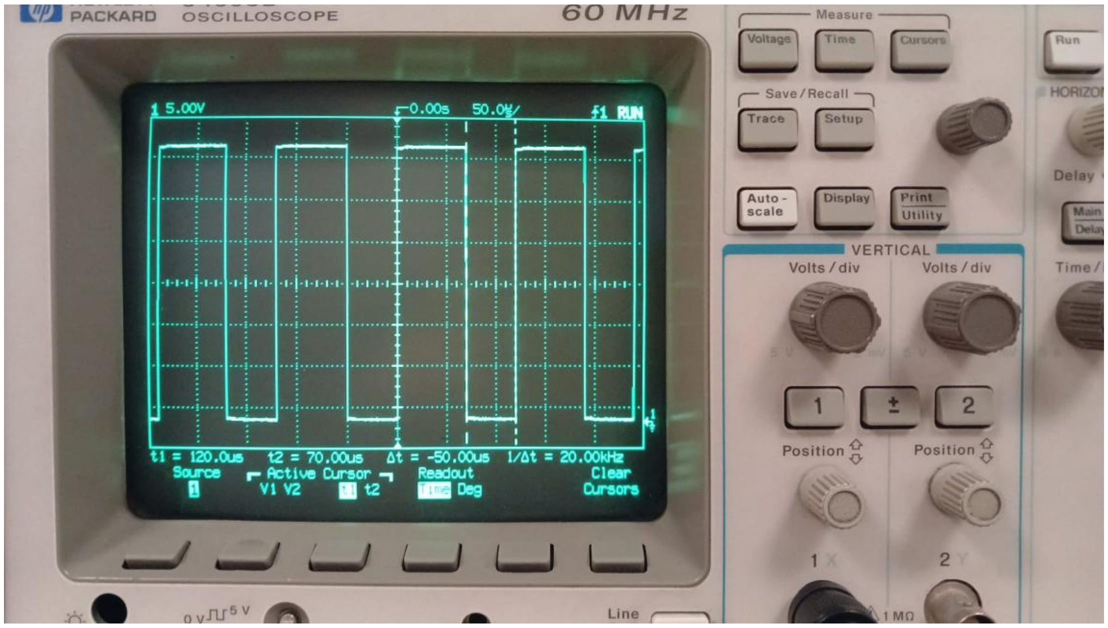

Figure 2.1: Cursors set at 70µs and 120µs, showing the low signal is active for 50µs

After comparing the wave generated by the testbench and the waveform seen on the oscilloscope, near no differences were found. This indicated that both the testbench and the actual design functioned as desired.

Conclusion:

The aim of this experiment was to utilize the IP Catalog built into Vivado to generate an arbitrary waveform. Firstly, the main part of the code was written to take advantage of the clocking wizard. Then, a testbench was written to get the ideal form of our wave. After that, the code was run on the BASYS 3 and the resultant waveform was measured using an oscilloscope. The generated waveform and the ideal waveform were found to be near identical. Through these processes, utilization of the IP Catalog of Vivado was understood.

Appendices:

FPGA_Output.vhd

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

entity FPGA_Output is
    Port (
        clk_fpga : in STD_LOGIC;  -- 100MHz Basys3 onboard clock
        rst_fpga : in STD_LOGIC;  -- Reset button input
        wave_pin : out STD_LOGIC  -- Output waveform signal
    );
end FPGA_Output;

architecture Structural of FPGA_Output is
    signal wave_signal : STD_LOGIC;
    signal clk_internal : STD_LOGIC;
    signal locked : STD_LOGIC; -- Indicates stable clock

    component clk_wiz_0
        port (
            clk_in1  : in STD_LOGIC;
            clk_out1 : out STD_LOGIC;
            reset    : in STD_LOGIC;
            locked   : out STD_LOGIC
        );
    end component;

    component WaveformGenerator
        Port (
            clk : in STD_LOGIC;
            rst : in STD_LOGIC;
            wave_out : out STD_LOGIC
        );
    end component;

begin
    clk_wiz_inst : clk_wiz_0
        port map (
            clk_in1  => clk_fpga,   -- External 100MHz clock
            clk_out1 => clk_internal,  -- Stable internal clock
            reset    => rst_fpga,
            locked   => locked
        );

    WaveGen: WaveformGenerator
        port map (
            clk => clk_internal,
            rst => rst_fpga,
            wave_out => wave_signal
        );
    -- Drive output pin
    wave_pin <= wave_signal;
end Structural;
```

WaveGenerator.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity WaveformGenerator is
    Port (
        clk : in STD_LOGIC;
        rst : in STD_LOGIC;   -- Reset signal
        wave_out : out STD_LOGIC  -- Output waveform
    );
end WaveformGenerator;

architecture Behavioral of WaveformGenerator is
    signal counter : INTEGER := 0;
    signal state : STD_LOGIC := '0';
    constant HIGH_TIME : INTEGER := 7000; -- 70us at 100MHz
    constant LOW_TIME : INTEGER := 5000;  -- 50us at 100MHz
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if rst = '1' then
                counter <= 0;
                state <= '0';
            else
                if (state = '0' and counter = LOW_TIME - 1) then
                    state <= '1';
                    counter <= 0;
                elsif (state = '1' and counter = HIGH_TIME - 1) then
                    state <= '0';
                    counter <= 0;
                else
                    counter <= counter + 1;
                end if;
            end if;
        end if;
    end process;

    wave_out <= state;
end Behavioral;
```

WaveGenerator_TB.vhd

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

entity WaveformGenerator_TB is
end WaveformGenerator_TB;

architecture Testbench of WaveformGenerator_TB is
    signal clk_tb : STD_LOGIC := '0';
    signal rst_tb : STD_LOGIC := '1';
    signal wave_out_tb : STD_LOGIC;
    -- constant CLK_PERIOD : time := 10 ns; -- 100MHz clock

begin
```

```
    DUT: entity work.WaveformGenerator
    port map (
        clk => clk_tb,
        rst => rst_tb,
        wave_out => wave_out_tb
    );

    process
    begin
    wait for 5 ns;
        while true loop
            clk_tb <= '0';
            wait for 5 ns; --CLK_PERIOD/2;
            clk_tb <= '1';
            wait for 5 ns; --CLK_PERIOD/2;
        end loop;
    end process;

    process
    begin
        rst_tb <= '0';
        wait for 600 ms;
        rst_tb <= '1';
        wait for 120 us;
        rst_tb <= '0';
        wait;
    end process;
end Testbench;
```

wave_constraints.xdc

```
set_property PACKAGE_PIN W5 [get_ports clk_fpga]
set_property IOSTANDARD LVCMOS33 [get_ports clk_fpga]
create_clock -period 10.0 -name sys_clk -waveform {0 5} [get_ports clk_fpga]

#PMOD Port Output
set_property PACKAGE_PIN J2 [get_ports wave_pin]
set_property IOSTANDARD LVCMOS33 [get_ports wave_pin]

#LEDs for testing
#set_property PACKAGE_PIN U16 [get_ports wave_pin]
#set_property IOSTANDARD LVCMOS33 [get_ports wave_pin]

#Reset switch because why not
set_property PACKAGE_PIN V17 [get_ports rst_fpga]
set_property IOSTANDARD LVCMOS33 [get_ports rst_fpga]
```