Ada Zağyapan

EEE102-02

9 April 2025

Lab-6: Arbitrary Waveform Generator

PURPOSE

The purpose of this experiment is to design, implement, and simulate an arbitrary waveform generator on Basys3 that integrates clock management and waveform generation using VHDL.

METHODOLOGY

Initially, the Clocking Wizard IP (clk_wiz_0) was instantiated to convert the board's 100 MHz clock into a stable, clean clock signal which is again 100MHz, which was then used to drive a custom waveform generator module that produces an arbitrary digital waveform with defined timing intervals and an asynchronous reset for reliable operation. In this case, the waves were on the pattern of 80µs high - 50µs low - 70µs high -50 µs low. A testbench was then developed to simulate the design and verify the accuracy of the generated waveform, after which all modules were integrated into a top-level design, mapped to the appropriate FPGA pins using a constraint file, synthesized, programmed onto the BASYS3 board, and validated with oscilloscope measurements.

DESIGN SPECIFICATIONS

clk_wiz_0: The Clocking Wizard module receives the board's 100 MHz input clock and an asynchronous reset signal and generates a stable, cleaned clock output along with a locked indicator that confirms when the clock is stable and ready for use. This stable clock signal is

essential for the subsequent modules to operate synchronously and with precise timing characteristics. In this case, a 100MHz to 100MHz clock is used. Hence, the frequency of Basys3's clock remains unchanged.

waveform_generator.vhd: The waveform generator module utilizes the stable clock and reset signals to produce an arbitrary digital waveform, employing counter-based and state machine logic to create custom pulse patterns with defined high and low intervals (80-50-70-50µs). The design outputs the resulting signal on the wave_out pin which is observed later via the oscillator (Figure 1).

top_module.vhd: The top-level module integrates the clk_wiz_0 and waveform_generator modules by routing the board's 100 MHz clock and reset signals to the Clocking Wizard, and then feeding the generated stable clock into the waveform generator. This module drives the wave_out signal which is then mapped to Basys's PMOD port in order to be observed by the oscilloscope (Figure 2).

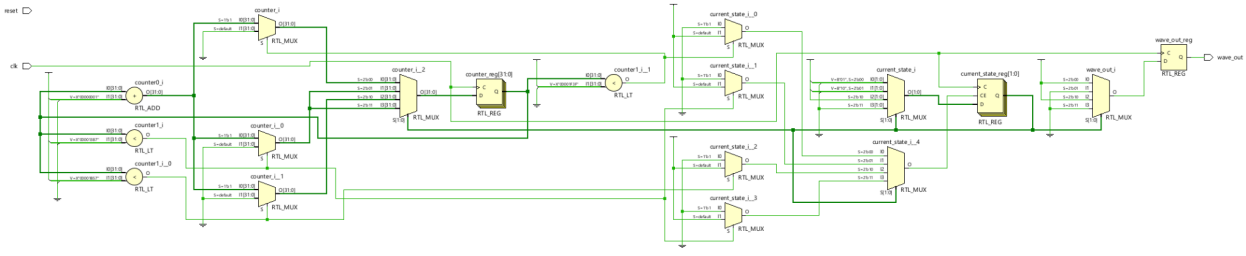

Figure 1: RTL schematic of *waveform.vhd*

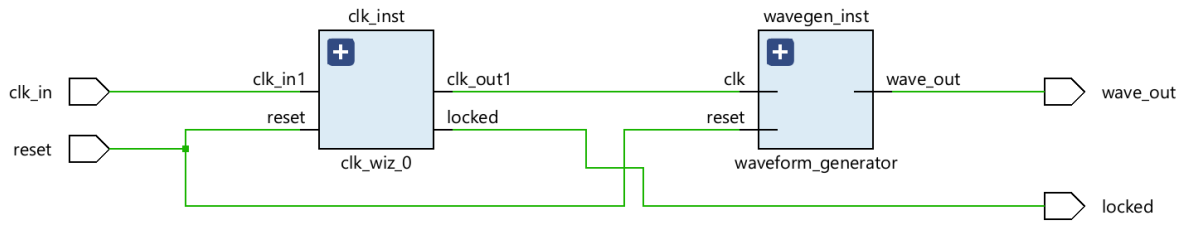

Figure 2: RTL schematic of *top_module.vhd*

RESULTS

The testbench simulation results confirmed that the Clocking Wizard produced a stable clock signal, which enabled the waveform generator to output an arbitrary digital waveform with precisely defined high and low intervals. In the simulation waveforms, the state machine and counter logic accurately generated the intended pulse sequence (Figure. There were no 10ns errors in the simulation either. On the BASYS3 board, the oscilloscope measurements reflected the same behavior observed in simulation. The output waveform exhibited the intended pulse widths and periodicity.

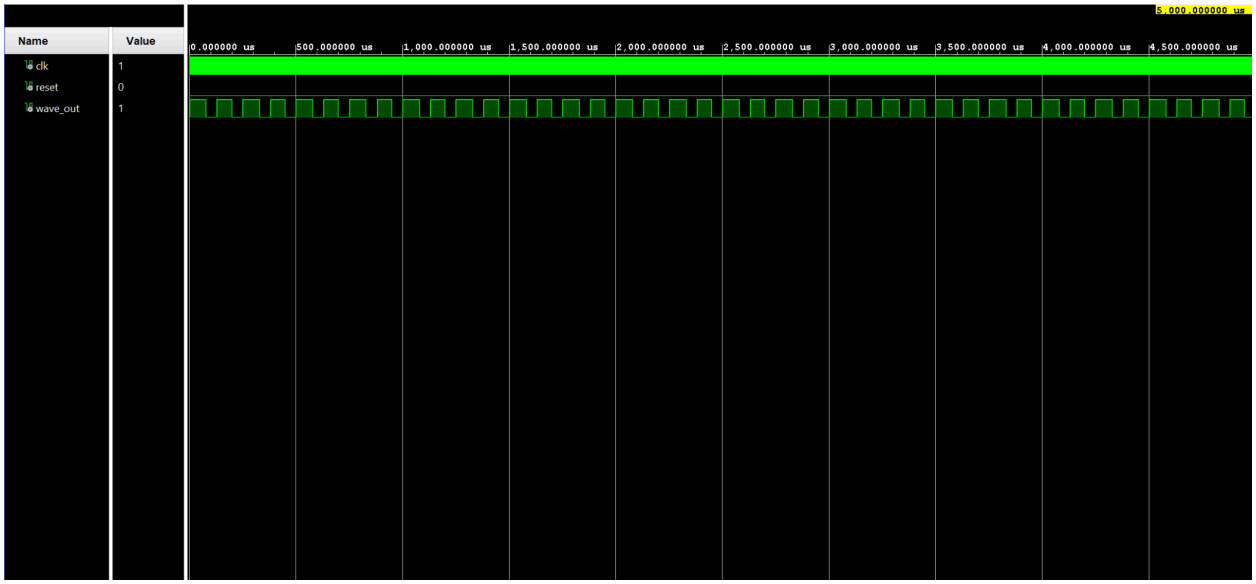

Figure 3: Simulated waveform of *top_module.vhd*

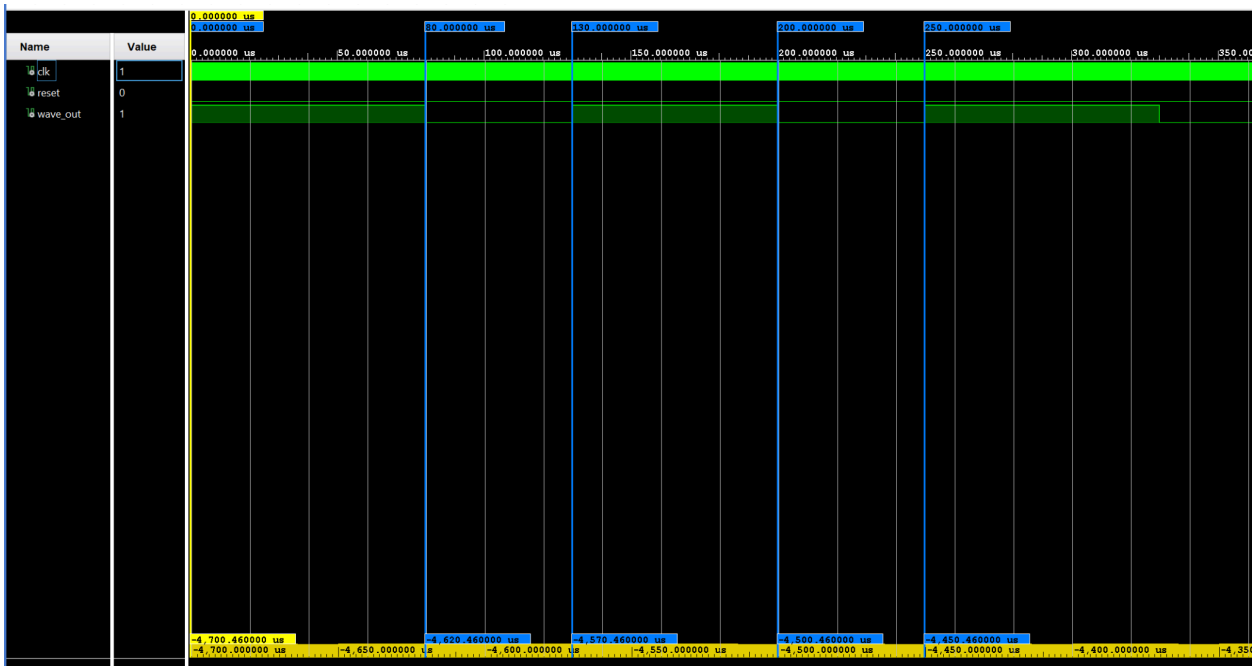

Figure 4: Close-up view of the simulation and timestamps

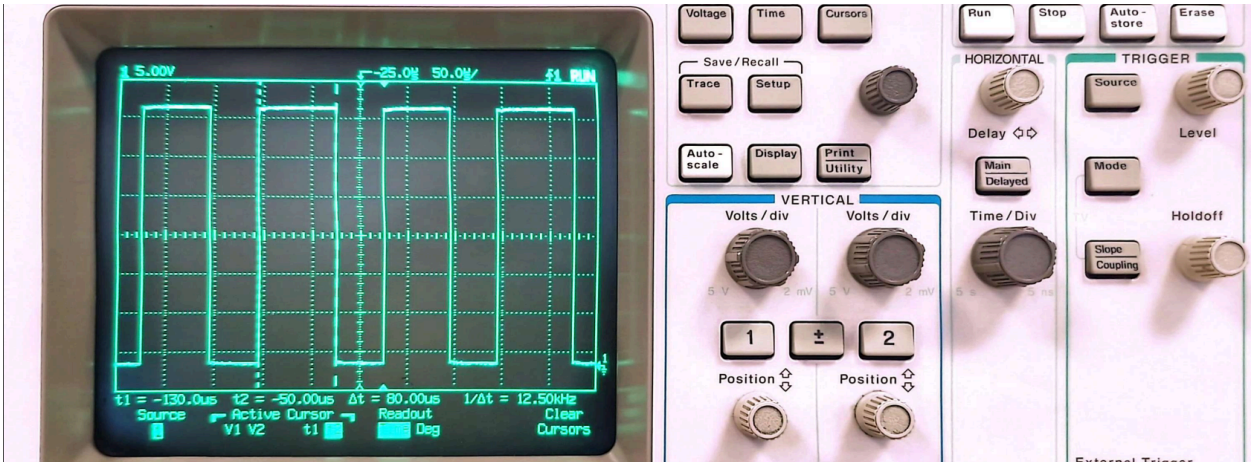

Figure 5: View of phase 1 (80µs high)

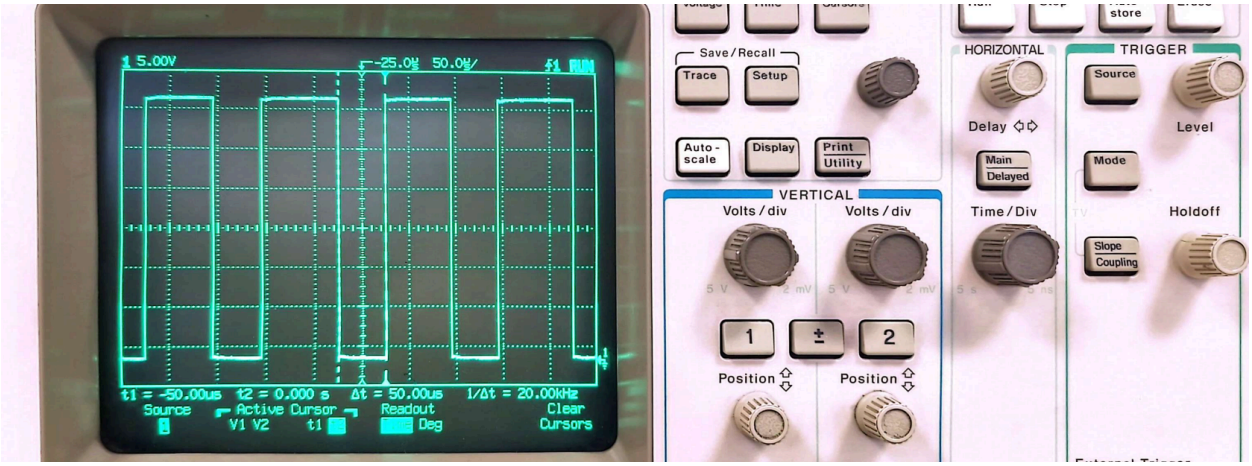

Figure 6: View of phase 2 (50µs low)

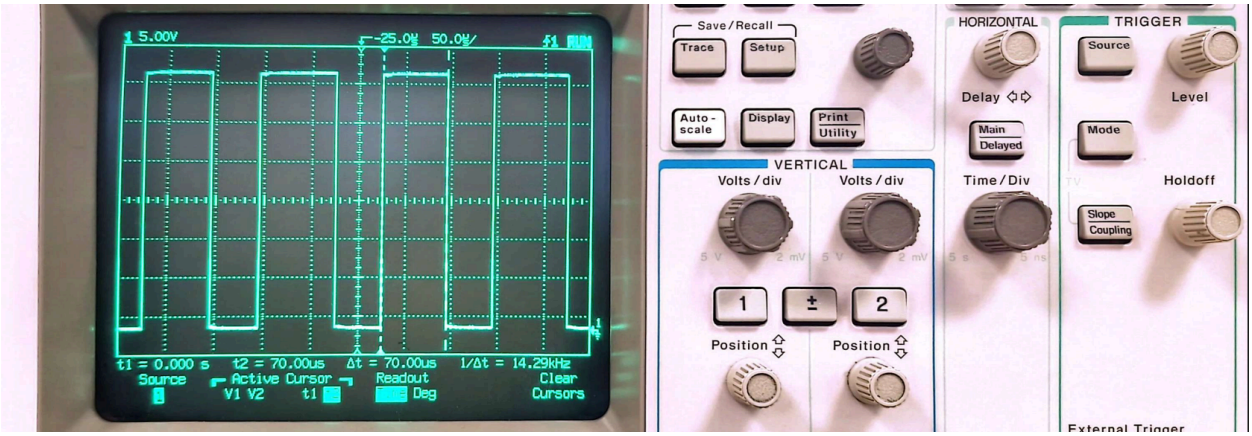

Figure 7: View of phase 3 (70µs high)

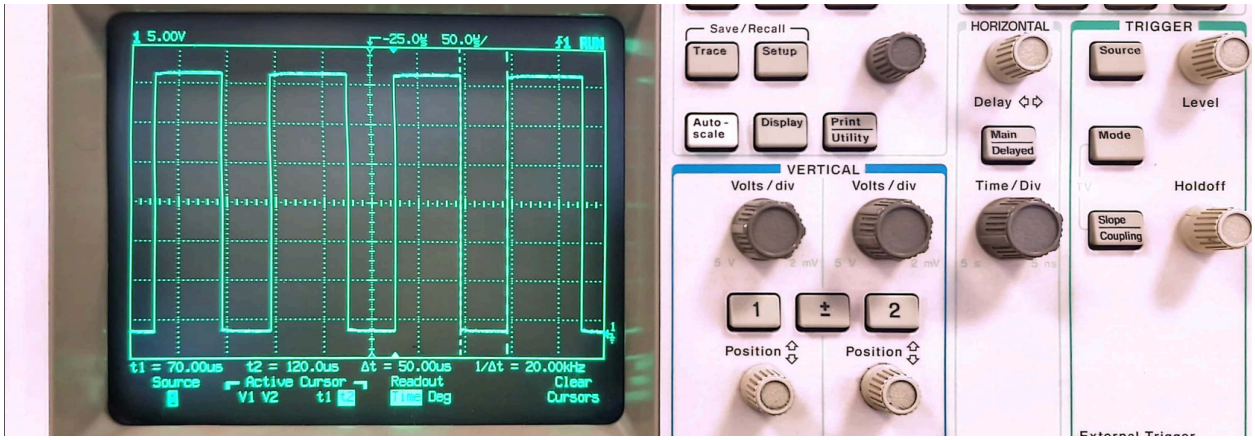

Figure 8: View of phase 4 (50μs low)

CONCLUSION

In conclusion, the goal of this experiment was to implement an arbitrary waveform generator using VHDL on the BASYS3 board. The process began with an understanding of clock management via the Clocking Wizard IP and the principles of digital waveform synthesis. After developing and simulating VHDL modules for clock stabilization using the wizard and the waveform generation, these components were integrated into a top-level design. The final implementation on the Basys3 functioned as expected. Both simulation and oscilloscope measurements confirmed that the generated waveform exhibited the intended timing intervals and followed the pulse patterns.

APPENDIX

waveform_generator.vhd

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

entity waveform_generator is
    Port (
        clk      : in  STD_LOGIC;
        reset    : in  STD_LOGIC;
        wave_out : out STD_LOGIC
    );
end waveform_generator;

architecture Behavioral of waveform_generator is

    type state_type is (phase1, phase2, phase3, phase4);
    signal current_state : state_type := phase1;
    signal counter       : integer := 0;

    constant phase1_cycles : integer := 8000;
    constant phase2_cycles : integer := 5000;
    constant phase3_cycles : integer := 7000;

begin

    process(clk, reset)
    begin
        if rising_edge(clk) then
            case current_state is
                when phase1 =>
                    wave_out <= '1';
                    if counter < phase1_cycles - 1 then
                        counter <= counter + 1;
                    else
                        counter <= 0;
                        current_state <= phase2;
```

```
                end if;
            when phase2 =>
                wave_out <= '0';
                if counter < phase2_cycles - 1 then
                    counter <= counter + 1;
                else
                    counter <= 0;
                    current_state <= phase3;
                end if;
            when phase3 =>
                wave_out <= '1';
                if counter < phase3_cycles - 1 then
                    counter <= counter + 1;
                else
                    counter <= 0;
                    current_state <= phase4;
                end if;
            when phase4 =>
                wave_out <= '0';
                if counter < phase2_cycles - 1 then
                    counter <= counter + 1;
                else
                    counter <= 0;
                    current_state <= phase1;
                end if;
        end case;
    end if;
end process;

end Behavioral;
```

top_module.vhd

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

entity top_module is
    Port (
        clk_in  : in  STD_LOGIC;  -- Input clock from the board
```

```
        reset   : in  STD_LOGIC;  -- Asynchronous reset
        wave_out: out STD_LOGIC;  -- Output waveform (assigned to U16)
        locked  : out STD_LOGIC   -- Clocking Wizard status
    );
end top_module;

architecture Behavioral of top_module is

    -- Clocking Wizard component
    component clk_wiz_0 is
        Port (
            clk_in1  : in  STD_LOGIC;
            clk_out1 : out STD_LOGIC;
            reset    : in  STD_LOGIC;
            locked   : out STD_LOGIC
        );
    end component;

    -- Waveform Generator component
    component waveform_generator is
        Port (
            clk      : in  STD_LOGIC;
            reset    : in  STD_LOGIC;
            wave_out : out STD_LOGIC
        );
    end component;

    signal clk_out_internal : STD_LOGIC;

begin

    clk_inst : clk_wiz_0
      port map (
        clk_in1  => clk_in,
        clk_out1 => clk_out_internal,
        reset    => reset,
        locked   => locked
      );
```

```
    wavegen_inst: waveform_generator
      port map (
        clk      => clk_out_internal,
        reset    => reset,
        wave_out => wave_out
      );

end Behavioral;
```