Alp Tekin

22403597

EE102-02

26.11.2025

Bilkent University

Department of Electrical and Electronics Engineering

Lab 6 Report: Arbitrary Waveform Generator

Purpose

The purpose of this experiment is to design an arbitrary digital waveform generator in VHDL. The waveform is produced using a custom clock generated by the Clocking Wizard IP.

Design Specifications

The design was implemented using a VHDL file that contains all waveform generation logic. Inside the architecture, a Clocking Wizard IP (clk_wiz_3) is used to generate a clean 50 MHz clock signal (s_clk_50MHz) from the 100 MHz input. locked signal from the IP is synchronized into the same clock domain to avoid clock domain crossing issues. The VHDL file "Arbitrary_Waveform.vhd" has two inputs and one output. clk_in is the 100MHz clock of the Basys3 board, reset_in is a reset signal which is driven by the center button and wave_out is the digital output waveform sent to an external PMOD pin. The waveform generation logic is implemented using a single 50MHz clock signal which is s_clk_50MHz. I created a finite-state machine with four states (80 µs high, 50 µs low, 70 µs high, 50 µs low) and the internal counter is used to control the output timing. The counter compares its value against predefined constants (C_80US, C_50US, C_70US) that are calculated based on the 50 MHz clock period which is 20 ns. The wave_out follows the required 80 µs- 50 µs- 70 µs- 50 µs sequence repeatedly. All logic operations in a single synchronous process and it ensures stable timing and clean periodic waveform.

Methodology

Unlike other labs where we used modular designs, I implemented this lab as a single VHDL file that contains the entire waveform generation logic. The design uses Clocking Wizard IP to produce a clean 50MHz clock signal from the Basys3. Clocking Wizard IP allows us to produce a clock signal from 100 MHz internal clock rather than defining the clock divider inside VHDL, which ensures a stable and precise timing source. After generating the new clock, the locked signal from the IP is synchronized into the same clock domain to avoid any problems. I first define the inputs: clk_in receives the original 100 MHz clock from Basys3, while reset_in is used to reset the whole system. Output wave_out signal is routed to an external pin to be observed on the oscilloscope. To create the required waveform, I defined constant values corresponding to the timing intervals (80 µs, 50 µs, 70 µs) which were calculated based on the 50 MHz (20 ns) clock period.

- 80 µs logic High wave was calculated using 80.000 ns / 20 ns = 4.000 clock cycles
- 50 µs logic Low wave was calculated using 50.000 ns / 20 ns = 2.500 clock cycles
- 70 µs logic High wave was calculated using 70.000 ns / 20 ns = 3.500 clock cycles

A single synchronous process was used and all timing logic was placed inside this process. This single clock domain ensures that each part of the waveform generated accurately and cleanly. After writing the VHDL code, I created the constraint file and assigned the waveform output to one of the PMOD pins. I also specified the internal 100 MHz clock and mapped it to clk_in input. Finally, I prepared a testbench to verify that the generated waveform is behave as expected pattern and timing. I compared these results with the measurements obtained using the oscilloscope.

Results

Since the entire design was implemented using a single VHDL file, the whole structure of the system can be clearly observed in the RTL schematic, as shown in Figure 1.

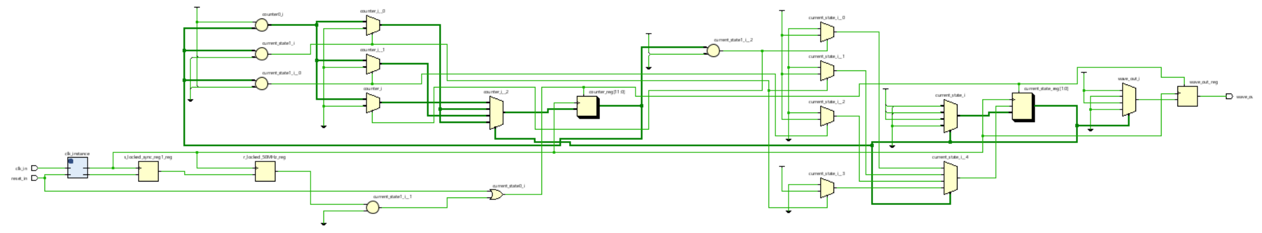
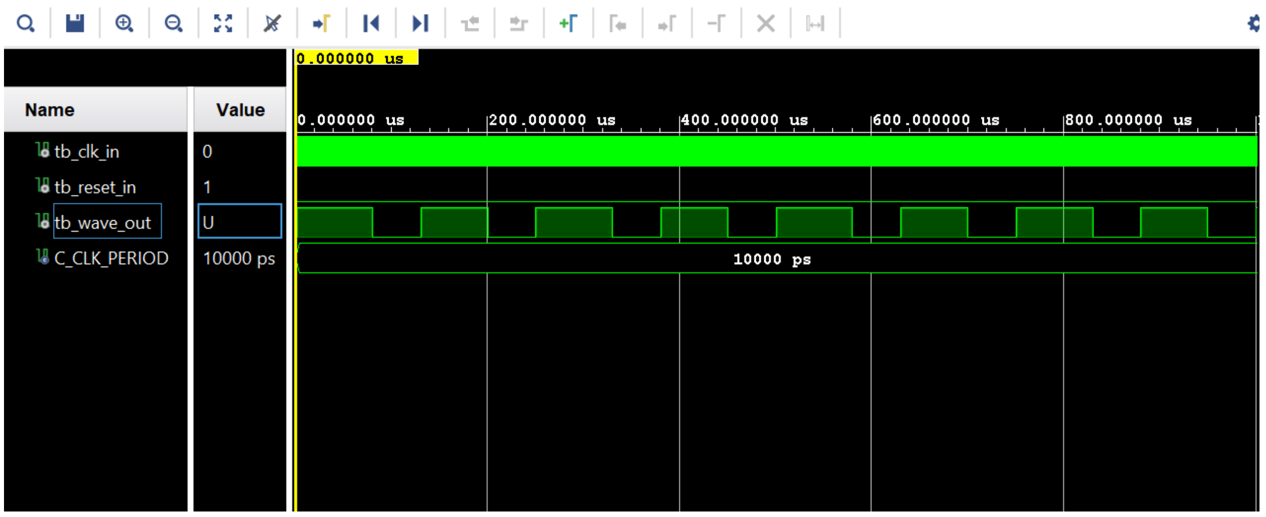


Figure 1: RTL Schematic of Waveform Design

Before implementing the design to Basys3 board, I first needed to ensure that the clock signal and the overall the system were operating correctly. Thus, I wrote and run a testbench to simulate the waveform and verify that the output followed the intended pattern and timing values. The general behavior of the clock and the initial waveform appeared correct in the simulation. The simulation result can be seen in Figure 2.



Name	Value
tb_clk_in	0
tb_reset_in	1
tb_wave_out	U
C_CLK_PERIOD	10000 ps

Figure 2: Simulation Result of the Waveform Design

To ensure that each interval of the waveform followed same pattern and there were no clocking issues or delays, I examined the simulation in detail. Figure 3 show the time interval of each pulse that corresponds to the arbitrary waveform. The timestamps reveal that the differences between rising and falling edges match the expected durations of 80 µs, 50 µs and 70 µs. The blue markers clearly indicate that each segment of the waveform aligns with the time constant defined in the design. This confirms that the finite-state machine and counter logic were functioning correctly in simulation.

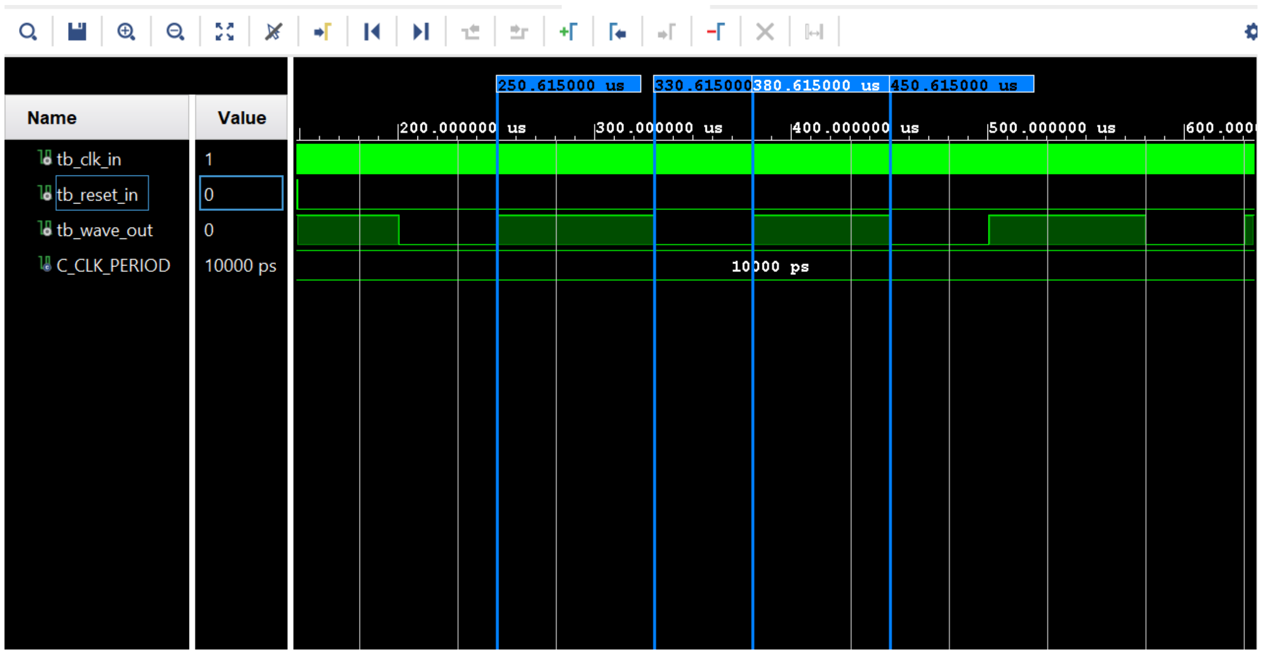


Figure 3: Simulation Result of the Waveform including Timestamps

After that, I implemented the design on the Basys3 board and made the proper connections to observe the signal behavior on the oscilloscope. Figure 5-8 shows the behavior of the waveform after implementation.

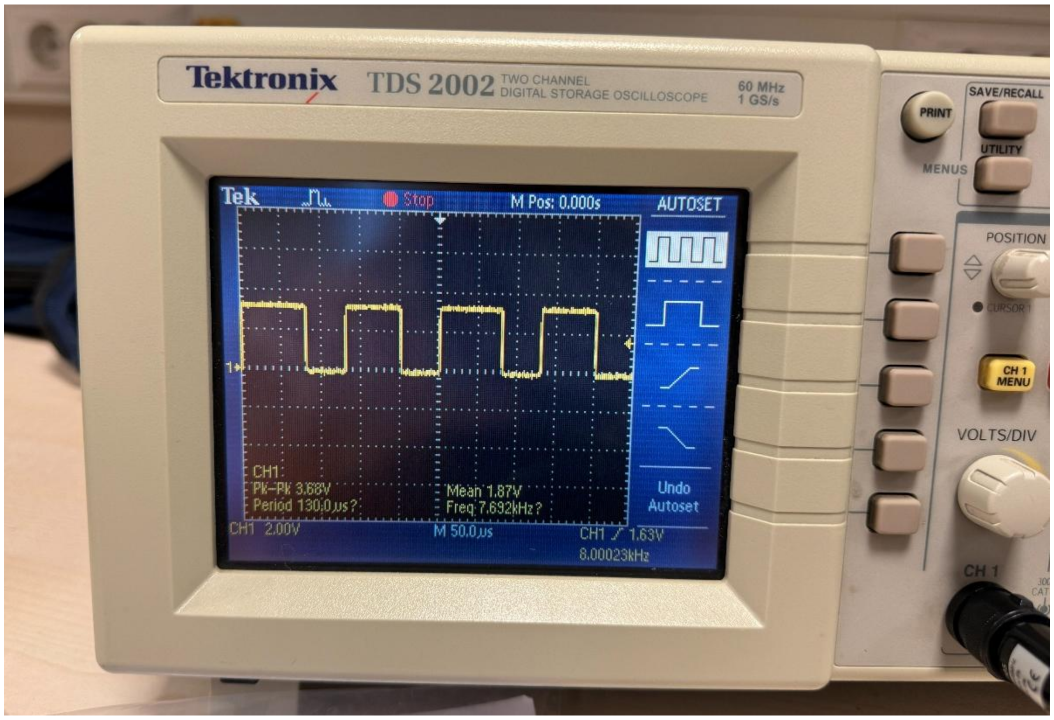


Figure 5: General Behavior of Arbitrary Waveform on Oscilloscope

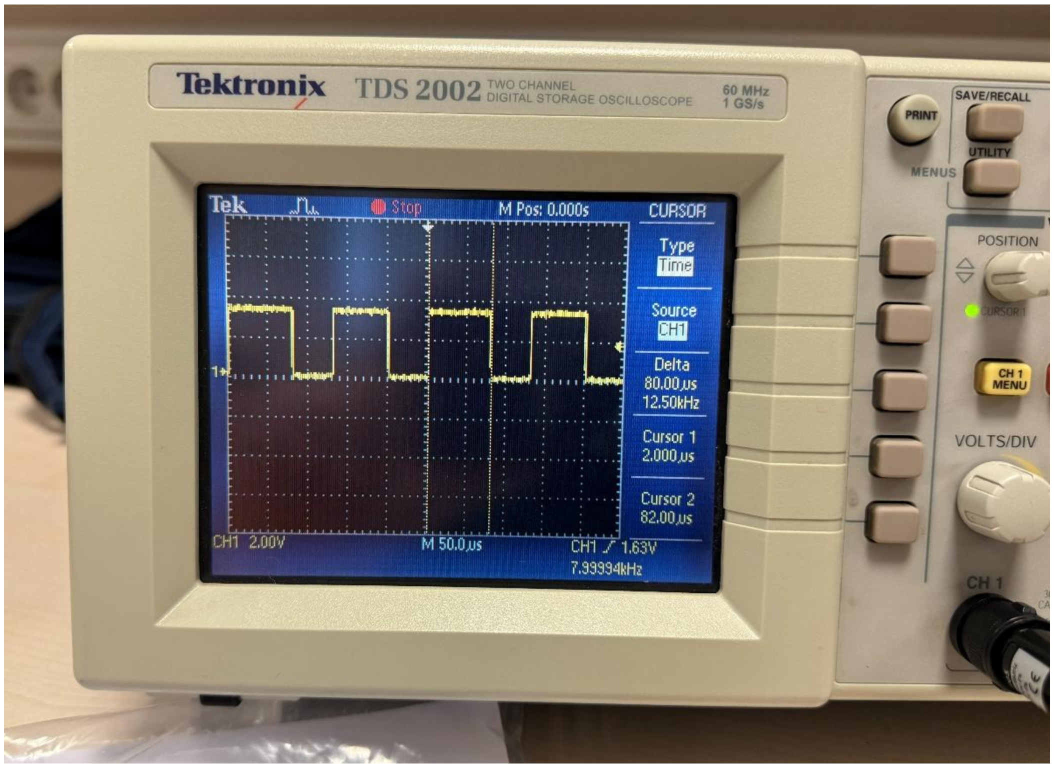


Figure 6: 80 μs Logic High Wave on Oscilloscope

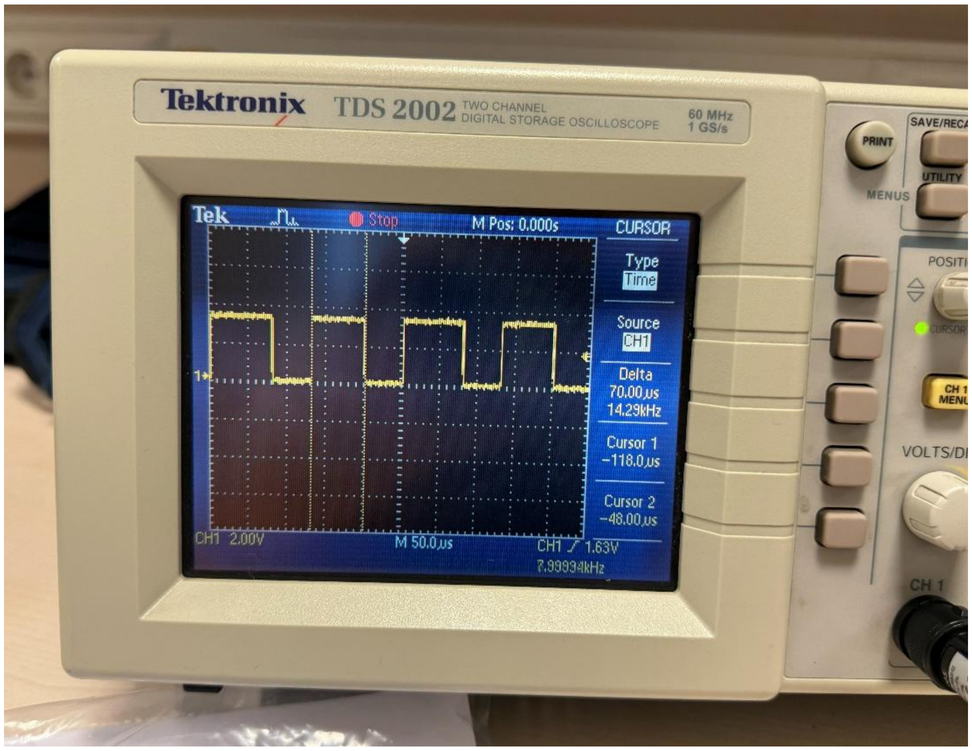


Figure 7: 70 µs Logic High Wave on Oscilloscope

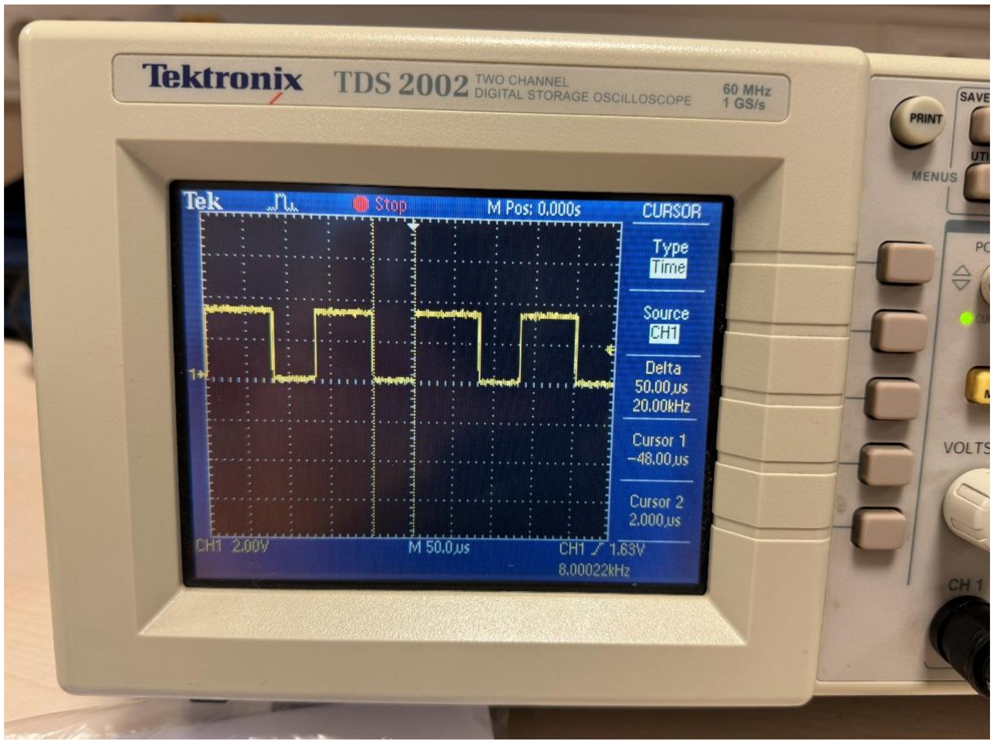


Figure 8: 50 µs Logic Low Wave on Oscilloscope

Finally, the testbench results and the oscilloscope results are matched perfectly, which means that the waveform is generated with the correct time intervals in Basys3. No unexpected delays were observed and each segment of the waveform behaved exactly as designed. The results verify that the implementation if fully functional.

Conclusion

In this lab, an arbitrary digital waveform generator was successfully designed and implemented using a clean 50 MHz clock produced by the Clocking Wizard IP. This lab was very helpful for me to learn how to generate an arbitrary waveform, which will also help me in my term project. I learned how to use the Clocking Wizard IP in Vivado. The process of verifying the waveform through both simulation and oscilloscope measurements allowed me to clearly see the relationship between VHDL design, clocking and real hardware behavior. Overall, this lab strengthened my understanding of synchronous digital design and demonstrated the importance of precise timing control when implementing the design to Basys3. Although I encountered many timing-related issues during the design process, resolving them improved my understanding of clocking, synchronization and precise waveform generation.

Appendices

arbitrary_waveform.vhd

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

entity Arbitrary_Waveform is
   Port (
      clk_in   : in STD_LOGIC;
      reset_in : in STD_LOGIC;
      wave_out : out STD_LOGIC
   );
end Arbitrary_Waveform;
```

```
architecture Behavioral of Arbitrary_Waveform is
    component clk_wiz_3
    port
     (
      clk_in1  : in  std_logic;
      clk_out1 : out std_logic;
      reset    : in  std_logic;
      locked   : out std_logic
     );
    end component;
    signal s_clk_50MHz : std_logic;
    signal s_locked    : std_logic;
    signal s_locked_sync_reg1 : std_logic := '0';
    signal r_locked_50MHz     : std_logic := '0';

    type state_type is (S_HIGH_80, S_LOW_50_1, S_HIGH_70, S_LOW_50_2);
    signal current_state : state_type := S_HIGH_80;
    signal counter : integer range 0 to 4000 := 0;
    constant C_80US : integer := 4000;
    constant C_50US : integer := 2500;
    constant C_70US : integer := 3500;
begin
    clk_instance : clk_wiz_3
    port map (
      clk_in1  => clk_in,
      clk_out1 => s_clk_50MHz,
      reset    => reset_in,
      locked   => s_locked
    );
    process(s_clk_50MHz)
```

```
begin
    if rising_edge(s_clk_50MHz) then
        s_locked_sync_reg1 <= s_locked;
        r_locked_50MHz     <= s_locked_sync_reg1;
    end if;
end process;
process(s_clk_50MHz)
begin
    if rising_edge(s_clk_50MHz) then
        if reset_in = '1' or r_locked_50MHz = '0' then
            current_state <= S_HIGH_80;
            counter <= 0;
            wave_out <= '0';
        else
            case current_state is
                when S_HIGH_80 =>
                    wave_out <= '1';
                    if counter < C_80US - 1 then
                        counter <= counter + 1;
                    else
                        counter <= 0;
                        current_state <= S_LOW_50_1;
                    end if;

                when S_LOW_50_1 =>
                    wave_out <= '0';
                    if counter < C_50US - 1 then
                        counter <= counter + 1;
                    else
                        counter <= 0;
```

```
                            current_state <= S_HIGH_70;
                        end if;

                    when S_HIGH_70 =>
                        wave_out <= '1';
                        if counter < C_70US - 1 then
                            counter <= counter + 1;
                        else
                            counter <= 0;
                            current_state <= S_LOW_50_2;
                        end if;

                    when S_LOW_50_2 =>
                        wave_out <= '0';
                        if counter < C_50US - 1 then
                            counter <= counter + 1;
                        else
                            counter <= 0;
                            current_state <= S_HIGH_80;
                        end if;
                end case;
            end if;
        end if;
    end process;
end architecture Behavioral;
```

constraint.vhd

```
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 U18 [get_ports reset_in]
set_property IOSTANDARD LVCMOS33 [get_ports reset_in]
set_property PACKAGE_PIN J1 [get_ports wave_out]
set_property IOSTANDARD LVCMOS33 [get_ports wave_out]
```

arbitrary_waveform_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Arbitrary_Waveform_tb is
end Arbitrary_Waveform_tb;
architecture Behavioral of Arbitrary_Waveform_tb is
  component Arbitrary_Waveform
  port(
     clk_in   : in  std_logic;
     reset_in : in  std_logic;
     wave_out : out std_logic
    );
  end component
  signal tb_clk_in   : std_logic := '0';
  signal tb_reset_in : std_logic := '0';
  signal tb_wave_out : std_logic;
  constant C_CLK_PERIOD : time := 10 ns;
begin

  uut: Arbitrary_Waveform PORT MAP (
      clk_in   => tb_clk_in,
            reset_in => tb_reset_in,
            wave_out => tb_wave_out
          );
    clk_process :process
    begin
        tb_clk_in <= '0';
        wait for C_CLK_PERIOD/2;
        tb_clk_in <= '1';
        wait for C_CLK_PERIOD/2;
    end process;
    stim_proc: process
    begin
        tb_reset_in <= '1';
        wait for 100 ns;
        tb_reset_in <= '0';
        wait for 1000 us;
        wait;
    end process;
end Behavioral;
```

References

https://digilent.com/reference/programmable-logic/basys-3/reference-manual?srsltid=AfmBOopQ9GSjm2wPIWEts5LwB21y_mydC9ZHSzPgHWcgh_E9Jw4Z7w8G

https://www.realdigital.org/doc/ae6ce4fbf81065307776fd9d0911ec7d

https://www.instructables.com/Digital-Clock-in-VHDL/