# Lab Report 6: Arbitrary Waveform Generator EE 102 Section 02 Bilkent University

April 3rd, 2024
Doğa Zeynep Tarman
22303157

## Purpose of the Experiment

The purpose of this laboratory task was to design an waveform generator of our choice on our FPGA boards and observe the results using an oscilloscope.

## Methodology

For this laboratory, I first started to examine the lab document which was sent to us carefully. From what I could understand, we were asked to write a VHDL code which would generate some random waveform (we could also do more than one) using the clocking wizard that was available in the program that we are using. To accomplish this, I decided to first decide on what I would define as ”arbitrary”. Following this, I started to try and write a code in VHDL which would be able to implement the idea that I had in mind. I also wanted to learn something new, so I decided that it would be a nice idea to try doing something like a servo motor. After everything was complete, I connected my Basys 3 to the oscilloscope and observed the results. All the code that I have written (except for the constraints) can be found in the Appendix part of this report.

## Design Specifications

Since I had to connect my Basys 3 to an external component, an oscilloscope, I had to make sure that my inputs and outputs were set correctly. For this, I used a special part of my Basys 3 which we have not used in the previous laboratories before, the part named ”JB”. I connected the GND to the crocodile clip of my oscilloscope, and the part named ”1” to the tip of the probe. I then obviously connected the probe to the oscilloscope. By following these steps, I was able to observe the waves on the oscilloscope, which I will describe in more detail in the Results section of my report.

## Results

This laboratory mainly focused on our ability of writing the necessary VHDL code to generate the arbitrary waveform with using the clocking wizard. After feeling confident that I wrote the code correctly, I ran a simulation and got a waveform as it can be seen in Figure 1 and Figure 2.

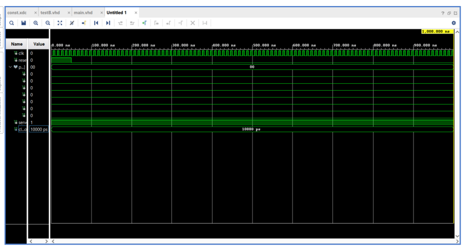

Figure 1: The Whole Waveform

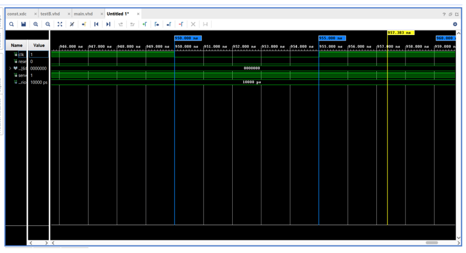

Figure 2: A Part of the Waveform

To see these results more concretely, I proceeded to connect my Basys 3 to the oscilloscope using a probe and saw how the waves were. I observed that there were ”spikes” of these signals, which were a little parasite-heavy as it can be seen in the following figure, Figure 3.

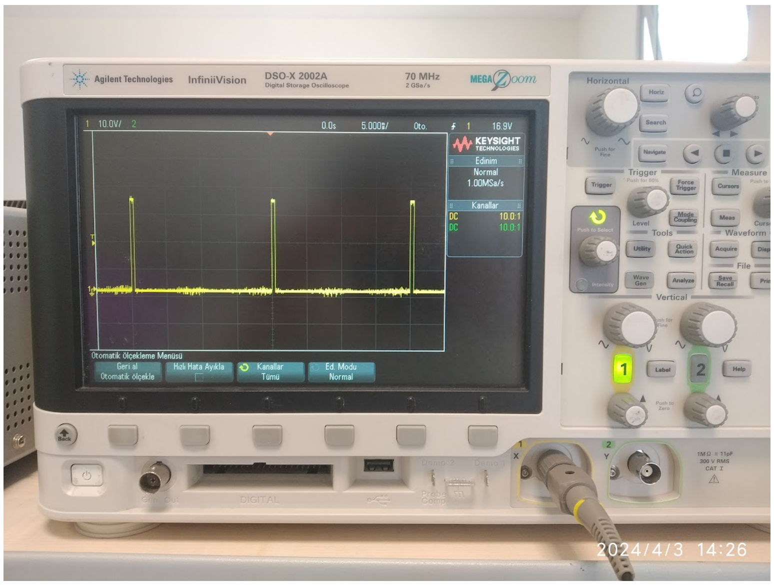

Figure 3: The Waves Observed in the Oscilloscope

## Conclusions

During this laboratory, I tried to write a code for an arbitrary waveform generator which used the clocking wizard in VHDL. Out of all the laboratories that I have completed up until now, this was one of the hardest one by far. On top of the difficulty of the laboratory, I made several mistakes trying to connect the oscilloscope to my Basys 3. Starting with the difficulties that I have faced during the code-writing part, although I have avoided mentioning this in the previous lab reports, I faced problems with the variable names. I realized that randomly, Vivado decides that a certain name is not ”valid”, and makes me face many critical errors. I made sure that the variable name was not one of the reserved words or anything every time, but that was not the case. Sometimes even erasing the word and writing the exact same word again resolved the issue. However, in this laboratory, as I was already having trouble with the code itself this sort of a bug made things even more difficult. Moreover, when I tried to connect my Basys 3 to the oscilloscope, I did not know what to do at first. Even though I defined the outputs beforehand using the built-in I/O Mapping feature of Vivado, connecting them was a huge trouble for me. I then proceeded to connect the wrong outputs to the oscilloscope and accidentally caused sparks on the Basys 3. I consider myself quite lucky as my Basys 3 is (hopefully) still unharmed and it didn’t blow up or anything. Overall, this laboratory was interesting an equally challenging for me. I believe that it was beneficial as we will be presenting our project demos in around two weeks from now.

## Appendix

main (main.vhd)

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

entity clock64 is
    Port(
        clk    : in  STD_LOGIC;
        reset  : in  STD_LOGIC;
        clk_out: out STD_LOGIC
    );
end clock64;

architecture Behavioral of clock64 is
    signal temporal: STD_LOGIC;
    signal counter : integer range 0 to 780 := 0;
begin
    freqDiv: process (reset, clk) begin
        if (reset = '1') then
            temporal <= '0';
            counter  <= 0;
        elsif rising_edge(clk) then
            if (counter = 780) then
                temporal <= NOT(temporal);
                counter  <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    clk_out <= temporal;
end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity servo_pwm is
    PORT (
        clk   : IN  STD_LOGIC;
        reset : IN  STD_LOGIC;
        pos   : IN  STD_LOGIC_VECTOR(6 downto 0);
        servo : OUT STD_LOGIC
    );
end servo_pwm;

architecture Behavioral of servo_pwm is
    signal cnt : unsigned(10 downto 0);
    signal pwmi: unsigned(7 downto 0);
begin
    pwmi <= unsigned('0' & pos) + 32;
    counter: process (reset, clk) begin
        if (reset = '1') then
            cnt <= (others => '0');
        elsif rising_edge(clk) then
            if (cnt = 1279) then
                cnt <= (others => '0');
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end process;

    servo <= '1' when (cnt < pwmi) else '0';
end Behavioral;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sClock64 is
    PORT(
        clk  : IN  STD_LOGIC;
        reset: IN  STD_LOGIC;
        pos  : IN  STD_LOGIC_VECTOR(6 downto 0);
        servo: OUT STD_LOGIC
    );
end sClock64;

architecture Behavioral of sClock64 is
    COMPONENT clock64
        PORT(
            clk    : in  STD_LOGIC;
            reset  : in  STD_LOGIC;
            clk_out: out STD_LOGIC
        );
    END COMPONENT;

    COMPONENT servo_pwm
        PORT (
            clk   : IN  STD_LOGIC;
            reset : IN  STD_LOGIC;
            pos   : IN  STD_LOGIC_VECTOR(6 downto 0);
            servo : OUT STD_LOGIC
        );
    END COMPONENT;

    signal clk_out : STD_LOGIC := '0';
begin
    clk64kHz_map: clock64 PORT MAP(
        clk, reset, clk_out
    );

    servo_pwm_map: servo_pwm PORT MAP(
        clk_out, reset, pos, servo
    );
end Behavioral;
```

Test Bench (testB.vhd)

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

entity sClock64TB is
end sClock64TB;

architecture Behavior of sClock64TB is
    component sClock64
        Port(
            clk   : IN  STD_LOGIC;
            reset : IN  STD_LOGIC;
            pos   : IN  std_logic_vector(6 downto 0);
            servo : OUT STD_LOGIC);
    end component;
    -- In
    signal clk   : STD_LOGIC := '0';
    signal reset: STD_LOGIC := '0';
    signal pos   : std_logic_vector(6 downto 0) := (others => '0');
    -- Out
    signal servo : STD_LOGIC;
    constant clk_period : time := 10 ns;
begin
    UUT: sClock64 PORT MAP (
        clk => clk,
        reset => reset,
        pos => pos,
        servo => servo
    );

   clk_process :process begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
   end process;

    stimuli: process begin
        reset <= '1';
        wait for 50 ns;
        reset <= '0';
        wait for 50 ns;
        pos <= "0000000";
        wait for 20 ms;
        pos <= "0101000";
        wait for 20 ms;
        pos <= "1010000";
        wait for 20 ms;
        pos <= "1111000";
        wait for 20 ms;
        pos <= "1111111";
        wait;
    end process;
end;
```